In this lecture you will understand:

* Current software scenario
* Why should we use Java
* Different Java Technologies
* Java based software development
Software Scenario

The slide shows different broad categories of software that exist today. For example, we can build desktop applications today i.e. applications which work on a desktop machine which may or may not need an Internet connection are desktop applications. Typically a Grocery shop has an Inventory application or there can be simple payroll processing system or there can be a simple admission system where people join courses and then the office tries to take admission for the courses. That can be a simple desktop application. Then we can have Web applications, application where we are required to book airline ticket over the Internet or we are required to gather some weather information or we are required to find out the road or the map for a particular destination. These are web-based applications. Then we can have Embedded applications where we write programs for Embedded devices like Microwave oven or a washing machine or a mobile phone. Next is a Distributed application. A distributed application is required to be built in scenarios where we have a multiple branches of a bank. A transaction that happens in one branch of a bank, if another branch of the same bank wants to have an access to that transaction data then a distributed application is to be build. We know Anywhere banking, means if we are a account holder then we can be anywhere in the world to be able to operate that particular account. If we have an account in a bank at Nagpur, we can as well be in Mumbai, we can instruct the bank over the Internet to issue a Demand Draft in favor of some particular party. Means we do not have to be geographically present where the bank branch is, we can be really anywhere but still be able to operate that account easily. To facilitate working of such application, we need to write complicated programs and those programs are distributed applications. We also have Operating System software which actually makes Windows work or Linux work or Unix work. There is an operating system called BSD. The different operating systems are nothing but programs. To build these programs we have to employ language. Even in the mobile phone, for example, in a Nokia mobile phone there is a operating system which is running whose name is Symbian Operating System. We also have compiler software which converts programs in C language, C++ language, Java language or any high-level languages into a form that the machine understands. The machine understands machine language. So, the compiler will pick up the program that we write and convert it into a form which the machine understands. Compiler compiles the program means which takes the instruction given by the program in a particular language and converts it into machine language because machine understands only machine language. Then we have Device Driver’s program. If we purchase a new hardware to attach to the PC, the moment we attach it to the PC we are also required to install the Device Driver program which will drive this device. For example, we purchased a Laser Printer then along with the Laser Printer we also get a CD-ROM which contains the Device Driver program which knows how to drive the Laser Printer. So for any new hardware, we are required to install a Device Driver program. Software world had become complex. There are variety of programs that are required to be build. We have to build Device Driver program, Operating System, Compiler. We have to create complicated distributed applications. Device Drivers, Operating Systems and Compiler are almost exclusively written in C or C++. Device Drivers most of the time are written in C language. With Device Drivers, Operating Systems and Compiler, performance or speed has extreme importance. Suppose there is a driver which drives the screen which is capable of displaying different characters, images and graphics on the screen. If the performance of that driver is poor then every application running on the machine is going to suffer. So for performance reason, these Device Drivers are written in C language. On the other hand, there are Desktop applications, Embedded application, Web applications and Distributed applications which can be build using any of the languages like Java, VB, C# or VC++. In the slide, at the left hand side we have system software i.e. Device Drivers, Compilers and OS and on the right hand side we have application software i.e. Desktop applications, Embedded application, Web applications and Distributed applications together are known as application software. Java has role to play only in the application software and not in system software.

Why Java?
It is extremely easy to develop applications in Java because there are very competent tools present for java programming which helps to locate mistake in the program as we start typing. This can be experienced when we use applications like MS-Word, Power Point, Excel. In all these, the moment we type a word, these applications are able to check and report to us whether we had committed a mistake in the spelling or not. If the words are misspelled, they are underlined by red. Once the sentence is completed, if there is any grammatical mistake, it is pointed and underlined by green in color. Same kind of help can get while writing a program. This is possible in tools that are present in Java language while development. If we try to call function fun( ), then the arguments that are required to pass that function those are shown immediately through a feature intellisense. Not only as the program is typed we are pointed out where we had committed a mistake, but also if a call to different functions is made then the arguments, parameters we pass to the function if shown right there as we typed the program. Java is rich in examples and sample programs. The entire documentation is available made by Sun at http://java.sun.com/docs/. Java promotes Rapid Application Development. Because of the constant pressure of time, it has become extremely important that whichever programming language we use, we should build programs using that language rapidly. If rapidly we are supposed to build an application then the tools and the language that we are proposing to build this software should be mature enough to support the Rapid Application Development. Java is mature in this. Java lets allows to build applications quickly. If we want to build a media player or a Embedded Web Browser or a Embedded Excel Browser then the Rapid Application Development tools that come with Java, help in building these applications quickly. Java is the language which can be used along with the same tool for building variety of applications whether it’s a Desktop, Web, Distributed or an Embedded application. For building any of these applications, we do not have to learn a new language. Imagine a scenario where a person is working in a company which is building desktop applications. After a few years passed, the person has gathered enough experience in building desktop applications. If somebody comes and says we are now going to build distributed applications. Only we have to learn a language that is going to build the distributed applications. The programmer does not like it. The programmer wants that the language that he knows should be used in building variety of applications. Just the nature of the application change, we should not be made to learn a new language. Language is a means to an end. If it is a means to an end then just because the end change the means need not change. For example, a carpenter who makes chair, table, cupboard, etc. or whatever, the tools he uses are more or less same. Just because the end application changes, we should not be learn a new language. With the language that we have, we should be able to use that to build any application. Java supports this. It is very easy to manage large projects, projects which are to the tune of 10,000 or more lines of code. For building large applications, the code we are writing is tremendous. If we want to handle so much code then the language should be competent in doing so. With Java, we would be able to manage big projects. To facilitate all these, there are source control system mechanism, good quality debugging tools mechanism and ways to organize the different files present in the project. Because in big program, we are going to have multiple files. If lots of files are there then the tool that we are using to build that application should be competent enough to properly organize these files into different directories of the project. All database related files can go into a Data folder, all application related programs can go into an Application folder, all resources can go into another folder. All the management of the project becomes crucial if the application is big. Debuggers are available to eliminate errors that occur in the program and when we build a big project, there is a system for the development of large projects. So source control system comes. Software development process is a on going process. The source control system available in Java is SVN.

Java Technologies

Java technologies can be categorized into three broad categories i.e. Java 2 Standard Edition, Java 2 Mobile Edition and Java 2 Enterprise Edition.

J2SE Track

J2SE track is used for building desktop applications. Today in desktop application, we do need at times connectivity not only with the network but also with Internet. Example of this is, an AVG Antiviral software. This software needs updation of its signature. So we schedule an update and the
software updates itself. There are situations that the application might need to connect to the Internet
may be only for a small period of time but that might be required. So the application is capable of
achieving this. In the internet, we have a Google toolbar or a Yahoo toolbar which is available in the
form of a plug-in. Internet application is basically a Desktop application which renders any HTML
file that we get from any of the web pages. But there we have to have a provision such that people
can plug-in their interfaces into the Internet Explorer. We expect that the desktop programs to be
functionally rich. For example, we expect every single desktop software to provide a help. There are
provisions to search. For example, in Internet Explorer we download a page; we can do Ctrl + F and
try to search for a particular term on that page. We are intermittently required to use Internet to
update the software. So we can do updation, get help and search using Internet Explorer. J2SE
permits to achieve every task.

J2ME Track

In J2ME track, we write programs for embedded devices. Embedded devices have a very small
screen and there are variety of mobile phones available with different screen geometry. There are
different user interfaces. Some mobile phones uses stick to make selection. We can have button based
interface. Smart phones have stick based interface. We also expect intermittent connectivity from a
mobile phone. We are not permanently connected to the internet. But of we want to send mail to
somebody, s smart phone have capability of let us send the mail. Between two mobile phones, if we
want to exchange the music files then we can do so using Bluetooth. We might be required to
synchronize the mobile phone with a PC. If we have address book on the PC, we can take the
addresses on the mobile phone or vice versa. Synchronization is very commonly required. All these
the J2ME track allows to build in the applications.

J2EE Track

J2EE applications are Internet based. For example, portals. Portals like Yahoo where we get
information about variety of things. We can get information about stock, weather, airline delays and
get the information about everything. If we have a education portal then may be on that education
portal, there are ways to do newer courses, provision to do some certification, provision to download
certain tutorials, provision to read ebooks, provision to test the skills in a particular language. Overall
echo system of that area can be build into a portal. Themes can be used in different websites which
can be personalize accordingly we want. Modules can be downloaded into the website. For example,
a calendar module can be present in a website. There might be a news module, reminder modules,
weather modules, stock modules. All can be plug into the website as and when needed. To carry out
all we need a mature platform and J2EE provides that so that we can build modern application for a
modern age.

Java Based Software Development

We need following things for Java based software development.

- We need Integrated Development Environment to which we can type program, compile
  program, debug program, get help, get online help. So we need a mature IDE for building
  Java Based Software.

- We need Java Development Kit which contains a compiler, linker, debugger, etc. Java
  Development Kit is known as JDK.

- We need JRE i.e. Java Runtime Environment.

So all these we need for any Java based software development. There are variety of IDEs that we
can choose. Microsoft provides VJ++ as part of Visual Studio.Net software. From Borland, we
can have software development tool as JBuilder. Symantic provides Visual Café. From Sun
Microsystems, we can have NetBeans. NetBeans is a RAD tool. It generates a skeleton code
every time we start a program. We do not have to code everything from scratch. Some readymade
code we get and then we can make some additions, deletions to that code to get the work done.
We can have debugging help if we commit mistakes. NetBeans is a Industry standard IDE. When
we download and install on machine then we get a icon as shown in the slide. JDK and JRE is
available from java.sun.com. JDK 6.0 is downloaded and when we install it, it is get installed in
c:\Program Files\Java\jdk1.6.0 folder.
In this lecture you will understand:

* Learning steps
* Constants and Variables
* Types of constants and Variables
* How to identify types
* Rules for creating variables
* Keywords
In The Beginning...

Learning Java language is similar to other languages like English. While learning English, everyone begins with alphabets and digits. After that we need to know, how to combine these alphabets to form words and how to combine digits to form different numbers. The next step is to combine the words and numbers to form sentences. After that combine the sentences to form paragraph. Once we know the paragraph, we can form more paragraphs and continue. To learn Java, we have to begin with alphabets and digits. But we also need to know the special symbols like *, /, \, ,, +, -, etc. Once we know the alphabets, digits and special symbols, we need to combine them to form constants, variables and keywords. Once we know how to make use of constants, variables and keywords, we combine them to form statements or instructions. Collection of the instructions builds a program. When we know how to build program, we can go on building more and more programs. This is the whole process of learning the language.

Alphabets, Digits, Special Symbols

Java language permits to use all alphabets irrespective of they are capital letters i.e. A – Z or small case letters i.e. a - z. We are also allowed to make use of digits ranging from 0 – 9. On the keyboard there are 32 special symbols. Everyone special symbol can be used in Java.

Constants & Variables

These alphabets, digits and special symbols are combined to form constants and variables. For example, in the expression 3x + 2y, 3 and 2 are constants as their values do not change. The thing which is fixed and unchangeable is a constant whereas x and y can take any values. So they are known as variables. In Java terminology, these constants and variables are also known as literals and identifiers respectively. If we write instructions like x = 3, y = 5, x = 7, etc. as shown in the slide, then these x, y, z, w are known as variables whereas 3, 5, 7, 3.14 and ‘A’ are known as constants. In these set of instructions, 3, 5 and 7 are integer constants, 3.14 is a real constant and ‘A’ is a character constant.

Integer Constants

The variety of integer constants we can construct. For example, 421, -62, +45 or 4096. While creating these integer constants we have to observe certain rules. For example, within a integer constant we are never supposed to have a decimal point. Hence, 72 and 72.0 value wise might appear same but from Java point of view 72.0 is an illegal integer constant because of the decimal point. The integer constant may be either positive or negative. The default sign is positive. Within integer constant, there should never be a comma and a space. Hence if we write 32,500 which makes 32500 readable but it is not valid in Java. When comma is given 32 becomes a different constant and 500 becomes another constant. If we want to write 473, do not give space between 4 and 7 and a space between 7 and 3 to make the number readable. If we give space between them, 4, 7 and 3 becomes another constant. The valid range of integer constants is -2147483648 to +2147483647.

Real Constants

While creating real constants, we need observe the following rules.

- In a real constant, it is understood that there would certainly be a decimal point present.
- The real constant can be positive or negative. The default real constant is positive.
- There should not be a comma or space within the real constant.
- Valid range of real constants is -3.4 x 10^{18} to +3.4 x 10^{18}. 

•
Real constants can take different forms. For example, 427.62, +24.295 and -0.00254 can be expressed in the form 4.2762E2, 2.4295e1 and -2.54e-3 respectively. The first form is known as fractional form whereas second form is known as exponential form. We can use both the forms as Java language understands both of them. While using exponential form, whatever is to the left of E is called mantissa and whatever to the right of E is called exponent. E can be in small case or capital case. If we want to use population of the world, then the range that the integer constant offers may not support. At that time we can use real constants.

Character Constants

‘A’, ‘m’, ‘3’ or ‘+’ are the examples of character constants. When a character, digit or a special symbol is written within a pair of single inverted commas form a character constant. The character that can be written within single inverted commas can be a capital letter, a small case letter, a digit or a special symbol. For example, as shown in the slide Nagpur is not a character constant. The first example is also wrong because the single quote marked with circle that we have to use but Z uses reverse single quotes. Both the quotes have to slant to the left.

Java Data Types - Primitives

The different data types present in Java are as follows. char can range from 0 to 65535 which occupies 2-bytes in memory. byte can take any value from -128 to +127 which occupies 1-byte in memory. short integer can take any value from -32768 to +32767 which occupies 2-bytes in memory. int can take a value which falls in the range -2147483648 to +2147483647 and occupies 4-bytes in memory. long offers much bigger range. This can take care of the population of the entire world which occupies 8-bytes in memory. More the number of bytes occupied in memory, bigger is the range. float is -3.4e38 to +3.4e38 which occupies 4-bytes in memory. double can take any value from -1.7e308 to +1.7e308 and occupies 8-bytes in memory. boolean take only 1-bit in memory and take value like true / false. All these data types together are known as primitives. The sizes of the data types are always fixed but not so fixed. Because the char, float, int, etc. determines the behavior but not the size. If we have used short in a particular implementation of Java, we might find that there the short might actually occupy 4-bytes. From the sake of convenience of the computing the architecture, it might occupy 4-bytes but the range it offers is -32768 to +32767. This behavior is guaranteed the range will be -32768 to +32767 but the bytes that will be occupied is 2-bytes through out the different computing platforms with Java is being used. In that case, the size is not fixed but the behavior is fixed. The type certainly determines the behavior not the size of a particular data type.

Variables

As many types of variables exists as the number of types of constants. It is because, every single computing language follows the basic rule i.e. the particular type of constant can be stored only in the same type of variable. If there are 20 types of constants then there are 20 types of variables in that language. This is true for all languages that as many types of constants are supported those many types of variables are supported. It is necessary to identify the type of variables and constants. If we say that there is a variable y in which we want to store constant 4 then 4 indicates that it is an integer constant whereas by looking at y it never indicates its type. We cannot say that y is an integer variable all the time because if we store 1.5 in y then it would become a real variable. If we store 4 in y then it clearly indicates that we can do any arithmetic on y whereas if we say y = ‘A’ then on ‘A’ we cannot do any addition, multiplication, division, etc. It means unless and until type of a variable is identified what operation should be permitted and what operation should not be permitted on that variable that does not get decided. Hence, it is necessary to be able to identify the type of the variable because once type is identified then what operations to be permitted and what cannot be permitted that can be decided.

What Happens in Memory...
Computer memory consists of millions of cells. For example, we do $x = 3$, out of the several locations in memory one of the locations get chosen and once 3 gets stored at that location a name $x$ is given to that location. Similarly if we do $y = 4$, then some other location is chosen and a value 4 would get stored at that location and name $y$ is given to that location. If we say $z = x + y$ then we are not going to add alphabet $x$ with alphabet $y$. we are trying to add the value that is stored under the location name $x$ with the value that is stored under the location name $y$. So $3 + 4$ is 7. The 7 will get assigned to $z$ means some location is chosen, value 7 would get stored in it and the name $z$ is given to that location. If we say print $z$ which means that print the value that is stored under the location name $z$ and not the alphabet $z$. If we now say $x = 5$, 5 is get written at a location name $x$. But as shown in the slide, at location name $x$, there is already a value 3. When we say $x = 5$, the value that is stored at location name $x$ is now overwritten with 5. The moment the value 3 is get overwritten by 5 at that time we come to know what the word variable means. Something which has the ability to vary is a variable. So variable is an entity whose value can change. Variable is a name given to the particular location in memory.

**How To Identify Types**

As shown in the slide, 3 is an integer constant, 3.0 is a real constant whereas ‘3’ is a character constant. The type of $a$, $b$ and $c$ cannot be identified. Looking at $a$, $b$ and $c$, we cannot determine their types. We cannot say that $a$ is a real variable because if we store 3 in $a$ then it will actually become an integer variable. Looking at the constants we can easily identify the type of constants but when we look at variable names we will never be able to identify just by looking at their names. Hence, every single language has some or the other mechanisms which will help to identify the types of the variables. In Java language, we have to identify the type by indicating the type of the variable right at the beginning. We can do that by saying int $a$. When we do so, it means $a$ is going to be an integer type of variable capable of holding an integer constant. On similar lines, we can say float $b$ and char $c$. based on what type of constant we are proposing to store in a variable, we have to appropriately define the type of the variable. int, float and char are the data types or primitives, $a$, $b$, $c$ are variables whereas 3, 3.0, ‘3’ are constants or literals. When we want to give different names to locations in memory, we can give different names other than $a$, $b$, $c$ or $x$, $y$, $z$ to the locations. For example, we can give a variable name like $xyz$. We also can give the name of the location in memory as $abc123$. However, if we attempt to give the name of the location in memory as 123$abc$ then it is not allowed.

**Rules for Building Var. Names**

Rules for building variable names are as follows.

- A variable name cannot ever begin with a digit.
- Rest of the characters can be alphabets, digits or underscores. A variable name can begin with alphabet and rest can be alphabets, digits or underscores. For example, pop98 is a legal variable name as it begins with alphabet p and rest are alphabets like o, p and digits as 9, 8. si_int is a legal variable name as it begins with alphabet and rest are alphabets or underscore.
- Variable name can be of any length i.e. pop98 has 5 characters whereas si_int has 6 characters.
- There should not be a comma or a space within the variable name.
- The variables names in Java are case sensitive. For example, abc and ABC are two different variables.

**Java Keywords**

There are as many as 48 keywords available in Java. int, float and char are the keywords which we had seen. The keywords are the words whose meaning already stands explained to the machine. For
example, if we say int z, it means we do not have to explain the further meaning of what is int as meaning of int already stands explained. So when we say int z, it means we are trying to create a variable whose type is int. We do not have to further explain what is int z. Reserved word are similar to keywords. If we try to create some statements like

```java
    integer a
    real b
    character c
```

none of them will work because meaning of integer, real and character is not explained to the machine. We should say

```java
    int a
    float b
    char c
```

to make it work. If we say int float and then float = 3 then it will not work because we are trying to use float as a variable name. If we try to treat char as a variable name then again it will not work. As float and char both are the keywords and we cannot use them as variable names. While creating variable names avoid the keywords completely.
In this lecture you will understand:

* First Java program
* Printing values
* General form of println( )
* Function, Class and Package
* Type of comments
Where Do We Stand?

We begin with alphabets, digits and special symbols. We learned to how to combine these to form constants, variables and keywords. We learnt how to identify the constant types and variable types. We made use of keywords like int, float, char etc. to declare the variables of a particular type. While building these variables we need to follow a certain set of rules. It’s time to build these constants, variables and keywords together to create statements or instructions and once we know how to build these statements or instructions then we would see how they are to be combined to create a program. So we are done with first two steps and the next logical step would be to learn how to construct statements or instructions. But we would straightway begin with a program. Once we are done with the programs we will try to comeback and find out what statements or instructions knowingly or unknowingly we made use of while writing these programs. So straightway begin with our first java program.

The First Java Program

We would say p = 1000.50f which indicates that it’s a float number. Any time we have to refer to a real number or a float number we have to add f at the end of the number. If we do not do so in that case the number is treated as a double. Double is another type of a real number stands for a double precision number. Right now we use a float integer indicates that 1000.50 is a float we have to add a suffix f at the end of 1000.50. We store 1000.50 in a variable called p. Then we say n = 3 and r = 15.5f. Once again 15.5 is a float hence we are adding f at the end. So we have set up p, n and r. Now we would say si = p * n * r / 100, we are trying to get simple interest calculated. Now once we used variables we have to define them. We need to identify the type of the variable because unless and until the type gets identified what operations are to be permitted and what are not to be permitted on that variable that does not get decided. Here in p we are proposing to store a floating point number or a floating point constant or a floating point literal. Hence p has to be a float variable. r and si would also qualify as float variables because in r we are storing 15.5f. si would also be float because when we do p * n * r / 100, the answer of simple interest would be in some rupees and some paise, basically a float number. So we can say float r and float si. If we do not want that then we are also allowed to say float p, r, si. Means instead of writing three float statements to declare p, r and si respectively we are free to combine them into one single line and write like float p, r, si. On similar lines we also define variable n i.e. int n. So we are defined variables, we are initialized them and then we have used them to carry out the multiplication and division operations, obtain the values and store it in si. Few tips to understand are as follows.

- Every single variable that we proposed to use in the program will have to be declared. No variable can ever go undeclared. If the program has 200 variables then we have to define everyone of them. If we forget to define any of the variables, the compiler will immediately tells that these variables we had forgotten, so define them and go ahead.

- Secondly +, -, *, and / that are used here are known as arithmetic operators. Out of these four, we had used only * and / standing for multiplication and division. You might say that why we are using this * operator for multiplication why not p x n x r / 100. If we use x, then it will get confused with x. So a creator of Java language a man called James Gosling, decided that x did not be used, we use * to represent multiplication because it looks similar to a cross. Then / is used for a division. Clearly differentiate between a slash and back slash. Do not try to do 2 \ 3, do not try to do the line in between 2 and 3 and also do not try 2 ÷ 3. None of these are correct. If first two are used the compiler might report an error. In the third, it is not present on the keyboard. Point to note is that make use of * operator for the multiplication and for division always use a slash. So that completes calculation of simple interest.

Printing Values…

In memory, any time we create variables locations are reserved in memory. In one location value 1000.50 will be stored and the name p will be given to that location. Similarly 3 will be stored in another location, name n will be given to that and lastly 15.50 will stored in some location in memory and name r will be
given to that location. Now when \( \text{si} = \frac{\text{p} \times \text{n} \times \text{r}}{100} \) is done, we are not trying to multiply alphabets but we are trying to multiply the values stored in \( \text{p} \) with values stored in \( \text{n} \) with value stored in \( \text{r} \), whatever is the result is divided by 100. Whatever final result obtained we are trying to store it in another location name \( \text{si} \). Assuming that 1000.50 * 3 * 15.50 turns out to be 465.25, will get stored in a location and the name \( \text{si} \) will be given to it. This is not enough because ultimately the value that is stored in a memory location we cannot see that value with a naked eye. So there is a mechanism which will pick the values present in the location named \( \text{si} \) and put it at the place where we can see it i.e. the screen. So we need some agent which will pick value from \( \text{si} \) and put it on the screen and that agent is a function called `println( )` which is always followed by a pair of parenthesis inside which we are supposed to write the name of the variable whose values we intend to print on the screen. Our interest lies in printing the values of \( \text{si} \). Hence we are saying `println( \text{si} )`. Before `println( )` we have to say `System.out.`, and then `println( )`. Out of these `System.out.println( )` as we said, `println( )` is a function, out that we are prepending before `println( )` is an object and `System` is a class. In `println( )`, `ln` stands for line. When we say `println( \text{si} )` means print the value of \( \text{si} \) on the screen and then positioned the cursor at the beginning of the next line. Positioning cursor on the next line is achieved in C and C++ using `'\n'`. Likewise in Java if we want to position the cursor at the beginning of the next line we have to say `println( )`, cursor goes to the next line. So here value of \( \text{si} \) will be printed in one line and then cursor will be positioned at the beginning of the next line such that if we decide to print out any more values they are bound to come in subsequent lines. Whenever we have to print a value we have to say `System.out.println( )`.

### Statement Terminators

Suppose there are two sentences one is I am a boy and another is I go to school. The first statement does not end at boy because there is no full stop at the end. Likewise if we say p, r, si and in the next line we write `int n`, just because we wrote `int n` in the next line it does not mean that first sentence has ended. The way just because we write I go to school in the next line does not mean that the first sentence has ended. In English language any time when we have to end the sentence, we always use a full stop. In Java instead of full stop `;` (semicolon) is used. Similarly, we say `int n;`. The `;` acts as a statement terminator. It is known as a statement terminator because it indicates where a particular Java statement is ending. Now if we say `float p, r, si;` then we say `int n;`. We can also declare it as `float p, r, si;` and `int n;` both in the same line. We are allowed to write multiple statements in one single line so long as we terminate each statement properly with a `;`. In English, we write a sentence, put a full stop and then begin the next sentence in the same line. Same is true with Java. We type a sentence, give a `;`, type the next statement may be in the same line. So we are indeed allowed to write a statements in same line provided we separate them properly using a statement terminator or using a `;`. Java is a free form language. Free form language means it does not matter in what form we type it. If there are five statements in a program, we type those five statements in one line or five lines or 2 lines or 4 lines that is entirely our choice. Means a Java program can take any form which we think is best. So it’s a free form language.

### What To Execute

It has been suggested by John Gosling is that put all the statements into a public static void main( ). Now if all these statements belong to main( ) then we can say execute main( ) and it would be easy to everybody as to what is it that we want to get executed from the machine. So, main( ) really becomes a collective name given to a set of statements. To avoid confusion between what belongs to main( ) and what not, put a pair of braces for the main( ) function. These pair of braces is known as scope delimiters because they delimit the scope of main( ). Delimit the scope of main means they indicate how many statements belong to main( ) hence known as scope delimiters. So here, when we say `public static void main( )`, main( ) is really a function as it is followed by a pair of parenthesis. Whenever in a Java program a word followed by a pair of parenthesis is seen, assume that its a function. We do not understand what is a meaning of public, what is a meaning of static and what is a meaning of void. At least to begin with some of these things we will have to take as it is. main( ) has to be prepended with public static void the way `println( )` has to be prepended with `system.out`. In
addition to public static void main( ), we also have to say before it, is class SimpleInterest and then to indicate how many statements belongs to this class we once again have to put a pair of braces. These also delimit the scope of the class called SimpleInterest and then we have to put before class SimpleInterest one more line saying package simpleinterest and semicolon. After class SimpleInterest there is no semicolon. After public static void main( ) there is also no semicolon but after package simpleinterest there is a semicolon. The name of the package is in a small case whereas the name of the class in SimpleInterest in which S and I are capital. main( ) is completely in small case. Always be careful about the case in which we have typed. So we have a package simpleinterest to which there belongs a class called SimpleInterest within which there is a function called main( ) and inside main( ) we have written several statements.

Tips...

Following are the some tips about the program.

- A Java program is nothing but a collection of one or more packages.
- Inside each package there can be multiple classes. In our case within the package called simpleinterest there was only one class which was also given the name SimpleInterest with a different case. So a package can contain multiple classes. In that sense package acts as a container which can hold several classes together.
- Each class can contain multiple functions. The class called SimpleInterest had only one function called main( ).
- We created variables. Any and every variable created in a Java program will always have to belong to some or the other function. It means there can never be a situation where we have a class which contains 5 functions and then we have defined some variables in between these functions. That is never allowed. The variable has to clearly belong to one out of these five functions. There can not be any hanging variable floating around in between functions. It has to belong uniquely to some or the other function.
- Every single function that is written that also can not float around. The function also belongs to some or the other class.
- Every single class has to belong to some or the other package. So the outermost container is package inside which a smaller container is a class, inside which a smaller container called a function and inside the function there are variables.
- The package, class, public, static, void all are keywords.
- While defining main( ) we had said public static void main( ). That order does not matter. If we said static public void main( ) that is also allowed. But do not say void static public that would be meaningless. void has to be just before main( ). Choice is available only in between public static and static public. Usually its public static void main( ).
- Java is a case-sensitive and a free form language. We have to be very careful while typing the programs in NetBeans.

Comments Are Useful

Any time we write a program, always give a comment at the top indicating what the program is all about. What is a goal and what is a purpose of the program. We are writing a comment saying Calculation of simple interest and placing it within a /*, */ combination. The /* has nothing to do with division or multiplication. Means the moment the context of symbol changes, its meaning is also likely to change. When we said p * n * r / 100, there * was for multiplication and / for division whereas with /* and */ and Calculation of simple interest, there is no question of multiplication or division. The comment we give is only for our and others understanding, the computer will
completely ignore a comment. To indicate to the machine kindly ignore the commented statement, we put it within a /*, */. So whenever the machine finds a /*, */ it believes that nothing to do with it and it goes to the next statement. The comment is sometimes known as a note or a remark.

### Types Of Comments

There are different types of comments we can write in a Java program. For example, there can be single line comments. Typically single line comments are not written within /*, */ instead we use a notation called //. If we give the comment at the beginning then in that line whatever we type everything goes to be a comment. So if we use a // it has to be after the statement. So, give it either in the line preceding the statement or in the same line after the statement. If we use a multi line comment just put it within a /*, */ combination. This is what is typically done at the beginning of the program. We indicate the goal of the program, the author of the program, at what date was program written. It is a good habit to develop to begin with while writing Java programs that we give sufficiently detailed comments at the beginning about the ownership, authorship and the purpose of the program. We also have a utility called javadoc utility which generates help about the program. If we want that certain comment should go into that help we have to write within a /** and */ as shown in the slide. If we write the comment in this manner in that case it will go into the help file when we generate it using a javadoc utility. These are the three basic types of comments that we can create in a Java program a single line comment, a multi line comment or a documentation comments.

### Tips About Comments

Following are the few tips about the comments.

- We are allowed to write any number of comments any where in the program. So we can write a comment after the statement, if we want we can also write it before the statement as shown in the slide.

- If we write nested comments that is not ok means within a /*, */ combination never try to give another /*, */ combination.
Using NetBeans

In this lecture you will understand:

* Understanding Integrated Development Environment offered by NetBeans
* Creating a program using NetBeans
* What is Java Platform
* Java FAQs
NetBeans IDE

NetBeans is an Integrated Development Environment which is a collection of several programs. For example, there is an editor which allows to type a program and if any mistake is committed in typing, it also allows to edit the program. So for typing and editing we would be using the editor part of the NetBeans IDE. It also has a compiler which takes the Java program that we have typed as input and converts it into a Bytecode. The Bytecode is not executable code. This is one step before execution. This Bytecode is later on converted into executable machine instruction but in the second stage. Unlike traditional languages like C or C++ where the program on compilation gets converted into an exe file that does not happen in case of Java. The program gets converted into some intermediate code known as Bytecode. The Bytecode is independent of any machine language. It also contains an interpreter which converts the Bytecode into the machine language. Machine language instructions are the one which actually get executed. Then it also has a linker to link the libraries and debugger to help to eliminate the errors. Bug means an error, debugging means process of removal of errors and debugger is a program which helps in removing the errors. Since all these five programs editor, compiler, interpreter, linker and debugger, all of them have been gathered together into the IDE that means the integrator. It is a development environment which is an integrated environment, collection of all these five different programs.

Steps In Using NetBeans

Double click on the NetBeans icon and then NetBeans will start. Then we have to create a New Project to begin with. Any new program we have to begin with creation of a project. The wizard in the NetBeans will create some skeleton code for the program. For example, in every single program there is a public static void main(). That is the wizard will generate readymade for us the moment we create a New Project. We only have to type the code that we want within that public static void main(). That is the utility of the wizard. It creates certain boiler plate code, readymade code, routine code that we need almost in every program. The wizard can provide the skeleton code. Once we have added the code to the code generated by the wizard then we have to compile the source file and the compiled version is the Bytecode and not the executable code. Then run the program and when we run the program that time the Bytecode is taken as the input and then the Java interpreter converts it into the suitable machine language instructions which gets executed on the machine.

Create New Project

To create a new project, click on the File menu of the NetBeans IDE 5.5, a menu will pop up and then select New Project.

Select Project Type

Once we select the New Project menu item, it will ask what type of project do we want to create, a class library, a Java project with existing source, etc. We want to create a Java Application so we select a Java Application. Then click on Next. This Java Application will create a new J2SE application which we can read in description that it creates a new J2SE application in a standard IDE project. We can also generate a Main class in the project. Standard project used an IDE generated and build script to build run and debug the program.

Project Name & Location

The dialog which appears we have to type the project name for the project. Suppose we give the name as SimpleInterest. Then we have to indicate that where the project should get created i.e. the project location. In this slide it is C:\Users\Kanetkar\JavaPrograms. We can choose some other location. If we want to browse the drives and the folders within those drives, we have the Browse button. We can click on that then we will get a File dialog which navigates through drives and directories. Choose a suitable directory into that this project will get created. When we do so, in the
Edit box simpleinterest.SimpleInterest entry will be made. First word simpleinterest is the name of the package. .classname is SimpleInterest. The first word simpleinterest is completely in small case which is package name and then class name we are giving as SimpleInterest. By default simpleinterest.Main gets created here. Delete Main and in place of that type SimpleInterest because we want main( ) present inside SimpleInterest class. So always this procedure that whatever is the package name, same we will give as the class name is followed i.e. packagename.classname. Finally click on the Finish button of the dialog.

Wizard Generated Code

When we click on Finish button, we get a wizard generated code. There are comments at the beginning following that there is a line package simpleinterest ;. Then it contains document comment which is used by javadoc. Then there is public class SimpleInterest and an opening brace. Inside that there is a function called public SimpleInterest( ) and then a public static void main ( String[ ] args ) and opening brace. This is a program that has been created for us readymade by the wizard. main( ) is written in a class called SimpleInterest. Package name is simpleinterest inside that the function public SimpleInterest( ) known as a constructor function. We will concentrate only on main( ) i.e. public static void main ( String[ ] args ). So inside the package there can be multiple classes, in this case package is simpleinterest inside which class is also SimpleInterest. In some other situation there might be a package name pkg, inside which there might be two classes, class One and class Two. The way here we have only one class i.e. SimpleInterest, multiple classes is also feasible. Inside each class then there can be multiple functions and inside functions there would be variables that we will have to define. The brace which is present at the end of public static void main( ) it brought to the next line by hitting Enter key to realign it.

Add New Code

Now the code we add is as follows. Declare p, r, si as float, declare n as integer, initialize p, n, r to values 1000.50f, 3 and 15.5f, use them to get simple interest si = p * r * n / 100 and once si is calculated print it out by System.out.printf ( "%f", si ). We can also do so by using System.out.println ( si ). When we use printf( ), we have to write "%f" indicating that si is a float. Instead of trying to remember %f and other format specifiers that exists in C language for printf( ), we use println( ).

Build & Test

To build the program means convert the program into the Bytecode. Go to the Build menu and from the menu, we will see an item Build SimpleInterest Project, select it. The moment we select that menu item, in the NetBeans IDE at the bottom, it will show the result of the compilation. If we get build successfully it means there is nothing wrong with the program, we have been able to convert the program into Bytecode successfully. If for some reason we are unable to get the built done successfully then any errors that the compiler has located in the program they would be listed out exactly in the same window just below the program window and then the errors can be identified, read those errors, go back to the program, rectify those errors and then once rectification is done, save the project and once again click on Build. From the menu, select Build Project to get the program recompiled. So unless and until, the program is error free we will have to keep repeating the cycle of rectifying mistakes and then trying to build the program once again. Once we have completed the successful building of the project then go the Run menu. From the menu we will be able to locate an item called Run SimpleInterest Project, select it. When we do so, we would be able to execute the program and once we execute the program, in the window we will see there is 466.something written. The output that System.out.println( ) printed on the screen that is the value of si. Now instead of everytime going to Build menu and then selecting Build SimpleInterest project or everytime going to Run menu and then selecting Run SimpleInterest project, there is a more effective way of doing it i.e. by using function keys F11 and F6. Everytime to compile the project hit F11. Once it is compiled or
build successfully then hit F6. So F11 and F6 are shortcuts for building the project and executing the project.

What Happens

The program name is SimpleInterest.java. When we did F11 we submitted the program to the Java compiler. The Java compiler is known as javac compiler. javac means java compiler. This compiler takes the SimpleInterest.java as the input and produces a Bytecode which it stores in file called SimpleInterest.class. Since, the program had only one class, only one .class file got created. If there been multiple classes multiple .class files would have been created. When we submit the SimpleInterest.class to the Java interpreter which is the part of the Java Virtual Machine, converted from Bytecode into native machine language instructions and those machine language instructions are actually executed on the machine. If we want to locate where this SimpleInterest.class is present, go to the File view on the left hand side of the IDE where different classes that are there in the project those are shown. If we go to the Files tab, we will be able to locate SimpleInterest.class. At the bottom we can also locate SimpleInterest.java. Basically Files tab shows all the files that are present in the project. So once we build the program successfully SimpleInterest.class is indeed present in the Files view of the project.

What Is Run Anywhere

Java programs are run anywhere programs. Why? In our case, we had SimpleInterest.java. Using the compiler we converted it into the Bytecode. We can realize that Bytecode is independent of the machine, whether it’s a Windows machine, whether it’s a Mac machine or whether it’s a UNIX machine, it does not matter. Its intermediate code is not specific to any specific machine's machine language instruction. The Java interpreter has to convert this into the actual machine language instruction which is the job that is done by the Java Virtual Machine ( JVM ). Since the Bytecode is independent of machine, we can take the Bytecode to any machine so long as there is a JVM present on that machine, we would be able to successfully execute the program there. This explains the meaning of run anywhere. We create the code, Java Bytecode on one machine and take it to any other machine. If there is a JVM installed on that, we would certainly be able to run the Java program there. So it is said that write once, run anywhere.

What Is Java Platform

At the bottom, we have the hardware. On this hardware there is the operating system is running. It might be Windows operating system, Macintosh operating system, UNIX operating system or a Linux operating system. Any one of these operating systems should be running on that hardware. On top of this there would be another program running known as Java Virtual Machine ( JVM ) and then there are set of classes available, a library of classes available known as Java API. The way we had a SimpleInterest class, likewise Java designers have also created readymade classes that will help in building programs quickly. That set of classes is known as Java API. So API a library of classes and all these classes are organized in the form of different packages. The way in our one package there can be multiple classes, similarly, in this library also related classes have been packaged together under one common package. On JVM there are lots of classes available in the form of package libraries and on top of that the SimpleInterest.java runs which contains the Bytecode. The Java API and JVM together are commonly known as Java Platform. So if Java Platform stands installed on the machine means JVM stands installed and the libraries are available for any program like SimpleInterest.java to be able to use them. This constitutes the Java Platform.
Generic Programs

In this lecture you will understand:

* More general programs
* Modulus operator
* Command-line arguments
Receiving Floats

We begin with a package and the name of the package is sample. Then we write import java.io.* and import java.lang.*. io is input/output whereas lang is language. Name of the class is Sample which contains public static void main ( String[] args ). args is an array of string arguments. NetBeans wizard will always create the main( ) in the form given in the slide where main( ) contains String[] args as the argument for main( ). Following main( ) we write throws Exception. It is important because we are receiving value of a float from the keyboard. While receiving these values, if something goes wrong then a runtime error would occur means error that occur when we are executing or running the program. When these errors occur, we say that either we will deal with these errors or somebody else will deal with these errors. If we want somebody else to deal with this error, in that case we need to throw that error by saying throws Exception where Exception is a name of the class and throws is a keyword. So whenever we receive any input i.e. int, char, float, double, etc. following main( ) we should write throws Exception. Then declare the variables float p and String str. Receive the input from the keyboard into float p. For that we have to receive the float into a string and then from the string we have to convert it into a float. To do so, we need a BufferedReader b. Now we say b = new BufferedReader ( new InputStreamReader ( System.in ) ). If we want to receive input from keyboard in Java, there is no function like scanf( ) in C programming or like cin in C++ programming. Java does not uses the scanf( ) and cin because scanf( ) and cin do not receive the input in type safe manner means in a float we are allowed to receive int, in double we are allowed to receive a float which breaks the type safety. In a float we should be allowed to receive only a float, in a char only a char, in a double variable only a double, etc. are not enforced by scanf( ) and cin. So Java designers decided that all input and output is based on a stream of characters. Stream of characters can come from a keyboard; stream of characters can come from a file or from a network. So all input and output in Java is a stream based. So firstly do new InputStreamReader. new will always create a new object in memory. So a new object called InputStreamReader is created and passes the parameter as System.in to it. System.in refers to console input. So there is a console input stream where the input is assumed to be coming from keyboard. System.in already stands created. We are trying to create a new object called InputStreamReader into which we are trying to store System.in means the console stream. Then we say new BufferedReader. The way new InputStreamReader created one object, new BufferedReader would create another object of the type BufferedReader. So BufferedReader is the outermost object which has an embedded InputStreamReader object which has a embedded System.in object. Once the BufferedReader object is created in memory, wherever it is created in memory, its address is returned to us which gets collected in b. After that we can say b.readLine( ). When we do so, means we are trying to call a method inside the BufferedReader object and name of that function is readLine( ). b.readLine( ) can read a line from the keyboard because within BufferedReader we have an InputStreamReader object which contains a console input object. Hence, b.readLine( ) will read from the console. To InputStreamReader if we provide a File object instead of a System.in object then through the BufferedReader object b we can start reading from the file. Likewise in BufferedReader, if we store a network object instead of System.in then the same b will help to read from the network rather than from the file or from the keyboard. When b.readLine( ) is done, we would be able to read one complete line which is a string of characters and returned to us in str. Then convert that string into float using Float.parseFloat( ) which is similar to out.println( ) where Float is an object and parseFloat( ) is some method inside it. Float.parseFloat( ) receive a string and convert it into a float. For example, if to b.readLine( ) we supply a float value 3.14 then in str 3.14 gets stored as a string of characters and not as a float. So this string 3.14 is actually get converted into a number which is a job of parseFloat( ). The float obtained is collected in p and then printed using System.out.printf( ). For the Float.parseFloat( ) to work, we say import java.lang.* at the top. Likewise for BufferedReader and InputStreamReader to work we import java.io.* at the top. BufferedReader, InputStreamReader and Float are classes. A class is a user defined type whereas an object is like a variable.
**Generic Program**

Name of the package is simpleint. Then import java.io.* and import java.lang.*. Name of the class is SimpleInt inside it there is main( ) which throws a runtime exception if that occurs. Then we say BufferedReader b = new BufferedReader ( new InputStreamReader ( System.in ) ). Once b is ready then we can display a message on the console to enter the values. Means we are asking for the values of p, n and r to be entered. Then do b.readLine( ) which will read a string even though actually that string contains a float. We will separate that by using Float.parseFloat ( str ) which converts string to a float and collect in p.

**Slide 5**

Similarly we can receive the integer by calling b.readLine( ) and then use Integer.parseInt ( str ). Then collect the value in n which is number of years. On similar lines receive the value of rate of interest using b.readLine( ). Rate of interest is a float so use Float.parseFloat ( str ) to convert it into an actual float value and collect it in r. Once p, n and r are ready, we can utilize them to calculate the new value of simple interest and finally print it out using System.out.println( ). println( ) always prints a string. So, if we want to print a message, we can say System.out.println ( "Simple int. = Rs. " + si  ) and a value can be attached to it using a + operator. + does the concatenation of two strings. So, si is first converted into a string and then concatenated at the end of Simple int. = Rs. and finally this message along with the value of simple interest is printed on the screen.

**How To Supply Values**

When we actually type and execute this program in NetBeans, it is found that any time Enter values gets printed. As shown in the slide, it is the input edit box in which we are supposed to type out the values. So the value of principle, value of number of years and also the value of rate of interest gets typed out in the same box. Finally we get the answer for the simple interest.

**Interchange Contents of 2 Variables**

Now we want to interchange the contents of two variables. Means suppose there are two variables c and d holding the values. We want that whatever is present in c should go to d and whatever is present in d should go to c. The NetBeans wizard creates the package and class statements. So it is dropped. In main( ) receive values for c and d. Suppose the values that are supplied are 5 and 10 respectively. c and d are the integer variables. Into c and d, 5 and 10 gets stored. t is also an integer variable which does not contain anything. So as shown in the slide x is put in t. To exchange the values of c and d, we say t = c. c contains a value 5, so 5 will get assigned to t. So, the values of c, d and t are 5, 10 and 5. Once value present in c is safely stored in t, we can now overwrite the value stored in c with 10 by using c = d. When we do so, d which contains 10 is assigned to c. So c contains 10. Now if we want that 5 i.e. present in t should overwrite that is present in d then we can do d = t. When we do so, d starts containing value 5. Once this is done we can print values of c and d using System.out.println ( c + " " + d ). So we can print multiple variable values. So the values get printed as 10 5 i.e. c contains 10 and d contains 5. So the contents of c and d are interchanged.

**One More Way**

There is one more way to interchange the contents of c and d. Once again receive the values of c and d. Suppose we supply values of c and d are 15 and 10 respectively. Then we do c = c + d. So 25 will get assigned to c whereas d contains 10. We can easily obtain 15 from the values 25 and 10 present in c and d. So do, d = c – d i.e. 25 – 10 = 15 will get stored in d. So d starts containing 15. Now from 25 and 15 present in c and d, we can easily get 10. So c = c – d will give 10 and get stored in c. So originally c and d were 15 and 10, now they become 10 and 15. It means that we had interchanged the contents of variable c and d without using the third variable t. Print the values of c and d which
will give 10 15. The reason to carry out the task of interchanging the contents of two variables is as follows.

- If a University may want to arrange roll numbers or enrollment numbers in ascending or descending order. To arrange 10 number in ascending or descending order if 1st number is greater than the 2nd, we will exchange 1st with the 2nd. If 3rd is bigger than the 7th then 3rd and 7th is to interchange. Means while arranging the numbers in ascending or descending order, we might require to interchange between the pairs of numbers.
- Books in a library we can arrange in an alphabetical order by name. If this is the requirement then we will be required to interchange between several pairs of names.
- In a bank we can arrange the transactions that take place in a chronological order by dates.

In short, if there is a sorting either numerical or alphabetical or chronological by date then we will have to interchange the contents of two variables.

### Sum of Digits

First receive a 5 digit number into a variable n. Suppose the number is 26913. To carry out sum, first we must be able to carry out separation of each digit from a 5 digit number into a separate variable. 2 is get stored into d1, 6 into d2, 9 into d3, 1 into d4 and 3 into d5. If the digits get separated into d1, d2, d3, d4 and d5 then sum can be obtained as d1 + d2 + d3 + d4 + d5. If we do 26913 / 10 we get 2691 which is quotient but we want digit 3. So we do 26913 % 10 then we get the remainder as 3. Whenever we want the remainder, make use of the % ( Mod ) operator or modulus operator.

### The Whole Picture

To begin with we receive the value of n and suppose it is 26913. Then we do d5 = n % 10 i.e. 26913 % 10 which gives 3 and gets stored in d5. Once 3 is extracted then we want to extract 1 in variable d4. For that first do n = n / 10 which gives 26913 / 10 = 2691 which is stored in n. Now again do d4 = n % 10 which extracts 1 and stores in d4. Reduce 2691 to 269 using n = n / 10. So now n contains 269. To extract 9 do d3 = n % 10 which stored 9 in d3 and reduce 269 to 26 using n = n / 10. So now n contains 26. Now extract 6 from 26 using d2 = n % 10. So d2 now contains 6 and reduce 26 to 2 using n = n / 10. Now n contains 2. 2 can be directly assigned to d1 or we can say d1 = n % 10 which are same. Finally add all these and store in s using s = d1 + d2 + d3 + d4 + d5 and print it which displays 21 for a number like 26913. % operator can be used in many cases.

- In determination of a leap year
- In determination of a number is odd or even
- While determining a number is prime or not

In short, if we are interested in quotient use / operator and if we are interested in remainder use % operator.

### Command Line Arguments

We have String[ ] args in main( ) which is an array of strings. If we want to print out the values of arguments then we can do by using System.out.println ( args[ 0 ] ), System.out.println ( args[ 1 ] ) and System.out.println ( args[ 2 ] ). It is assumed that 3 arguments are supplied at command prompt which are collected by main( ). Once main( ) collects these three arguments in array of strings then we can print each of the argument by saying args[ 0 ], args[ 1 ] and args[ 2 ].

### Supplying Command Line Arguments
In the NetBeans IDE, locate the project and then click on Run. When we do so, we get a chance to supply command line arguments. For example, cat dog parrot. So System.out.println( ) will print these 3 arguments cat dog parrot.
In this lecture you will understand:

* Types of Java Instructions
* Type declaration instruction
* Expression instruction
* Nuances of boolean, char, int, etc.
* Mixed mode expressions
Where Are We....

We began with alphabets, digits and special symbols. Then we learned how to combine them to form constants, variables and keywords. Next step was creation of statements or instructions which was skipped and we have straightway gone ahead and written a few programs. Now that we are familiar with atleast some programs. There are different types of instructions. One is type declaration instruction then expression instruction. There are several control instructions. There is a synchronization instruction and a guarding instruction. Out of these type declaration, expression, control and guarding these are something which we have knowingly or unknowingly used.

Type Declaration Instruction

We will begin with the type declaration instruction. For example, int i, j, k or float a,b,c or char ch. All of these are known as type declaration instructions because we are trying to declare the types of the variables. i, j and k we are saying to be integer or integer type of variables. Suppose we say int i and in the next line we say i = 5, then it is allowed to combine them and say int i = 5. It means that wherever we are trying to define a variable i at that time we can also initialize its value to some value like 5. If we do not initialize then default value is 0. We are initializing multiple variables as we defined them. It is wrong to say that we can initialize only one variable. Here we have initialized a, b and c. Moreover c is being initialized to result of the expression a + b * 5 % 2. Now in the first declaration int i, j, k, it does not matter if we say i, j, k or j, k, i or k, j, i. The order of variables can be anything. But in the last example, c will have to be defined after a and b have been defined. Because while initializing c we are indeed making use of a and b. Unless a and b stand defined by that time we cannot think of using them to evaluate some expression. So the rule is, define the variable before point of usage of the variable. Just before we use a variable by that time if we have defined the variable. No need to go write always at the top of the main( ) and then try to define the variables. We can define the variables anywhere but before starting to use them. For example, suppose we have a, b, c, d as four variables and we want each one of them to be initialized to a value 5. We can initialize by saying a = b = c = d = 5. Means we can initialize multiple variables to some value like 5. But if we combine them and say int a = b = c = d = 5, that does not work. Because when we say int a = b by that time b does not stand defined. Since it does not stand defined we cannot use it within an expression. So the first one will work, second one will not work. Be careful while initializing multiple variables to the same value.

Expression Instruction

Expression is the arithmetic expression. s = d1 + d2 + d3 or si = p * n * r / 100 all these are arithmetic expression instructions. While writing these expression instruction we are using operators like +, -, *, / and %. All of them together are called as arithmetic operators. But the exponentiation operator is missing. Many other languages do provide exponentiation through a ^ ( carat ) symbol or through a ** operator. For example, 2 ** 5 or 2 ^ 5. So either 2 ** 5 or 2 ^ 5 gives 2 to the power of 5. However in Java none of these would work. If we want to raise one number to the power of another the way to do so is to say, double a ; and then Math.pow ( 2, 5 ). Math is a class and pow( ) is a function inside that class. So we can invoke the pow( ) function by saying Math.pow( ) which receives 2 and 5 as the input. It will do 2 to the power of 5 and then final result of the 2 to the power 5 i.e. 32 will be returned and collected in some variable a. We are using double a instead of float a because pow( ) function always returns a double value. double stands for double precision whereas float stands for single precision. The more precise number will be the double precision number. i = j * k + 3 cannot be rewritten as j * k + 3 = i because = does not mean left hand side is equal to right hand side, = is another operator called assignment operator. The assignment operator is supposed to evaluate the expression on the right hand side and assign its value to the variable on the left. So on the left hand side of the = an expression cannot occur because we cannot assign the value to an expression. We can assign the value only to a variable. Hence the second one is incorrect because on the left hand side = there is an expression to which value of i cannot be assigned. When we say i = 3, we cannot say 3 = i. Since we cannot assign a value of i to a constant and if that assignment is permitted then constant no longer remain a constant. Hence, in other words on left hand side of = a constant cannot occur, an expression also cannot occur. Only thing that can occur is indeed a
variable. So if anything other than that occur on the left hand side of =, will report an error. Suppose we say \( a = b \ (c + d) \). Now this appears to be ok but it is wrong. Because Java assumes absolutely no operator to be present anywhere. Just by putting \( c + d \) within parenthesis, normal arithmetic may mean that \( b \) is being multiplied by \( c + d \) but in Java programming unless we explicitly mention the star operator, its not going to assume that we are trying to carry out multiplication between \( b \) and \( c + d \). Means no operator is ever assumed, we have to be explicitly mention the operator that we intend to make use of at any place.

**Boolean Type**

We had done only a int, a char and a float. There are other primitive types as well. A boolean we learned can take a value as true or false. Result of a boolean expression is also either true or false. To a boolean variable \( a \), assign a value false. boolean is a keyword. We can also say System.out.println( "a = " + a ). \( a \) is false so \( a = false \) is printed out. We can also do System.out.println( "Result = " + ( 4 > 3 ) ). ( 4 > 3 ) is true. Hence, outcome will be Result = true. Means \( 4 > 3 \) is either replaced by true or by false based on whether it’s actually true or not. So any condition we write will evaluate to either truth or falsity whatever it evaluates to, accordingly it will be replaced with either a true value or a false value. Never try to use true and false as variable names. In C, true means 1 whereas false means 0. No such thing in Java programming. True is not 1 and false is not 0. So if we do int a and then \( a = 3 < 4 \), do not think that \( 3 < 4 \) is true and hence to be replaced with 1. 3 < 4 is true hence, true is to be assigned to a but that does not happened because a is an int. Hence, the only way to do so is declare a boolean b and then assign true or false to it by saying \( b = 3 < 4 \). 3 < 4 is true and hence b will now start holding a value true.

**Char Type**

If we do char ch = 'A' and then try to print out ch, 65 which is the ASCII value of A is not stored in ch instead the Unicode equivalent of A gets stored in it. Every ASCII character is 8 binary digit long whereas every Unicode character is 16 binary digit is long. Binary digit in short can be called as bit. Hence, ASCII is always is a 8 bit number whereas Unicode character is always a 16 bit number. Unicode offers the flexibility of using any language, any script within the program. With ASCII it was not possible because it was only a 8 bit character set whereas to accommodate different scripts and different languages Unicode scheme of things suggests that every character be stored as a 2 byte entity rather than one byte. So a size of a char is always 2 bytes rather than 1 byte. Now if we say char ch = 'A' and then in dh we will try to assign that a value of ch + 1. Then we print out dh. We will find that dh = ch + 1 is indeed wrong because on a char we cannot perform such operations. The solution for that is to do int dh = ch + 1. The operations like char dh = ch + 1 are not permitted because whenever we try to do arithmetic, just before doing the arithmetic a char will be promoted to an int. When we do int + 1, it will result into a int and int when we try to store in a char, there is a possible loss of precision. In this case there would not be any but in some other situation where we are trying to store a number like 2,15,000 into a char that will lead to a problem. So any time a int we attempted to assign to a char, we would certainly report as an error saying that there is a possible loss of precision to happen if we carry out this operation. So such operation is rejected. So make dh as an int then we can get an integer answer into dh.

**Integer Type**

- Integer comes in 4 different flavors. There is a byte, a short, an int and a long.
- Most common amongst them is the int.
- Byte and short are used in place of int whenever we want to save space and we are frequently required to do based on for what device we are trying to write a program. If we are trying to write a program for a PC then we can assume that ample space available but if we are trying to write a Java program for an embedded system or for a mobile phone, the space constraints should be there and they would definitely not enjoy as much space as the PC does. So when we use a byte or short in place of a int because int will occupy more space than a byte and a short. If we are trying to store age of a person, no point in using int or a long int for that because byte and short can easily take care of them with lesser number of bytes. Age of any person is rarely go to beyond 100
years. So a byte or a short is perfect for storing an age rather than sacrificing more bytes by using an int or a long int for storing age of a person.

- Long is used whenever we intend to give a very big integer value.
- Whenever we write a number without a decimal point, it is always treated as an integer. For example, 25 can be a short int, might be an int, might be an byte, might be a long int because 25 falls in the range of every one of these four integer types. But by default 25 treated as an int, not as a byte, not as a short, not as a long int.
- Whenever we carry out assignment the value being assigned should never exceeds the range of the variable. For example, byte a = 300, short b = 40000 and int c = 2200000000. In this byte is only one byte entity it can never hold anything bigger than 127. Short is a 2 byte entity it can never hold anything beyond 32767 and int is a 4 byte entity it can never hold anything bigger than 2147473647. So all these numbers are exceeding the ranges of byte, short and int. So they would be rejected and would indeed be reported as errors.

Real Type

- There two types available, one is a float and another is double.
- Most common amongst them is a float.
- Whenever we want to deal with very big real numbers then we need a double. Because a float offers a range up to $10^{38}$ means 1 followed by 38 zeroes. Only if the program at hand needs to deal with numbers bigger than this number then only think of a double, otherwise most of the times float is used.
- A number with a decimal point is going to be treated as a double.
- During assignment value being assigned should not exceeds the range of the variable.

Mixing Types

- Suppose we have float a = 3.14 and int b = 3.14. This is wrong because 3.14 is a double, it is not a float. A number with a decimal point is not a float, it's always a double. int b = 3.14 is wrong because there is a loss of precision when we are assigned 3.14 which is a double to a int called b. To make it correct, declare 3.14 as a float by adding f at the end which might be a f or F. Alternately leave 3.14 as it is means leave it as a double and make a as a double by saying double a. If we can do then 3.14 which is a double without any loss of precision can indeed get stored in a float a. If we want to assign 3.14 to a int b, by default it would be wrong. Hence, we will have to specifically first convert that 3.14 which is a double into an int then only try to assign it to b. The way to do so is to say, a = (int) 3.14f. 3.14 is a float, we are trying to convert it into an int by using typecasting operation. Typecasting means casting a particular type into some other type. We are converting 3.14 into an int then assigning it to int a.
- Use the typecasting to say short a = (short) 32768 and then attempt to print out a. We get -32768. This is so because the range of a short int is -32768 to +32767. By attempting to assign 32768, we are crossing the positive side’s maximum value i.e. +32767. But 32768 otherwise is only an integer. It's a number without a decimal point, so it's an integer. We are first attempting to convert it into a short using typecasting and since short is only two bytes; within two byte 32768 cannot be stored; only upto 32767 can be stored. So the number on the other side i.e. a negative side gets picked up. First number on negative side is -32768, hence it is assigned to short a. If we say (short) 32769 in place of 32768 then second number of the negative side i.e. -32767 would have been assigned to the short a. Same would happen if we try to cross the -ve side, we will be dropped to the +ve side; the way if we crossed the +ve side we are taken to the -ve side.
- The widening conversions will take place automatically. Widening means conversion from a
data type which offers a smaller range to a conversion of a data type which offers a bigger range. If we say long l = 40000 * 2, then 40000 and 2 are integers. We are trying to convert result into a long integer. 40000 * 2 both being integers will fetch us an integer result but we are trying to convert it into a long int before it gets stored in l.

- So make it clear that why int a = 3.14f without the type casting operation failed because it is a narrowing conversion which always going to report an error unless we specifically convert the type using the typecasting operation. If we do short s = 40000, it is wrong because 40000 is an integer and we are trying to assign it to short.

- Any possible loss of precision is reported as an error. For example, s = 2 * 5 is ok but if we say s = 2 * i is wrong even though i holds same value 5. We believe that we will get 10 which is wrong. First one works whereas in second case we get an error because i has a value 5 but it might happen that between int i = 5 and s = 2 * 5 there might be some other Java statements which may modify the value of i. If value of i gets modified and becomes 100000, then 100000 * 2 is going to certainly exceed the range of a short int. Hence any time we use a variable in an expression which is likely to result into a loss of precision that will be reported as an error. When we do s = 2 * 5, it is understood that 2 * 5 never going to exceed the range of a short which is upto +32767.
Precedence And Associativity

In this lecture you will understand:

* What is mixed mode arithmetic?
* What is precedence of operators?
* What is associativity of operators?
* Printing functions and their features
Mixed Mode Arithmetic

Here we are using a combination of operands in their different forms. Every one of these expressions are not really mixed mode expressions. The first expression 5 / 2 is an integer mode expression because both the operands in this expression are integers. Likewise when we say 5.0f / 2.0f, both of them are floats. Hence, this is not a mixed mode arithmetic instruction whereas when we say 5 / 2.0f, it is a mixed mode because 5 is an integer whereas 2.0 is a float. In the first expression i.e. 5 / 2, 5 / 2 is bound to output a value 2 because both the operands 5 as well as 2 are integers and the rule is that if both the operands are integers answer must always be an integer. So 5 / 2 bound to give us the integer value 2. When we do 5.0f / 2, 5.0f is a float whereas 2 is an integer. So this becomes mixed mode arithmetic where one operand is float and other is an integer. In such a case, the integer is first automatically promoted to a float means when the computer gets to evaluate this, it will try to evaluate 5.0f / 2.0f and that results to 2.5. Likewise when we do 5 / 2.0f, 5 is an integer and 2.0f is a float which is a mixed mode arithmetic and such mixed mode arithmetic is carried out by first promoting integer to a float. So 5 become 5.0f and 5.0f / 2.0f gives the output as 2.5. Next 5.0f / 2.0f, both are floats so the result is 2.5. When we do 2 / 5, both are integers so the result is 0. Now when we do 2.0f / 5, this is again mixed mode arithmetic. 5 will be promoted to 5.0f, 2.0f / 5.0f is bound to be 0.4. Again in 2 / 5.0f, 2 is first promoted to 2.0f / 5.0f will give the result 0.4. In the last 2.0 / 5.0f, we get the result as 0.4. But in this case 2.0 is a double whereas 5.0f is a float. So whenever we do the operation between double and the float, the float be will promoted to double. So the expression that will be evaluated is 2.0 / 5.0f and that once again give the result 0.4. It means an operation between int and float becomes an operation between a float and float. Likewise an operation between a double and a float ultimately becomes an operation between a double and double.

All Possible Cases

If the two operands, operand1 and operand2 are integers then the result is going to be an integer. If the two operands, operand1 and operand2 are floats then the result is bound to be a float. The machine can never carry out the operation between int and a float. Hence, int first gets promoted to a float. The int, float operation becomes a float, float operation and that bound to give a result float. Likewise if we do a operation between a float and a int, int is first promoted to a float, float and float would once again gives the result as a float. So as shown in the slide, we know that if both the operands are integers then only we will get the result as integer otherwise we always get the result as float. The result is always be a float if one of them is float or if both the operands are floats.

Which Is Correct?

Suppose we write some expression in which we are trying to convert Fahrenheit degrees to Centigrade degrees. The formula to do so 5 / 9 * ( f – 32 ). We have initialized f to a value 212.0f. f is Fahrenheit degree and c is the centigrade degree. Both of them we had taken as floats. Instead of saying c = 5 / 9 * ( f – 32 ), we can also do 5 / 9.0f * ( f – 32 ) or we also do ( f – 32 ) * 5 / 9. Now for 212 degree Fahrenheit, we know that equivalent centigrade degrees are 100. However, when we try to do 5 / 9 * ( f – 32 ), the standard formula for converting a temperature in Fahrenheit degrees into Centigrade degrees, we expect 100.0 but we get 0.0. Because in the expression c = 5 / 9 * ( f – 32 ), f – 32 will be done first as it is parenthesized and then 5 / 9 will be done. But when 5 / 9 is done, 5 and 9 both are the integers, so we are going to get the answer as integer and 5 / 9 is going to give us as a 0 and 0 multiplied by any float will gives 0.0. So the value of Centigrade degrees in first expression is turned out to be 0.0. To avoid this we will keep 9 as 9.0f and we keep 5 as it is. When we carry out this expressions evaluation, again first f - 32 is done and then 5 / 9.0f is evaluated. When we do 5 / 9.0f, 5 is first promoted to 5.0f and then 5.0f / 9.0f will be float / float that will give a result as float. float * result of f - 32 is also a float. So now we get 100.0 in c which we expect. In the third expression we are saying ( f - 32 ) * 5 / 9. In this expression first f - 32 is done, 32 is an integer whereas f is a float. So when we do float – int, it becomes float – float and gives the result as float. Then the result is multiplied with 5 which is an integer. When we do float * int, int is promoted to float. So float * float will give the result as a float. This float is then divided by 9. So float / 9 will become float / 9.0f because 9 is promoted to float i.e. 9.0f. float / 9.0f gives the result as a float. Hence c will contain a correct value i.e.100.0. Thus the order in which we write the expression is
very important. The order in which we write the expressions indeed change the results. In all the three expressions the parenthesis is important. If we do not give a pair of parenthesis around \( f - 32 \), then 32 will get multiplied by 5 in the last expression. But we do not want so. Hence, parenthesize \( f - 32 \). This is so because * enjoy the higher priority than -. Like BODMAS rule, here is also a set of rules for operators in which operations are carried out.

**Hierarchy/Priority/Precedence**

As shown in the slide, \( c = a + b * c \% 5 - d * 6 \). This expression uses several operators. We have seen that = enjoys the lower most priority. The arrow shown in the slide indicates that the priority or precedence increases towards the top. Means *, /, % enjoys the higher priority than +, - and +, - enjoys the higher priority than the = operator. Unlike BODMAS where Division enjoy the higher priority than Multiplication, in Java programming *, / and % all enjoy exactly same priority. Similarly + and - enjoys same priority which is again not so in BODMAS whereas in BODMAS Addition enjoys higher priority than Subtraction. In this expression, the first operation that would be done is \( b * c \) because *享受s the higher priority than = and +. Second operation that is done is the % operation and the third is \( d * 6 \). % operation when carried out, it will be done on the result of \( b * c \) and then on the result of \( b * c \), using % operator with 5, operation is carried out. The third operation will be \( d * 6 \). Fourth will be the addition and fifth will be the subtraction. Finally we do the assignment. Now sometimes it is believed that *, % and * that are being used in the expression, we are evaluated in the same order in which they appear in the expression. Firstly * then % and then * gets evaluated which is a wrong belief. In this expression it is so but in every case it may not be so. Do not think that * is done earlier because it occurs earlier in the expression. In fact *, % and * enjoy same priority. Means now there is a conflict that which one has to be done first and which one should be done latter. This conflict is solved using associativity of operators.

**Associativity - I**

Take a simple expression \( c = 3 / 2 * 5 \). In this expression either / or * will be done earlier. * and / both enjoy same priority. Since both enjoy the same priority, we have to decide their associativity. Every single operator has got some associativity and some precedence. If precedence is same then associativity decides which operation would be done earlier. The associativity has been determined as Left to Right for the / as well as the * operator. This associativity is predetermined by designers of Java. We cannot change it. Both the operators * as well as / enjoy Left to Right associativity means out of these two / as well as the *, that operator will be performed first or that operation is performed first which has a unambiguous left operand. Unambiguous means non-confusing. Now we will follow the table given in the slide. In case of / operator, left operand of the / operator is 3 whereas right operand of / some may believe that is 2, some may believe that it is either 2 * 5 or 5. It means that left operand is ok whereas right operand is not ok. Some may believe it is 2, some might believe it is 2 * 5. In case of * operator, the left operand of it some may believe that it is 2 whereas some will believe that it is 3 / 2 whereas right operand is unambiguous. Right operand is 5. In this case left operand is not ok whereas right operand is perfect. Right operand for * operator is 5. Hence, in this case the first operation that will be done is 3 / 2 because / and * enjoy same priority so associativity will help us to settle the tie and while settling this tie only that operator will be evaluated first which has a unambiguous left. For a / operator left operand is 3 whereas for * operator it is 2 or 3 / 2. Means for the / operator, left is unambiguous i.e. non-confusing. Hence, / operation will be done first and then * operation will be done and last is the assignment operation.

**Associativity - II**

Suppose we take expression like \( a = b = 5 \). In this expression there are two operators, = and =. We will settle this tie using associativity of operators. Assignment has a Right to Left associativity indicating that right operand will be unambiguous. Whenever we come across such an operator that operator would be done earlier. Now as shown in the slide if we take the first = operator, its left operand is a which is unambiguous whereas somebody might believe that right operand is either b or b = 5. We believe that left operand is non-confusing whereas right operand is likely to be confusing. As against this if we take the second = then left operand might be b or a = b whereas right operand is 5. Means left is confusing whereas right is non-confusing. So first operation that would be carried out is the second = operation.
because for the second =, right operand is non-confusing. Then first = is done. Means here we cannot say that first = will be done first because it occurs first whereas in actuality it occurs first but it is done latter. So it is wrong to believe that what occurs first will be evaluated first. When there is a tie between operators is decided by what is the associativity of operators. Every operator will either have a Left to Right or Right to Left associativity. This associativity is predetermined by the designers of the language.

**Output Functions**

There are three output functions available `println( )`, `print( )` and `printf( )`. Whenever we use `println( )`, it will print some output and then position the cursor on the new line. So `ln` really stands for print and go to the next line whereas `print( )` will not take the cursor to the next line. It will keep it wherever it is positioned at the end of the `print( )` function whereas `printf( )` is used for formatted printing which is similar to the `printf( )` we heard in the C programming. `print( )` and `println( )` both are to be used by saying `System.out.println( )` or `System.out.println( )`. If we execute the 4 instructions given in the slide in a Java program, the output will be as shown in the slide. We can observe that First Second Third has come in the same line because we are using `print( )` function. It prints First, position the cursor in the same line, print Second and when `println( )` goes to work, it prints Third and then the cursor goes to the beginning of the next line. `println( )` does not mean begin printing in a fresh line. It means print and then position the cursor to the beginning of the next line. So Fourth comes in the next line.

**printf( )**

The general form of `printf( )` is as shown in the slide. We say `System.out.printf ( "format string", list of variables )`. List of variables is optional. We may say `System.out.printf ( "Enter values of c and d" )`. If we do so then this message will get printed as it is on the screen. But if we want to print messages the `println( )` and `println( )` are handy. We should use `printf( )` whenever we want to format the output. So `f` in `printf` stands for. We may use a variable or a constant or an expression in the list. Means it is strictly not the list of variables. List may contain constants like 35 or expressions like 2 + 6 % 3. All of them are integers hence all three would be printed using `%i`. `%i` is the format specifier for printing out the integer.

**printf( )**

The format string can contain not only format specifiers like `%d`, `%f` or `%c`, it may also contain certain escape sequences and also any other characters. `%d` and `%i` are similar. d stands for Decimal integer means integer expressed in decimal numbering system. We can print float using `%f` whereas we can print char using `%c`. Suppose we say `System.out.printf ( "\n" )`. The \n means a new line and it nothing but an escape sequence. The way every format specifier begins with the % likewise the every escape sequence begins with the \ in `System.out.printf ( "\nSimple Interest = Rs. %f", si )`, Simple Interest = Rs., all of these belong to the category of any other characters and they get printed as they are in the slide whereas %f is the format specifier. Using %f the value of si will get printed out, assuming that si is some float variable. We are using %f to print the value.
Decision Control Instruction - I

In this lecture you will understand:

* What are Control Instructions
* What are decision control instruction
* if, else keyword
* Relational operators
Instructions

The instructions that are available in Java are as follows.

- Type Declaration Instruction
- Expression Instructions
- Control Instructions
- Synchronization Instructions
- Guarding Instructions

Out of these instructions first two are done. So we will begin with the Control Instructions.

Control Instructions

Control instructions are the instructions which control the sequence of execution of instructions in a program. Sequence of execution of instructions means deciding what is executed first, what is executed after that, which statement is done after that and so on. This is a sequence. The sequence is decided by the control instructions. The different types of control instructions are as follows.

- Sequence control instruction
- Decision control instruction
- Repetition control instruction
- Case control instruction

Normal Java Program

The programs that we had seen so far follow the pattern given in the slide. We used to have a class called sample and inside the sample class we have a main( ). Then we can have some declarations of variables int, float, char etc. and then we receive some input and whatever input we receive, we use it for further more calculations and whatever results are obtained are printed. Every single program follows more or less this pattern. When the = is executed, every instruction by that time must have been executed. Means a = cannot work unless input is received. Input cannot received unless float declaration is over and unless int declaration is over we cannot do float declaration. Similarly when it’s a time to execute System.out.printf( ) every instruction by that time must have executed. Because the instructions are supposed to get executed sequentially whenever we do not indicate in which sequence they should get executed. In general, the sequence control instruction happens to be the default control instruction. Whenever we do not mention, the instructions are get executed sequentially or one after the other. But in every Java program we do not want this. We may want that after execution of a =, b = would not get executed but the control should go to the System.out.printf( ) or after declaration of int and float and receiving of input, we may straight away want to jump to System.out.printf( ). Means we do not want instructions in the program would get executed sequentially or one after the other. If we want that instructions should not get executed sequentially we can make use of decision control instructions.

Decision Control Instruction

First line is the comment which indicates the aim of the program. Name of the class is totalexpenses. In main() first define a BufferedReader b and to receive the input, we say new BufferedReader ( new InputStreamReader ( System.in ) ). Using b, read the values of quantity and price i.e. str = b.readLine( ). The quantity entered later on can be separate out by Integer.parseInt ( str ). Then again read price using str = b.readLine( ) and separate out by Float.parseFloat ( str ). We are displaying only one message and reading two numbers quantity and price. First the quantity is read and convert
the integer string into int using Integer.parseInt(str) and then price is read which is a float string and convert it into float using Float.parseFloat(str). So we have the quantity purchased and the price at which it is purchased. Now we should write the instructions to decide what should be the expenses.

Slide 7

If the quantity purchased is \( \geq 1000 \) then we wish to offer 10% discount otherwise we do not wish to offer any discount. Means the discount should offer or should not offer it depends upon the value that we had supplied at readLine() in the quantity. Means we have to take a decision at the execution as to discount should be given or should not be given. During the execution of the program, the value will be supplied for the quantity. Hence, this decision is taken during execution of the program that the discount should be offered or not offered. Hence, within main( ) after the quantity and price is received, we would now try to take a decision. The way to take a decision is a keyword known as if. if is always followed by a pair of parenthesis and inside the pair of parenthesis we are supposed to write a condition. In the program the condition is \( \text{qty} \geq 1000 \). If the quantity is found that it is \( \geq 1000 \) then we are proposed to give a 10% discount. So we would say dis = 10 where dis is an int. else means the quantity is not \( \geq 1000 \) then we do not propose to offer any discount. So we would say dis = 0. Then calculate the totexp by saying \( \text{totexp} = \text{qty} \times \text{price} - \text{qty} \times \text{price} \times \text{dis} / 100 \). Finally print the value of total expenses using System.out.println("Total exp = Rs " + totexp) and it test for different values. This program is different than the previous one because we cannot say that before execution of = every single statement above it must have been executed. This is so because we are using if else. Means either dis = 10 will work or dis = 0 will work. This program uses sequence control instruction as well as decision control instruction as some part of program executed sequentially and only the discount offer part is executed by making use of a decision control instruction. We can use any decision control instruction i.e. different control instructions within a program. We can use multiple control instruction within the program.

Tips

if and else both are the keywords. General form of if is as follows.

\[
\text{if ( condition )}
\]

\[
\text{statement1 ;}
\]

\[
\text{else}
\]

\[
\text{statement2 ;}
\]

if the condition is true then execute some statements otherwise execute some other statements. The condition that we write along with an if is build using \(<, >, \leq, \geq, =, !=\) which together are known as relational operators. They are known as relational operators because they help to test relationship between two entities. Now when we say \(a = b\), we are assigning the value of \(b\) to \(a\) and when we do \(a \equiv b\), we are comparing the value stored in \(a\) with a value stored in \(b\). So \(\equiv\) is used for comparison whereas \(=\) is used for assignment of the value present in a variable at right hand side to the variable at left hand side. So at the end of \(a = b\), both \(a\) and \(b\) will enjoy the same value whereas at the end of \(a \equiv b\) whatever values present earlier in \(a\) and \(b\) are continued with that values only.

One More Way

Same program can be done in one more way. Within main( ) first receive quantity and price, define the variables suitably and then we would check whether \(\text{qty} \geq 1000\). If \(\text{qty} \geq 1000\) then we would offer discount of 10% by saying \(\text{dis} = 10\) where \(\text{dis}\) is an int. We have not written anything else. Since we do not write anything else, we do not write even else. Then find the totexp using \(\text{totexp} = \text{qty} \times \text{price} - \text{qty} \times \text{price} \times \text{dis} / 100\) where \(\text{totexp}\) is float and print it. So from this we can make a conclusion that else block is optional. Means if we need it then use it otherwise no need to write it in a program. If we use else then it is mandatory to use an if whereas if we use if then we may or may not use else. So if there is an else, there must be an if and if there is an if there may or may not be an
else. It is important to initialize the value of dis = 0. Unless we initialize the value to 0, the program will not work correctly especially when the condition fails. Everywhere we cannot drop the else block. Only in some cases we can drop it.

**Slide 10**

Now we want to calculate the gross salary earned by an employee. Begin with main( ). In main( ) declare variables and receive the value of basic salary. Then we would say if ( bs >= 1500 ) then calculate da which is 95% of basic salary. Then calculate hra which is 20% of basic salary whereas ca is 12% of basic salary. In else i.e. if basic salary < 1500 then da is 92% of basic salary, hra is 15% of basic salary and ca is Rs. 200. The gross salary, gs = bs + hra + da + ca. Finally print the gross salary. When this program is executed, it will report error. Because the compiler believes that we had written a program as shown in the box in the slide. Means when if ( bs >= 1500 ) then only da will get calculated. hra and ca do not belong to the if block. Means even if the condition turns out to be true, all the statements do not get executed; only da is calculated. In this program, the else is a hanging else because after if and da, we have two expressions hra and ca and then suddenly we have an else. This else does not match with if. So the compiler does not understand this form. The hanging else is never allowed. Every else must have to match with some or the other if. Only the first statement after if belongs to the if block similarly only the da belongs to the else block. Just indenting the three statements to the right does not mean that all they belong to if or else block. To rectify this situation the correct program is to next.

**Correct Program**

We need to put the da, hra and ca of the if block within a pair of braces. Similarly the da, hra and ca of the else block also go within a pair of braces. The pair of braces are scope delimiters. They delimit the scope of if and the scope of the else. They indicate that how many statements belong to if and how many belong to else. If we do not use braces then only the first statement after if and the first statement after else belong to if and else block respectively. Since we want multiple statement belong to if and multiple statement belong to else then we have to use pair of braces to indicate how many belong to if and how many belong to else.

**One More Form**

So we know one more form in which if can be written.

```java
if ( condition )
{
    statement1 ;
    statement2 ;
}
else
{
    statement3 ;
    statement4 ;
}
```

Default scope is the next statement after if and the next statement after else. To override the default scope, we need to make use of pair of braces. There is no one to one correspondence between if and else block. There can be 100 statements in if block and 2 statements in else block.
In this lecture you will understand:

* Program – Leap year or not
* Program – First day of any given year
* Nuances of Modulus Operator
* Nuances of Comparison Operator
Leap Year or Not

To determine whether a year is a leap year or not, create a project in NetBeans. Give the name of the class as LeapYear. Then we have main( ) inside the class. Inside main( ) we would ask the user to enter a year. After that we can receive the year using the procedure which uses BufferedReader object. Then using str = b.readLine( ) and convert the string into an integer using y = Integer.parseInt ( str ). After receiving the year, we have to determine whether the year is a leap year or not. For example, if the year is 1996 then we can say that it is a leap year because on dividing 1996 by 4 we get the remainder as 0. Likewise if we take the year as 2000, then on dividing it by 4, we get the remainder as 0 so we can conclude that the year is a leap year. Now suppose if the year is taken as 1900, then on dividing it by 4, we get the quotient as 475 and the remainder as 0. But 1900 is not a leap year. Division by 4 is not enough to take a decision that the year is a leap year or not. If the year happens to be a century year i.e. year like 1900, 2000, 2100, 2200, etc. then we need to check the division by 400 rather than 4 to determine whether that year is leap year or not. For example, 1900 is not a leap year because 1900 is not completely divisible by 400. 2000 is completely divisible by 400 so it’s a leap year. So for a century year we have to check division by 400 but if the year is non-century year then check the division with 4.

First Day of Any Year

Now we want to determine the first day of any given year. For that create a Java application in NetBeans and give the name of the class is FirstDay. In main( ) first define the year, y as int. Once year is defined, we can give a message to enter year using System.out.println( ) and then receive the year. Once the year is received, now we can actually write the logic for determining the first day of the year whether it is a Monday, Tuesday, Wednesday, etc.

The Logic Behind It

For example, suppose we supply the year as 2002. Then we want to determine what is the 1st January of 2002. The logic behind this is as follows. 1/1/1 i.e. 1st day of the 1st month of the 1st year means 1st January 1 that was Monday. If 1st January 1 was Monday then 8/1/1 was also Monday i.e. adding 7 to 1, we get 8/1/1. So again we get 15/1/1, 22/1/1, 29/1/1 all are Mondays. We are going on adding 7 to the previous day. If we go on doing this, till the time we will reach 1/1/2002 we will be able to
determine what is the coming day i.e. 1/1/2002. Adding 7 in this manner become too tedious which is
not the correct way to do this. So the correct way to do this is as follows. 1/1/1 was Monday then
29/1/1 is also a Monday. We are doing calculation that we are counting the days from 1/1/1 upto the
day prior to 29/1/1. A day prior to 29/1/1 was 28/1/1 means we are actually going to count days from
1/1/1 upto 28/1/1 and those are 28 days. So there are 28 days. 28 days turns out to be 4 weeks. But we
will do 28 % 7 instead of 28 / 7 which results to 0. Means days from 1/1/1 to 28/1/1 were such days
that all of them are evened out to be weeks. Hence the remainder is 0. If all the days are evened out to
be weeks then the coming day i.e. 29/1/1 must be a beginning of a fresh week and hence 29 is a
beginning of a fresh week so it’s a Monday. To determine 29/1/1, we counted days from 1/1/1 to
28/1/1 which are 28 days and they are evened out to weeks as 28 % 7 = 0. Hence the coming day i.e.
29 is a beginning of a fresh week i.e. Monday. Now we have to extend the same logic to 1/1/2002.
Means we have to determine the days from 1/1/1 to upto the day prior to 1/1/2002. A day prior to
1/1/2002 is 31/12/2001. So we have to count days from 1/1/1 to 31/12/2001. Whatever the days
turned out, we will try a % 7 on it. When we do % by 7 then we will get some remainder from 1 to 6.
If remainder is 0 then we can say that all the days are evened out into weeks. Hence the coming day
i.e. 1/1/2002 must be a beginning of a fresh week i.e. Monday. But it might give the remainder as 0 or
might the remainder from 1 to 6. If number of days % 7 turned out to be 1, then everything is evened
out but 1 is not evened out. That 1 day must be Monday hence coming day i.e. 1/1/2002 must be the
second day of the week i.e. Tuesday. So if remainder is 0; coming day is Monday, if remainder is 1;
coming day is Tuesday, so on. Now we implement this logic in program.

Slide 7

Inside LeapYear class, within main( ), we would first receive the year. Once the year is received, we
would try to determine the first day of the year. We have to determine the number of days from 1/1/1
to 31/12/2001 assuming that the year entered is 2002. From 1 to 2001 there are 2001 years. In each
year there must 365 days. Some of the years are of 366 days but every year had at least 365 days. So
for calculating number of days from 1 to 2001, we have to do 2001 * 365 which are called as
normaldays but it is assumed that the year entered is 2002. If the year received is 2005 then we have
to count up to 31/12/2004. So in general, we can say normaldays = ( y – 1 ) * 365. So this accounts to
365 days in a year. Now we try to account for every extra day that is in the leap year. Leap year has
366 days. Every leap year offers one extra day. So if we say that between 1 to 2001, there are x leap
years then there are x extra days which are leap days. So how many leap days are there is directly
dependent on how many leap years are there between 1 to 2001. leapdays = ( y - 1 ) / 4 - ( y - 1 ) /
100 + ( y - 1 ) / 400. Leap year occurs in every 4 year. So in 1 to 2001 there are 2001/4 leap years are
present. Year is 2002. So ( y – 1 ) make it 2001. So we can say 2001/4, those many extra days are
present there or leap days are present there. But this is partially true. As 4 is a leap year then 8 is a
leap year, 12 is a leap year, and so on 96 is a leap year but after that 100 is not a leap year as 100 is
not divisible by 400. So after 96 instead of 100, 104 is a leap year. Again 200 is not a leap. So by
doing ( y – 1 ) / 4, we are already counted years like 100, 200, 300, etc. to be leap years which is
wrong. So, we can uncount them by subtracting all these century years. So if year is 2002 then ( y – 1
) / 100 i.e. ( 2002 – 1 ) / 100 = 20. So 20 extra years we had counted, so we can uncount them by
deducting. But in this process we had also deducted years like 400, 800, 1200, 1600 and 2000. All of
them are divisible by 400 and hence leap years. So now we can recount them by adding back ( y – 1
) / 400. So we can say that totaldays = normaldays + leapdays. We have days from 1/1/1 to
31/12/2001. Since we know totaldays, we can say firstday = totaldays % 7. Now if firstday = 0 then
we can conclude that all the days from 1/1/1 to 31/12/2001 are evened out into weeks. So the next
day is the beginning of the fresh week i.e. Monday. But we are not guaranteed that the firstday will
be 0. So we can use decision control instruction and check that out. If firstday is 0 then it is Monday
and write at least 6 ifs for checking different days i.e. 1 for Tuesday, 2 for Wednesday, so on. This
way we can determine first day of any given year.

Are You Sure?
Now we will take a look at the nuances of the % operator and the == operator. If we do 4 % 3, it will give the remainder as 1. 3 % 4 will give 3. -3 % 4 gives -3. 3 % -4 gives 3. -3 % -4 gives -3. When all the expressions are evaluated, sign is not taken into account. Only 3 % 4 is really done which gives us 3. With this 3, the sign of the numerator is always associated i.e. the remainder will always take the sign of the numerator. In -3 % 4, numerator is negative hence remainder is negative. In 3 % -4, numerator is positive hence remainder is positive whereas in -3 % -4, numerator is negative so remainder is also negative. Now if we have a, b and c with initial value as 5 and then we check whether a is same as b and b is same as c and if so then we will print Hello otherwise we will print Hi. It appears that a, b and c all are 5 so Hello would get printed. When we say a == b == c, b is not get compared with c. a == b gives true, when it turns out true, that true cannot be compared with c. So we will get an error when we do this. To print Hello, we have to write two conditions a == b and b == c and then combine these two separate conditions using logical operators.
Logical Operators

In this lecture you will understand:

* Nuances of decision control instruction
* What are logical operators
* When to use them
* Hierarchy of operators
What Will Be The Output

Suppose there is a package called sample which contains a Sample class and a void main( ). Suppose we have two variables a and b with initial values like 5 and 10 respectively in main( ). We check a condition if ( a >= 20 ). Then we do b = 30 and then print the value of b. After if ( a >= 20 ) we had given a semicolon. The moment we put that semicolon, the Java compiler believes that if ( a >= 20 ) then the semicolon be executed whereas b = 30 is completely out of scope of that if. if condition is true then execute the semicolon. Then execute b = 30. If the condition is false then straight away execute b = 30. It means b = 30 is get executed any way. So this is the reason that on execution we would find the output is 30. The ; is a null statement and its job is to do nothing but it’s a perfectly legal Java statement. So, if we tend to give a ; at the end of every statement, we also tend to give it at the end of if; that results into a problem. Because the condition is true then we do not want to do anything and if we not really want to do anything then there is no need to check the condition. So, writing the ; at the end of if, even though grammatically it might be correct but logically its wrong.

What Will Be The Output

We have again a Sample class inside which we have main( ). Inside main( ) we have a = 10 and then check the condition if ( a = 5 ). If the condition is true then print Hi. The program looks correct but when we do so, we get an error. Because whenever we write an if inside that a boolean expression is expected whereas we are giving an expression statement like a = 5. Unless there is some relationship that is being tested within an if as it’s a relationship which would results into a bool. Condition would be true or false that would happen when we check a condition. So a = 5 would not work.

Slide 5

Suppose we have a package called examination inside which we have a class called Examination. The package is present there when we create a new project. In the class we have a main( ). Inside main( ) we are trying to receive the marks obtained by a student in 5 different subjects. Collect them in m1, m2, m3, m4 and m5 where all of them are integers. Calculate the percentage. Assuming that the maximum marks that can be scored in any subject is 100. Percentage marks will same as average. Add marks of 5 subjects and divide by 5 to get the percentage or add marks of 5 subjects, divide by 5 and then multiply by 100 to get the percentage. per is an integer and then if ( per >= 60 ) and if it is true then print First division indicating that a student has secured a 1\textsuperscript{st} division. If it is false then we have to check several conditions. Inside the else block, we check if ( per >= 50 ) and if it is true then print Second division indicating student has managed to get the 2\textsuperscript{nd} division. It is not necessary to check whether per < 60 because it is implied that per < 60 as we are in else block. If this condition also fails then we will land in the else block and in the else block we again write several statements. Inside this we will check if ( per >= 40 ) and if it is true then print Third division indicating the student has secured a 3\textsuperscript{rd} division. Once again we are not checking whether per < 50 because as we land in else block, it itself indicates that per < 50. If this condition if ( per >= 40 ) also fails then we fail a student by printing Fail. The program works correctly. However, this is only a workable program but this is not the correct way to write it. Because with this program there are at least 3 problems associated. They are as follows.

- Matching too many ifs and elses. As here are three conditions so there are three pairs of ifs and elses. If there will be 10 conditions then we will have to manage 10 pairs of ifs and elses which is a tedious.
- if ( per >= 60 ) then we are printing First division. Here is a single statement so ok but if there are again numbers of statements then we will have to use pair of braces for the scope of the multiple statements. So demand for pair of braces will increase as number of statements increases.
- As the number of conditions goes on increasing, programs indentation level also go on increasing. So the program shifts to right and if the program reports an error, it is difficult to find the error as the entire program is not visible on the screen, it is shifted to right. So it becomes difficult to locate where we had committed the mistake.
Slide 6

Again we have a class Examination and in main() we have 5 variables which receives marks obtained by a student in 5 different subjects. Then calculate the percentage. The first condition is if ( per >= 60 ) and if it is true then print First division. Now instead of writing else we are writing another if here as else is optional. Now we check if ( per >= 50 ) and if it is true then print Second division. Again we drop else and write if ( per >= 40 ) and if it true then print Third division and this fails then write else which prints Fail. This program seems to be ok as not too many ifs and elses, not too many braces and not too much indentation level. But the program is wrong. For example, if someone scores 65% marks i.e. average obtained by adding m1, m2, m3, m4 and m5 is 65. For 65% marks, per >= 60 would be true. So, it will print First division. After execution of this the control goes to if ( per >= 50 ) and the condition gets satisfied and the program also prints Second division. Then it reaches to if ( per >= 40 ) and the condition also gets satisfied and the program prints Third division. Means if somebody scores more marks and pass in a good division, the program reports to pass him / her in all the divisions. So the program appears to be ok but logically it is completely wrong. If somebody score 65 marks then he / she has to pass only in the 1st division. So we need rectify this.

Slide 7

Till the calculation of percentage the program is same. Again we check if ( per >= 60 ) and if it is true then print First division. After that we would try to combine two conditions using the logical && operator. The && operator evaluates these two conditions and if both turn out to be true then the whole thing is true. If any one of the condition is false then the whole thing is false. For 65% marks 65 >= 50 would be true but 65 < 60 is false. Hence Second division would not be printed. But if somebody scores 55% marks then 55 will be > 50 and 55 < 60 then we can correctly allot a Second division. Then we write the next condition that if ( per >= 40 && per < 50 ) and if it is true then print Third division. If 1st condition fails control goes to next if. If 2nd also fails, it jumps to the 3rd if and if that also fails then in the else block we print Fail. The program look correct but something is still wrong with this program. For example, per = 65. 65 >= 60 is true so First division is get printed. After that control goes to second if. When 2nd if is evaluated 65 >= 50 is true but 65 < 60 is false which turns out to be false and since it turns out to be false we will reach the 3rd if. In the 3rd if, 65 >= 40 is true but 65 < 50 is false. So again it becomes false. Since it is false we will land in the else block and print Fail. Means if somebody scores 65% marks, the program passes him / her in 1st division as well as fails him / her in the examination. If we are using logical operators to check the ranges, forget the else block. So we need to rectify this.

Slide 8

Again till the calculation of percentage and the first three conditions of the previous program will remain same. Now instead of saying the else block, we should specifically check if ( per < 40 ) or not. If it is < 40 then print Fail. This is much better than what is done earlier. It does not increase the indentation level too much; there are no ifs and elses to be matched. Moreover pairs of braces are not present here. Even if we have multiple statements in each of these ifs, the pair of braces would be manageable. So as compared to the previous one this is a superior program.

One More Way

In this way calculate the percentage. Check the condition if ( per >= 60 ) and if it is true then print First division. Now in place of writing else we should write else if ( per >= 50 ) and if it is true then print Second division. If this condition also fails then again we check else if ( per >= 40 ) and if it is true then print Third division. If this also fails then we go to the else block and then print Fail indicating that student managed to fail the examination. In this case, the else at the end we are writing is working for all the three ifs. If 1st if fails, the control jumps to the 2nd if. If that also fails the control
jumps to 3rd if and if 3d if also fails the control goes to the else block. Means Fail gets printed if all
the ifs going to fail. So else is working for all the 3 ifs. Here also indentation level is manageable,
program does not shift to the right, not too many ifs and elses to be matched. Even if there are pair of
braces, it is easy to be matched. So we can use either logical operators or else if.

Logical Operators

We had seen logical && operator. Apart from this there two more logical operators, one is logical ||
and second is logical ! ( not ) operator. These logical operators are really useful in two situations. One
is in checking ranges and another is in solving yes / no problem. We had seen how to check the
ranges as seen in the examination program. So whenever ranges are to be checked, logical operators
are useful. Logical operators are also useful in solving yes / no problem. Yes / no problem means a
problem in which after having checked a complicated set of conditions the end result is only one or
the other. Means only one of the two possible outcomes is the result of the program but to be able to
arrive at this result we have to check multiple conditions.

Slide 11

Create a class Insurance which contains main( ) in which we receive the age, sex and marital status.
age is an integer whereas sex and marital status both are char. Now we check whether ms contains a
value m or not using if ( ms == ‘m’ ) and if it is true then we can say that the person should be
Insured. Otherwise we can check several conditions. Then we check if ( s == ‘m’ ) i.e. sex is male or
not and if it is true then check if ( age > 30 ) i.e. check the age of a person. If it is true then we would
insure the person otherwise we would not insure the person. if ( s == ‘m’ ) fails means the person
must be a female. In case of a female we check if ( age > 25 ) and if it is true then we insure her
otherwise not insure her. There is something wrong in this program. Means when we do ms == ‘m’
then the program will report an error as m to be an undefined variable. We want m to a constant. We
want to check whether the character constant stored in ms is same as the character constant m. By
putting single quotes around m, we can signify that m is a character constant. As shown in the slide,
the character constants are to be indicated in single quotes. For example, if we supply 26 as age, m
( male ) as sex and u ( unmarried ) as marital status then when we check ms == ‘m’, the control goes
to the else block. In the else block, we check s == ‘m’ which turns out to be true and since it is true
we check age and 26 > 30 would be false hence the control falls to the else block and in the else
block Not Insured is printed out. This program is correct but this is not the best way of doing it.
Because of the same three reasons that are discussed earlier. Programs shifts to the right, too many ifs
and elses to be matched and multiple pairs of braces to be matched. Moreover this is indeed a yes / no
problem because after checking complicated conditions the final outcome is either the person is
insured or the person is not insured. Person is to be insured in the following three situations.

- First situation is if the person is married. If the person is married no matter what is the age
  and sex of the person. We insure the person immediately.
- Second situation is if the person is unmarried, male and above 30 years of age.
- Third situation is if the person is unmarried, female and age more than 25.

In all the other situation we do not propose to insure the person.

Using Logical Operators

Till receiving the input i.e. age, sex and marital status the program remains same. Now check if ( ms
== ‘m’ ) or not. If so then insure the person otherwise do not ensure the person. But this is not the
only situation when we want to insure the person. There are more situations like unmarried, male and
above 30 years of age and third situation is unmarried, female and above 25 years of age. Means the
person is married or unmarried, male, above 30 years of age or unmarried, female, above 25 years of
age. So we separate them using logical || operator. Collect each situation in a pair of parenthesis. But
again when we say `ms == 'u', s == 'm', age > 30` these are again multiple conditions. So combine them using logical `&&` operators because we want that all the three conditions to be true for the person to get insured. Similarly `s == 'f' && ms == 'u' && age > 25` is true for the person to get insured. So this way we can combine multiple conditions using `&&` and `||` operators. This is much better than the previous program. No headache of matching too many `if`s and elses, no headache of matching too many pairs of braces. Moreover program does not shift too much to the right. For example, if 26, m ( male ) and u ( unmarried ) are the inputs given then `ms == 'm'` will fail. But since this condition fails the control cannot goes to the else block, it checks two more conditions. It check `ms == 'u' && s == 'm' && 26 > 30` and the condition fails. So 2nd situation is also false. In the 3rd condition, `ms == 'u' && m == 'f' && 26 > 25` condition fails as input for sex is m and we are checking with female. Now the whole thing goes false so the control goes in the else block and report that the person is not insured. This explains the use of the logical operators in two situations one in which we try to check ranges and second in which we try to check yes / no problem.
Conditional Operators

In this lecture you will understand:

* Working of && and ||
* Short circuit operations
* Variations of decision control instruction
* What are conditional operators
* When to use them
Working of && And ||

If we have two conditions then while evaluating them both might turn out to be true or both might be
false. Again it is possible that 1\textsuperscript{st} is true and 2\textsuperscript{nd} is false and the last possible way is 1\textsuperscript{st} is false and 2\textsuperscript{nd} is true. Whenever we have two conditions, the two ways to connect them together is to say condition1 && condition2 and condition1 || condition2.

• If condition\textsuperscript{1} as well as condition\textsuperscript{2} both are true then the result of condition\textsuperscript{1} && condition\textsuperscript{2} is also true. Also with condition\textsuperscript{1} || condition\textsuperscript{2} if both are true, the result will be true.

• If condition\textsuperscript{1} as well as condition\textsuperscript{2} both are false then the result of condition\textsuperscript{1} && condition\textsuperscript{2} is false. Also when condition\textsuperscript{1} || condition\textsuperscript{2} if both are false, then the result will be false.

• If the condition\textsuperscript{1} is true and condition\textsuperscript{2} is false then the result of condition\textsuperscript{1} && condition\textsuperscript{2} will be false. Also in condition\textsuperscript{1} || condition\textsuperscript{2}, when condition\textsuperscript{1} is true and condition\textsuperscript{2} is false then the result will be true.

• If the condition\textsuperscript{1} is false and condition\textsuperscript{2} is true then the result of condition\textsuperscript{1} && condition\textsuperscript{2} will be false. Also in condition\textsuperscript{1} || condition\textsuperscript{2}, when condition\textsuperscript{1} is false and condition\textsuperscript{2} is true then the result will be true.

Short Circuit Operations

Suppose we have a boolean variable a and then we say a = 2 && 3. We think that this is perfectly acceptable but Java compiler report error for this because 2 and 3 are not definitely truth values. Truth value like boolean, it cannot be integers like 2 and 3. Never think like 0 means falsity and every non-zero number means true. If we have a boolean a and we have two integer variables i and j holding values like 7 and 2 and then we say a = i < 5 && ++j > 6. In this i < 5 is one condition and ++j > 6 is second condition. ++ increments the value of variable j by 1. So it is similar to j = j + 1. When this operation is completed, we print the value of j. We think that 7 < 5 is false but ++j makes j as 3 and 3 > 6 is also false. So value of a will be false. When we print j, it will be 2. We think that ++j increments the value of j and make it 3 and then 3 would be compared with 6. But no such things happen. This indicates that ++j had never gone to work. This is because the fact that i < 5 is turns out to be false. The moment the 1\textsuperscript{st} condition turned out to be false the compiler thinks that there is no need to evaluate the 2\textsuperscript{nd} condition. Because both the conditions are connected using the && operator. If 1\textsuperscript{st} is false, no matter what is the result of the 2\textsuperscript{nd} condition. It might be true or it might be false, the result would any way be false. Hence it does not bother to evaluate ++j > 6. It remains as it is, j does not change. So it is reported as 2. If we want that j should indeed change then the short circuiting should not happen. If we want that both the operations, both the conditions should get evaluated then we need to connect them using a & operator rather than && operator. If now we try to print the value of j then it will print it as 3. Since j is printed as 3 means ++j indeed has gone to work. This is so because we had combined the two conditions using & operator and not with the && operator. Suppose we have two conditions c\textsuperscript{1} and c\textsuperscript{2} connected using logical && operator. Then we have a & operator connecting the two conditions c\textsuperscript{1} and c\textsuperscript{2}. The way there are two operators && and &. Similarly there are two operators for ORing i.e. logical || and |. We have connected the two conditions using both the OR operators.

• In c\textsuperscript{1} && c\textsuperscript{2}, c\textsuperscript{2} will go to work only if c\textsuperscript{1} is true. If c\textsuperscript{1} itself is false then there is no need to evaluate c\textsuperscript{2} because the result would be any way false.

• In c\textsuperscript{1} | c\textsuperscript{2}, both the conditions will always go to work.

• In c\textsuperscript{1} || c\textsuperscript{2}, c\textsuperscript{2} will go to work only if c\textsuperscript{1} is false. If c\textsuperscript{1} is itself true then no need to evaluate c\textsuperscript{2}.

• In c\textsuperscript{1} | c\textsuperscript{2}, both the conditions will always get evaluated.

Hierarchy of Operators
As shown in the slide, all the operators except logical ! operator are similar to each other in some way or the other. The similarity between them is all of them are binary operators. Binary means each operator would need two operands to work with. Unlike that the ! i.e. the logical Not operator is a unary operator. Unary operator needs only one operand to work with. The priority increases toward the top. Means ! operator enjoys the highest priority. Within arithmetic operators *, / and % enjoys higher priority than + and – operator. In general, all arithmetic operators enjoy higher priority than relational operator and relational operator will enjoy higher priority than logical operator. Within relational operators, <, >, <= and >= all of them enjoys same priority but enjoys higher priority than == and != operators. Within logical operators && operator enjoys higher priority than || operator. If we say a = -3 + b – 5 then we are using two – signs here but each – sign is different. 1st one is a unary minus because it is working only on 3 whereas – used 2nd time is a binary minus because it is working on b as well as 5. Unary – enjoys higher priority than a binary -.

Exchanging Blocks

We have a Sample class inside which there is a main(). In main() we have initialized two variables a and b with values like 3 and 4. Then we check a condition if ( a <= b ) and if it is true then we print the message A otherwise we will print B. When this program is executed, we get the output as A as 3 < 4 is true and since it is true, it will print A. Same thing can be also rewritten in another manner. In the 1st case in the if block we print A and in the else block we print B. Now we are reversing this. Wherever there was A, now there is B and wherever there was B, now there is A. In place of if ( a <= b ) we had made change it to if ( a > b ). Means we are exchanged the if block with the else block and to keep the logic same, to keep the functionality same we had reversed the condition. Instead of saying if ( a <= b ) now we are saying if ( a > b ). The output remains same. When we check if ( 3 > 4 ), it becomes false and since it is false the control goes to the else block and A gets printed out.

One More Way

Same can be done in one more way. main() will remain same. In the condition if ( a <= b ) since 3 <= 4 is true A gets printed out. If the condition not gets satisfied then B is expected to be the output. So the output is A. If we do not want to write if ( a <= b ). We do not want to exchange the if block with the else block as done in the previous slide. We want to keep println ( "A" ) in the if block and println ( "B" ) in the else block. If we do not want to say if ( a <= b ) then the same condition can be written in another way as if ( ! ( a > b ) ). ! is a logical Not operator. ! will work on a result of a condition. If the condition evaluates to truth then ! makes it false and if the condition evaluates to falsity then ! makes it true. In the example, a = 3 and b = 4 then a > b i.e. 3 > 4 evaluates to false and ! false makes it true. Since it results into true, A will get printed out. Parenthesis around a and b are necessary. Otherwise the ! operator will get apply on a rather than the result of ( a > b ).

Yet Another Way

Same can be done in one more way. Here we check the condition if ( a > b ) and if it is true then B gets printed out otherwise A would get printed. Instead of saying if ( a > b ) we say if ( ! ( a <= b ) ). a = 3 and b = 4. So a <= b i.e. 3 <= 4 evaluates to true and ! true evaluates to false. This result will take the control to the else block and the result A will get printed. This shows the use of logical ! operator.

Cond. Operators

The program is same. In place of printing out something in if and else block, we are assigning a value 10 or 20 to the variable c. Same thing can be done using conditional operators. We say if ( a <= b ) then assign 10 to c otherwise assign 20 to c. Using conditional operators this is written as a <= b ? c = 10 : c = 20. The ? : are known as conditional operators. We can also say c = a <= b ? 10 : 20. The 1st way to use conditional operators is wrong whereas 2nd way is correct. Means whatever is the result is assigned to some or the other variable. Unless it is assigned to some variable we cannot make use of ? and :. The ? and : are the only ternary operators available in the Java programming language.
Every other operator is either a unary operator or a binary operator. We have \( c = a \leq b \) is one argument, 10 is 2nd argument and 20 as the 3rd argument. Whatever we had done in 4 lines code earlier, now it is achieved in 1 line by the use of conditional operators.

**How Would You Convert**

Suppose inside main( ) we have variables \( a = 3, b = 4 \) and \( c \). Then we check if \( (a \leq b) \) and if it is so then 10 is assigned to \( c \). We do not have any else block here. The same thing can be done using conditional operators. We would say \( a \leq b ? c = 10 \). If we execute this, it will report an error. This is so because whenever we use conditional operators we do not have a choice of dropping the \( : \). We had that choice with if else since else block we know it is optional whereas \( : \) is not optional. If we use \( ? \), we have to make use of \( : \). There is no choice. We can eliminate this error.

**Remove The Error**

To eliminate the error in the previous program, then as shown in the slide we have not dropped the \( : \). It is written after \( c = 10 \). So \( : \) is necessary. But even this results into an error because \( ? : \) are very adamant operators. If we use one, we have to use the other and if we use 2nd, we have to use the 1st one. In other words \( ? \) and \( : \) will go hand in hand. Now if we used \( ; \), then it says that not only the \( : \) is necessary but there must a statement in the \( : \) part. But our logic of program does not demand any statement but \( ? : \) demand some statement in the \( ? \) part and some in the \( : \) part. Our functionality says that if the condition is false then do not do anything. We know that if we do not want to do anything then we do that using a \( ; \); i.e. the null statement. So now we are saying \( a \leq b ? c = 10 ; ; \). 1st \( ; \) we think will work like a null statement whereas 2nd \( ; \) would work like a terminator whereas in actuality it is exactly opposite. The moment we type 1st \( ; \), it starts acting like a terminator and then the compiler will complain saying you have to put something between \( ? \) and \( : \) but it is not get done. We have not put anything between \( ? \) and \( : \); because it treats 1st \( ; \) itself as a terminator. To eliminate this we should say \( c = a \leq b ? 10 : 0 \). This indicates that we have to assign some dummy value to \( c \). In this case we are using that dummy value as 0.

**Would This Work**

Suppose we say \( d = a \leq b ? c = 10 : c = 20 \) and then print the value of \( d \). This would work perfectly. If the condition \( a \leq b \) is true then \( c = 10 \) would go to work otherwise \( c = 20 \) would go to work. Whatever value \( c \) takes either 10 or 20, it would later on be assigned to a variable \( d \) on the left hand side. Suppose we have \( a = 2 \) and \( b = 4 \) and then we say \( a \leq b \ ? \ System.out.println ("Hi") : System.out.println ("Bye") \). If we try to execute this, it will not work because println( ), readLine( ), etc. all these things are not allowed in \( ? : \). In general we can say, we have expression1 \( ? \) expression2 : expression3. Moreover expression2 and expression3 should be of same type which is a requirement of \( ? : \). Means we attempt to say that \( c \) be either equal to 10 or 20.5 would be wrong because 10 is an integer and 20.5 is a double. So their types are not same. Either both of them will have to be double or both of them will have to be integers.

**Moral of The Story**

- If we use one operator \( ? \) then we have to use \( : \) and if we use \( : \) then we have to use the \( ? \); i.e. both of the operators would always go hand in hand.
- We may be required to write dummy expression the way we had initialized \( c \) to a value 0 in the \( : \) part.
- \( ? : \) are not replacements for if else. If at all they are replacements, they are replacements in very limited sense. Means in the \( ? \) and \( : \) part, there would have to be only 1 statement. If we have multiple statements, \( ? : \) will not go to work. Both the expressions are of same type in \( ? : \).
- Never nest the \( ? : \) operators. Means we cannot say \( d = a \leq b \ ? ( e = b > a ? 10 : 23 ) : 20. \)
Conversions And Comparisons

In this lecture you will understand:

* Conversion between int and float
* Conversion between char and float
* Representation of ints and chars
* ASCII codes and ASCII values
* Comparison of real values
Conversions

For example, in main() inside a class Conversions two integer variables a and b are declared with values 10 and 20. Then we define two floats c = 3.14f and d = 6.8f. The System.out.printf ( "%f %d", a, b ) is written to print values of a and b but there is a problem in the format specifiers that are used. Since a is an integer ideally we should print it using %d but we used %f for it whereas b which is an integer we had correctly used %d for it. Then we had written System.out.printf ( "%d %d", c, d ) for printing c and d. Both c and d are floats but we are trying to convert them into integers using %d specifier. When we attempt to execute this program we may expect the answers as a = 10.0, b = 20, c = 3 and d = 6 or d = 7. But when we execute this, it will report an error. Because we are not suppose to use printf() to do conversions but printf() is used for printing in some formatted manner. Expecting that printf() should do the conversion from int to float or a float to int is a wrong expectation. printf() is never used for conversion. To carry out narrowing conversions we have to use type casting whereas broadening conversions will happen automatically. When we execute this program we get IllegalFormatException conversion exception which indicates that printf() cannot be use to convert from int to float or a float to int. For example, if we want to carry out conversion from float to an int then suppose we have a variable called float d = 6.8f. We are trying to narrow it down to an int so a typecasting operation is used. Float can be converted to an int by casting it into an int. If we do z = d will report an error by saying narrowing conversions like this are not allowed. So the correct way is to put within a pair of parenthesis the type into which we want to convert. d is a float and want to convert into an int. Hence it’s a narrowing conversion. So we write z = ( int ) d. Now if we want to convert an int to a float which is a broadening conversion. Hence a = 10 which we want to convert it into float must assign it to a float using z = a. This is a broadening conversion. So 10.0 is get stored in z. So do not use printf() to convert int to float or float to int.

Conversions Continued...

In the Conversions class we have two integer variables i = 65 and j = 90 and two char variables ch = ’A’ and dh = ’Z’. Then we try to print them using System.out.printf ( "%d %d %c %c", i, j, ch, dh ). i and j are integers so we have used %d whereas ch and dh are chars so we have used %c for printing them. In the next System.out.printf ( "%c %c", i, j ) we have mismatch. Since i and j are integers, we have to use %d for them but we are using %c for printing them. In the System.out.printf ( "%d %d", ch, dh ) there is also a mismatch. ch and dh which are chars have to be printed with %c but we are printing them using %d. When we execute this program, 1st printf() will print the output as 65 90 A Z which is as we expect. But when the 2nd printf() get executed, it prints A Z whereas for the 3rd printf() we get output as 65 90. i and j are 65 and 90, their characters equivalent are reported as A and Z whereas A and Z are characters so their integer equivalents we get as 65 and 90. It means char to int and int to char conversions are seem to be predictable and hence reliable. int to float was not reliable using printf() but char to int and int to char conversions printf() can do reliably.

Representation

Suppose we take some variables int i = 65, float a = 3.14 and ch = ’A’. Some memory locations will be allocated for these three variables and when we try to store 65 in i, actual 65 will not get stored instead binary equivalent of 65 would get stored. Binary equivalent of 65 is 01000001. We can convert this binary into its equivalent decimal using 1 * 2^6 + 1 * 2^0 = 64 + 1 = 65. Now we want to convert 3.14 into its binary some 0s and 1s get stored in a. We are not interested in the method that how to convert 3.14 into its binary but whatever method is used some 0s and 1s get stored in a. However in char A, its ASCII code is used. Off late in place of ASCII code, Unicode is used. Basic difference between ASCII code and Unicode is, the ASCII characters are always 8-bit characters whereas in Unicode every character is a 16-bit entity. ASCII can be used only for specific number of characters i.e. only for 256 different characters because if every character is a 8-bit character then maximum combinations can be made is 2^8 = 256. But there are so many languages and so many scripts where 256 characters are not enough. We need more characters so Unicode character system came into existence where each character is represented using 16
binary digits rather than 8 binary digits. But ASCII and Unicode equivalents of characters like A, B, C, etc. are same. Whether its ASCII or Unicode, A cannot be converted into its binary by dividing by 2 or multiplying by 2. So for A either Unicode is used or ASCII code is used. Finally in ch 01000001 would get stored. The 01000001 got of int i = 65 and the 01000001 got of char ch = 'A' are turned out to be same. In both cases in i and ch 01000001 would get stored. Methods for obtaining their binary were different in different cases but final binaries are same.

**Methods Are Different**

Suppose we take int i = 65 and char ch = 'A'. i will occupy 4-bytes as integer occupies 4-bytes. Rightmost byte would be the lowest byte leftmost would be the highest byte. In left three bytes all 0s get stored whereas in the rightmost byte the binary of 65 i.e. 01000001 would get stored. In ch, in the rightmost byte 01000001 would get stored where ch is a 2-bytes entity. If we attempt to print out i using %c we mean that print the character equivalent of 01000001 which in the lowest byte and 00000000 in the next byte. Means out of the rightmost 2-bytes or rightmost 16-bits, the character equivalent of that is printed whereas when we print ch using %d means corresponding to the two bytes present in ch, print the equivalent decimal number. Using 1st printf( ) we get A whereas using the 2nd printf( ) we get 65. When we say System.out.printf( "%c", i ) only lower 2-bytes would get used for printing the character equivalent to present in the 2-bytes. So int to char and char to int conversions are perfectly reliable.

**Surprised?**

Suppose we have float a = 0.7f. Then check if ( a < 0.7 ) and if it is so then print A otherwise print B. Output of this is A. We expect that a = 0.7 and 0.7 cannot be less than 0.7 so condition fails and the output would be B. Suppose we have float a = 5.375f. Then if we again check if ( a < 5.375 ) and if the condition is true then print A otherwise print B. In this case the output is B. Both the programs are same but in place of 0.7 we are using 5.375 in the 2

**Binary of A Float**

Now if we take binary equivalent of 0.7, it is a recurring binary i.e. a never ending binary. If we try to treat 0.7 as a float then it will be truncated after 32-bits because recurring thing cannot go on and on. Float is ultimately a 4-byte entity. Hence it will be truncated, the moment we get 32-bit float. As again, if we try to treat 0.7 as double, then we will stop the recurring process the moment we have constructed a 64-bit binary equivalent. Both the binary equivalents are recurring. We stop float at 32-bits because float is a 4-byte entity whereas double will be stop at 64-bits because double is a 64-bit entity. Now if we compare 32-bit recurring binary equivalent of 0.7 with 64-bit recurring binary equivalent of 0.7 then in that case 32-bit recurring binary equivalent of 0.7 will turned out values wise less than 64-bit recurring binary equivalent of 0.7. For example, if we compare 1.333 with 1.3333333. In normal arithmetic, both of them are 4 / 3 which is a recurring number. If we compare both, 1

**Safety**

Now in 1

\[ a \] case while comparing, we try to compare a with 0.7f. Likewise in the 2

\[ a \] case a is compared with 5.375f i.e. float. When we compare a with float instead of double as in the previous case, we get output as B. This is so because 0.7 i.e. float is < 0.7 which is also float i.e. both are float
and binary of both will stop at 32-bits so value wise both are equal and the condition fails so the output is B. Same happens with 5.375. 5.375 which is a float is < 5.375f and the condition fails so the output is B. Now in place of a float, we can use double in both cases. When we do so, we get the output as B. This is so because double is a 64-bit entity. So when we compare two doubles, binary of 0.7 is equal to binary of 0.7 which is a double by default. Hence the condition fails, control falls in the else block and print the output as B. Again binary of 5.375 is same as binary of 5.375 which is double by default. Hence condition gets failed and the control reaches in the else block printing output as B. In this case it does not matter whether the number is recurring or non-recurring since both are double i.e. 64-bits. They value wise become exactly same. So such output is obtained. So never compare a float with a double. Always compare two floats or two doubles.

**ASCII Codes**

For character A, its ASCII code is 01000001. Likewise, there are codes for B, C, D, E, etc. They are known as standard codes because whenever we hit any key, same code would get generated. So standardization is necessary. It is difficult to remember the binary of any characters. We are not comfortable with binary equivalent of any characters but we are comfortable with decimal. So to remember the ASCII codes, convert them into their decimal equivalent. If A is 65, then B is 66, and so on. Decimal equivalents are commonly known as ASCII values. So ASCII code means binary equivalents whereas ASCII values are their equivalent decimals.

**Characters to be Represented**

We need ASCII codes for A to Z. So we need 26 codes. In addition to this, there are 26 a to z characters which we need to be represented. ASCII codes used for capital letters and that for small case letters are necessarily to be different. We can also represent 10 digits using ASCII codes. In addition to these, there are 32 special symbols on the keyboard. We also have 34 non-printable characters which are known as control characters and we have 128 different graphics characters. So it makes 256 different characters. These characters will range in their ASCII codes from 00000000 i.e. 0 to 11111111 i.e. 255. 0 to 255 are the ASCII values corresponding to 256 different characters.

**ASCII Values**

ASCII values of A to Z range from 65 to 90. Similarly ASCII values of a to z range from 97 to 122 whereas digits have ASCII values in the range 48 to 57. The ASCII value range of non-printable characters will range from 0 to 33 whereas graphic character will range from 128 to 255. The ASCII value for Tab is 9, Enter is 13, Escape is 27 and Space is 32. In general, all characters will range from 0 to 255. We are not trying to remember special symbol’s range because it is not a range but a series of range. In the range of 0 to 255, special symbols do not have a particular range but they are distributed in this range and forms series of range. So they form different slots and it is difficult to remember different slots. So as given in the slide, whatever does not fall in any of these ranges, forms a special symbol.
Repetition Control Instruction

In this lecture you will understand:

* Repetition Control Instruction
* General form of while
* Incrementation & Decrementation operators
* Variations of ++ and -- operators
Control Instructions

We had seen Sequence Control Instruction which is a default control instruction. We also had seen a decision control instruction and these instructions are implemented using two keywords if and else. We also made use of conditional operators i.e. ? and : which are ternary operators. We also learnt some relational operators like <, >, <=, >=, == and !=. They test relationship between two entities and they are binary operators. We also learnt three logical operators, &&, || and !. ! is the unary operator whereas && and || are binary operators. We also have & and | operators. In case of && and || operators, the condition evaluation is abandoned the moment the fate of the condition gets decided. So every condition connected using && and || operators may or may not be evaluated whereas with & and |, we can force every condition to get evaluated which does not happen with && and ||. Now we are taking a look at repetition control instruction or loop control instruction.

To Begin With…

Create a class Loop which contains main() and inside main() give the message as System.out.println ("Enter values of p, n, r"). Then receive the values of p, n and r using BufferedReader, InputStreamReader and System.in. Calculate simple interest as si = p * n * r / 100. Whatever value of si is obtained, print it using System.out.println ("Simple interest = Rs " + si ). p and n are integers whereas r and si are floats. To execute this, we will first do F11 to compile the program and then F6 to execute it. This program will work for one set of p, n and r. si will get calculated and then printed out. If we require this program to get executed for one more set of p, n and r then we have to execute this program once again i.e. we have to hit F6 once again, once again get the fresh values of p, n and r, calculate si and then printed out. If we want to execute this program for 10 different sets of p, n and r then we can do F6 10 times which will waste time. Moreover we have to execute the set of instructions in a program 10 times. But the language provides a way to repeat these set of instructions using repetition control instruction.

Repetition…

Create a class Loop which contains main() and inside main() declare variables p and n as integer and r and si as floats. We want to repeat "Enter values of p, n r", receive the set of values of p, n and r, calculate si and print each value of si for 10 different times. To do so, we will keep a keyword while after declaration. while is always followed by a pair of parenthesis. Inside the while we can then display the message "Enter values of p, n r", receive those values, calculate si and print it out. while means so long as. So long as the condition which is written within the pair of parenthesis of while remains true, the set of statements would get executed. If we want to get executed these set of instructions 10 different times, then we say while (i <= 10). Declare i as an integer. The default initial value for i is 0. 0 <= 10 is true, hence the set of instructions inside the while would get executed and the control will automatically come back to while. In while, when the set of instructions would get executed, control will automatically come back to while. When it comes back to while, the value of i is = 0. 0 <= 10 and that condition is again true. So once again instructions within while would get executed. Again control will come back to while. Again i = 0 and condition get satisfied. So the set of instructions would get executed so long as i <= 10. Default value of i is 0 and 0 is always <= 10. If 0 <= 10 is remain forever, then we come to know that we are in infinite loop which never comes to an end. This is that we do not expect. To avoid this, we begin with i = 1 because if we want to count we usually begin with counting from 1 rather than 0. Then use while (i <= 10 ). Write the instruction i = i + 1 because if we do not do this, i would remain stationary at 1 and if i is always 1 then i is always <= 10. So once again we will fall in the infinite loop. Put all these instructions in a pair of braces. The moment we surround these instructions within a pair of braces, means all the instructions belong to the while loop. If we do not put a pair of braces then immediately one instruction after while belongs to the scope of while. Means in the program in absence of a pair of braces, System.out.println ("Enter values of p, n, r") would get executed infinite number of times. When we put instructions within pair of braces, we find that these set of instructions would get executed exactly 10 times. Initially i = 1, i <= 10 gets satisfied, so set of instructions get executed. Now i = 2, again i <= 10 get satisfied, so set of instructions get executed. This would go on happening. When i = 9, 9 <= 10 is true, set
of instructions get executed and i becomes 10. Now 10 <= 10 also get satisfied and set of instructions would get executed again i becomes 11. It checks 11 <= 10 and the condition fails. The moment condition fails control jumps outside the while loop and then execution comes to an end.

Logic Doesn’t Matter

Whatever we write in while loop, it does not matter. In the program, we had written simple interest logic, but we can write any logic like average, leap year within while. When we want to repeat any logic, we can use while loop. Any logic that we write within while loop, surround the logic with i = 1, while ( i <= 10 ) and i = i + 1. The moment when we do so, the instructions within while would get executed 10 different times. i used in the program is known as a loop counter or an index variable. i = 1 is the initialization of the loop counter. When we do while ( i <= 10 ) it is the testing the value of the loop counter against the final value 10 and when we do i = i + 1, we are incrementing the loop counter. So, initialization, testing and incrementation is the part and parcel of any while loop. The loop counter that we have used is int. The loop counter can be any int, long int, float or char. The loop counters can be incremented or they can be decremented. If we want the count should vary from 10 to 1 then we go for decrementation.

Another Program

Suppose we wish to print numbers from 1 to 10. To do so, we would begin with a class PrintNumbers inside which main( ) is present. Inside main( ), we can initialize i as int i = 1. Then we can test the condition using while ( i <= 10 ) and inside while loop we can print the value of i i.e. numbers. If we forget the incrementation part then the program goes into an infinite loop. To avoid this infinite loop, we can make sure that after printing the latest value of i, we had incremented the loop counter. So in this way, we can print value of i so long as value of i will remain <= 10. So we are able to print the numbers 1 to 10 using this program. In place of i = i + 1, we can also say i++ as well as ++i. When variable is followed by ++ then it is called as post-incrementation operation whereas when we do ++i means before checking i against the final value, we are incrementing the value of i, this is a pre-incrementation operation.

Incrementation / Decrementation Operators

++ always increments the value of a variable by 1 whereas -- always decrements the value of a variable by 1. ++ is used for incrementation and -- is used for decrementation. The way there is a i++ or ++i, we can have i-- and --i. **, //, %% these operators do not exist because their usage would not make any sense. i = i + 1 is same as i++. If we write i+++ in place of i = i + 2, then it is wrong. Hence, we cannot try i = i + 10. In this way if we cannot increment by 2 then we never increment by 10. This is the limitation of ++ operator. ++ operator is never be able to increment by any value other than 1 similarly -- operator is never be able to decrement by any value other than 1.

Would This Work

Suppose we say i = j----2. We saw that +++ does not exists. The statement says i = j---2 would correctly work. Because say i = j---2 is treated as i = j-- - 2. Means we are deduction 2 from j--. Now if we say int i, j = 3 and then we say i = --j. j which was 3 would now get reduced to 2 and 2 would the get assigned to i. When we do i = --3, it is wrong. --j is ok because ultimately --j is equal to j = j – 1. Whatever is the new value of j is then get assigned to i. However when we do i = --3, like j = j – 1, it will become 3 = 3 -1 and no compiler would ever going to parse this expression because on left hand side of an expression we have a constant rather than a variable. So i = --3 would not work. Now if we have int i, j = 2, k = 3 and when we execute i = ( j + k )++ we think that 2 + 3 = 5 and 5++ would be 6 and then 6 would get assigned to i. But it is wrong. This expression does not work. This expression becomes j + k = j + k + 1. No compiler would parse this expression. This is an illegal expression. This expression i = j----2 would not work. If we give a space between 3rd and 4th – then this expression would work. Right now it is not working. First two expressions given i = j----2 and i = j----2 both do not work. But if we give a space between 3nd and 4th – then the 1st expression would
work. Means if between 3rd and the 4th - we give a space and between 2nd and 3rd - we give a space then it works without any problem. It has to work because from j--, we are subtracting -2 i.e. i = j-- - -2. If we decide to drop the spaces then 1st space i.e. space between 2nd and 3rd – is optional whereas 2nd space i.e. space between 3rd and 4th – is compulsory. So ++ and -- can only be done on variables.
While Loop

In this lecture you will understand:

* Pre incrementation and Post incrementation
* Nuances of while loop
* Compound Assignment Operators
* Usage of break keyword
* Running sums and products
Is i++ Same As ++i

Suppose as shown in the slide we do int i = 1. Then we say while ( i <= 10 ). So long as this condition is true, we would try to print the value of i by using System.out.println ( i++ ). Before this, we had done printing and incrementation separately. Here we are doing both the operations together. Since we are doing both the operations together, both cannot be done simultaneously. In this case, printing will be the first operation to be carried out whereas incrementation will be done later on. Since ++ is occurring after the variable, so the incrementation is done later on. Since it occurs after the variable, incrementation is bound to be done afterwards. When we trace this while loop the values that are printed out are from 1 to 10. Now we do the same using ++i operation. So begin with i = 1. Keep while as it is and while printing say System.out.println ( ++i ). Here ++ occurs before the variable so it is called as pre-incrementation. So first operation is the incrementation and 2nd operation is printing. So when we trace this while loop and print the values of i, it will print from 2 to 11. But we do not want this output. We want output as 1 to 10. This clearly indicates that i++ and ++i are not same, they are different. They are same when i++ is being done in isolation. Means when we do i++ ; then it does not matter that whether it is i++ ; or ++i ;. In this case post and pre incrementation is same. The moment when we associate this incrementation operation with some other operation, then question comes which operation is done 1st and which is later. If incrementation is associated with something else then ++i and i++ are different. To rectify the 2nd program, so that it can print numbers from 1 to 10, initialize or begin with i = 0 and in place of i <= 10 we can write i < 10 or i <= 9. It means i++ and ++i are exactly same whereas if we say System.out.printf ( "%d", i++ ) and System.out.printf ( "%d", ++i ), they are different.

Two More Ways

In this example we are associating ++ with the condition. So we are saying while ( i++ < 10 ) and then printing value of i or we can also do while ( ++i < 10 ) and then printing value of i. In the first case, when we do while ( i++ < 10 ), first operation is testing and the second operation is incrementation whereas when we do ( ++i <= 10 ), the first operation is incrementation and second operation is testing. So in the first case begin with i = 0. When we reach the condition, firstly testing would happen. So 0 < 10 is true and then incrementation is happen so i becomes 1. When we reach println( ), i is already 1. So 1 gets printed. So this loop correctly begins from printing 1. So after 9 gets printed control reaches to while and test 9 < 10 and it is true. So after incrementation i becomes 10 and 10 gets printed. Then again control reaches to while loop and 10 < 10 fails and control comes out of while loop. So this loop correctly prints numbers from 1 to 10. In the second case also we begin with i = 0. Firstly incrementation is done so i becomes 1 and then testing is done i.e. 1 < 10 is true. So when control comes to println( ), 1 get printed out. When i becomes 9, 9 <= 10 is true and 9 is get printed out and the control again reaches to while loop. Now i becomes 10 and 10 <= 10 is true. So 10 get printed out and control again reaches to while loop and i becomes 11. Now 11 <= 10 fails and control comes out of the while loop. So loop also correctly prints numbers from 1 to 10.

Compare

In the 1st example, we use System.out.println ( i++ ). In the another example, we use while ( i++ < 10 ). In the left hand side, the two codes that we have we are using i+++. In the example which are at the right hand side, we are using ++i. All these code snippets print the numbers from 1 to 10.

Print Nos. From 1 to 10

Create a class PrintNum inside which there is a main( ). Begin with i = 1. Test the condition while ( i <= 10 ). Within while loop we print the value of i and after that in place of doing incrementation using ++ or using i = i + 1, we use a += operator like i += 1. i += 1 can also be written as i = i + 1 or i++ or ++i. All the four different expressions would increment the value of i by 1. += operator is known as a compound assignment operator. Like +=, there are -=, *=, /= and %= operators. a = a * 10 can be rewritten in a more
compact manner as a *= 10. Likewise b = b % 5 can be rewritten as b %= 5. *= allows to multiply by a
value other than 1 and %= do modular division by a value other than 1. ++ would always be able to
increment the value of a variable only by 1 whereas with += we can do incrementation with any value. In
this sense, += is more powerful than ++. -=, *=, /=, += and %= operators are known as compound
assignment operators.

One More Way

Create a class PrintNum which contains main( ). Inside main( ), start with i = 1 and then the loop
while ( true ). Means rather than checking the condition, we are saying this is the truth value of the
condition. We are writing true, indicating that the condition is always being treated as true. So the
while loop repeat infinite number of times. Whenever we have true inside the condition, the while
loop continues to get executed on and on, infinite number of times. Within the while loop, we would
first print the value of i. So first 1 get printed out and then increment i by saying i++. If after
incrementation we put a closing brace then such loop would not make any sense. Because we
do not want values to be printed infinitely. But we want to print values from 1 to 10. So if we want to
print numbers from 1 to 10 then there must be a provision to get outside the loop and we have to
make this provision inside the while loop. So after i++, we will try to check a condition if ( i > 10 ).
This is the situation when we want to get outside the loop. So long as i <= 10, it is perfect. We can
print the numbers. But when i crosses 10, we want to come outside the while loop. This is feasible
through a break keyword. When the control reaches if ( i > 10 ) break ;, the control reaches outside
the while loop. In place of break, we can also say System.exit ( 1 ). break will always terminate the
execution of the loop whereas exit( ) would terminate the execution of program itself. Suppose there
are some statements after while loop. When break goes to work control comes outside the while loop
and the statements after while loop get executed. Now suppose we write exit( ) in place of break and
there are some statements outside the while loop, then when exit( ) get executed these statements
would not get any chance to be executed. exit( ) makes the program stops dead in its tracks. It will
halt there, stops the execution and terminate the program. 1 used inside exit( ) is the reason for
termination of the program. In this program there is only one exit( ). But the situation might come
that there are three exit( ) points in a program. In that case when the program execution actually
comes to an end, we would have no way to find out from where the program did terminate. To help
that to find out, to exit( ) we pass a number. So at three different exit points when we make a call to
exit( ), we would say, System.exit ( 1 ), System.exit ( 2 ) and System.exit ( 3 ). Now from wherever
the execution of program actually gets terminated, we can find out from the value that is used with
exit( ). So if we want to find out the reason for termination of the program or the place from where
the execution of a program came to an end, 1, 2, 3 used with exit( ) become useful. So we need to
pass a number to exit( ).

Calculate

Suppose we want to calculate results of three expressions. The first expression is n!. n! = 1 * 2 * 3 *
… * n. The second expression is summation of all i where i changes from 1 to n. i.e. 1 + 2 + 3 + …. +
n. The third expression is X^n i.e. go on multiplying X, n times. If X has value 2 then it will become 2
* 2 * 2 * …. * n. In normal arithmetic we call such operations as running sums or running products.

Running Sum & Products

Create a class SumProds inside which there is a main( ) and then receive the value of n. Whatever is
the value of n, we will try to calculate n!, then summation till n and also 2 * 2 * 2 * …. * n times.
Suppose the value supplied to n is 5 then we have to calculate 5!, 1 + 2 + 3 + 4 + 5 and 2 * 2 * 2 * 2 *
2. To implement running sum or running product, we will implement a loop. If n is 5 then run a loop
5 times. Initialize value as i = 1 outside the loop and inside loop do i++. Then the loop will vary from
1 to n in steps of 1. Inside the while loop we will do s = s + i. i ranges from 1 to 5 and we also want to
do sum of 1 to 5. The intermediate result of sum would be stored in s. When we so s = s + i, we must
initialize $s = 0$. To calculate factorial, we say $p = p \times i$ and initialize $p = 1$. This is so because if we do not initialize $p$ to 1 then it will be 0 and 0 multiplied by anything will always be zero. So while calculating running product or factorial, initialize the variable which holds the intermediate result to 1. In case of sum, if we do not initialize $s = 0$ then also it will be ok. Because default value is 0 and while calculation sum it is allowed that initial value of the variable holding intermediate result is 0. So also initialize the variable $pr = 1$ which holds intermediate result of running product. So if we initialize $p$ and $pr$ to 0, every time we get the result as 0 which we do not want. So initialize them to 1. When control goes outside the while loop, whatever values of the $s$, $p$ and $pr$ are printed out. All the variables are declared as integers. We try the dry run of the while loop for calculating sum. Begin with $i = 1$ and $s = 0$, $i \leq 5$ is true and control enters the loop. $s = s + i$ will get executed and 1 gets stored in $s$. So now $s = 1$, then $i++$ makes $i = 2$ and the control goes to the while loop. Again $2 \leq 5$ is true and then $s = 1 + 2 = 3$ get executed and $i++$ makes $i$ to 3 and once again control foes to while loop. Now when $i = 4$, $s$ contain 6 so $4 \leq 5$ is true then $s = 6 + 4 = 10$ get executed. Again $i++$ makes $i = 5$ and control goes to the while loop. Condition $5 \leq 5$ is true, $s = 10 + 5 = 15$ gets stored in $s$ and $i$ becomes 6. Again control goes to while loop and $6 \leq 5$ fails. At that time control comes out of the while loop. Then the value of sum i.e. 15 gets printed.
In this lecture you will understand:

* Sum of Series
* Working of for loop
* Comparison of while and for loop
* Variations of for loop
Slide 3

The sine series is given in the slide. Suppose we want to add first 10 terms of the series. This is also called as Taylor’s series. This is nothing but a running sum. But in this series, alternate terms we want to add and alternate terms we want to subtract. Inside main( ), first receive the value of x. If value of x is 0.5 then result will give the sin ( 0.5 ). So after receiving the value of x, run a loop 10 times. Do i++ inside the while loop and initialize i = 1 and s = 0 outside the while loop. Once this is done, inside the while loop, we will try to calculate the value of each new term. Term contains numerator and denominator. So calculate numerator and denominator separately. Once this is done then calculate the term = numerator / denominator. Once the term is ready, we can either add the term to s or subtract the term from s. If the terms are 1st, 3rd, 5th, 7th and 9th then we will add the terms otherwise if it is 2nd, 4th, 6th, 8th and 10th term then we will subtract the terms. i.e. odd terms are added and even terms are subtracted. In this, the variable i will keep track of which term we are calculating. So if i is odd then we will do s = s + term whereas if i is even then we will do s = s – term. The determination of whether i is odd or even is done using % operator. Then when control comes outside the while loop, we will print the value of sum which is a sum of first 10 terms of the series for a given value of x. This slide gives the pseudo code for this program.

Slide 4

Create a class Series inside which main( ) is present and then inside main( ), receive value of x. Then we use a while loop which runs 10 times. Initialize i = 1 outside the loop, and increment i i.e. i++ inside the loop. In addition to i, we also make use of a variable j, initially which is also 1. But each time through a loop, we will increment i in step of 1 whereas j in steps of 2. Suppose we want to calculate the term $x^1 / 1!$. So we need j to represent 1 or 1 can be represented using a variable j. So $x^j / j!$ becomes any generic term in this series. For the 1st time $j = 1$ but 2nd time it should be 3, 5, and so on. So whatever we have in power, we have to increment it in steps of 2. So we are initializing j = 1 outside the while loop and inside loop we are doing $j = j + 2$. Now if we want to calculate $x^j$ then we have to do $x * x * x * ... * j$ times. The denominator contains $j!$, means it is $1 * 2 * 3 * ...$ upto j. To implement these running product we will using while ( k <= j ) loop. Inside this loop, we say $d = d * k$, $n = n * x$ and increment k each time in step of 1 using k++. If j = 5 then within the while loop we say $d = d * k$ and k will move from 1 to 5. Similarly we will try to get 5! in the denominator. Likewise if $j = 5$ and we do $n = n * x$, 5 times so we get $x^5$. Outside the loop we will initialize k = n = d = 1. Then inside the while loop, we do $t = n / d$. Once the term is ready then we have to either add it to sum or deduct it from sum and it depends on the value of i. if ( i % 2 == 0 ) then we should do s = s – t otherwise it should be s = s + t. Means for even term, we have to subtract term from s and for odd term we have to add term to s. Initialize s = 0 outside the outermost loop. Then outside the while loop print the sum of the series.

Prime No.

Inside the class PrimeNumber, main( ) is present. Inside main( ), receive the number which we want to determine whether it’s a prime number or not in n where n is an integer. Suppose n has value 13. To determine whether 13 is prime or not, we attempt to divide 13 with 2, 3, 4, ..., 12. If no number from 2 to 12 can divide 13 then we can say that 13 is a prime number. So these divisions are done in loop. Initialize $i = 2$. The loop will be so that $i <= n - 1$ i.e. if n = 13 then loop will run around from 2 to 12. Each time through the loop, we can check whether i can divide n exactly or not using if ( n % i == 0 ). If this condition is satisfied, if n gets divide by some value of i, then we will print the message that Not a prime number. Otherwise we should do i++. If the number gets divided with any value of i in the middle then it is meaningless to check whether the same number would get divided with further values of i. So we will break the loop using break keyword and come outside the loop. So outside the while loop, we can report that the number is a Prime number. 13 is a prime number so outside while we should report that 13 is a Prime number. Reporting of a prime number is done outside because unless and until checking with all the numbers is done, it is meaningless to say that the number is prime. For 13, unless we check all the divisions from 2 to 12, we cannot say that 13 is prime. When the control comes outside the while loop, we
had done all the divisions. Control comes outside the while loop in two situations. One when \( i \leq n - 1 \) fails then control comes outside the while normally. Second when break is executed then control comes outside the while loop abruptly. Outside the while loop, we check if \( i == n \) and if it is so, then only we say that the number is a prime number. So for 13 the number is prime get printed. For \( n = 10 \), \( n \% i \) would fail and control comes outside the while loop. Again \( 2 == 10 \) would fail. So 10 is not a prime number.

**Loop Control Instructions**

We saw that loop control instructions can be implemented using while. Loop control instructions can also be implemented using two more loops i.e. for loop and do-while loop. Now we will discuss the for loop control instruction.

**The while Loop**

This program finds the average of three numbers or a marks obtained by a student in three different subjects. Collect the marks in \( m_1, m_2 \) and \( m_3 \), calculate the average and print it. We want to repeat this for 10 different students. The steps that are carried out for this program are written in the box. Initialize \( i = 1 \). So we had done initialization. Then check \( 1 <= 10 \). So we had done testing. Since the condition is satisfied, get the values of \( m_1, m_2 \) and \( m_3 \), calculate the average and print it. Then do \( i++ \) which is incrementation. So now \( i = 2 \). This is getting repeated. So as shown in the box, we know that test and incrementation is getting repeated. Initialization is done only once and it is done outside the while loop also. So any way initialization is done only once.

**The for Loop**

We will take the same program using for loop. for is a keyword. Like while, for is also followed by a pair of parenthesis. Inside the parenthesis of for loop, we say \( (i = 1 ; i <= 10 ; i++) \). for allows to do initialization, testing and incrementation at the same place whereas in while it is done at different place. A pair of braces are also put around the for loop, otherwise only the receiving part belongs to the for loop. The default scope is only the next statement after the for loop. If we want that multiple statements should belong to the for loop then put them in a pair of braces. When execution begins \( i = 1 \) is done i.e. the initialization part. After that control goes to \( i <= 10 \) which is the testing part. Then as the test condition is true control comes inside the loop and receives value of \( m_1, m_2 \) and \( m_3 \), calculate average and print it and then control goes to \( i++ \) which is incrementation. It means even if we had written \( i++ \) after the test condition, but \( i++ \) will get executed only when body of the for loop get executed. After test condition body of for loop is get executed. So as shown in the box, we know that testing and incrementation is repeated so long as \( i <= 10 \). When \( i \) exceeds 10 then the for loop terminate. If after for loop within main( ), there are any statements then those statements would get executed. We can observe that the only difference between while and for loop is the place where we write the initialization, test condition and incrementation. Working wise both while and for loop seem to be same. In both the loops initialization is done only once.

**Comparison - I**

Suppose we write two code snippets using while loop and for loop as shown in the slide. If we see the code snippet then the code snippet which uses while loop runs in an infinite loop whereas the code snippet which uses for loop runs in a finite loop. We know that the default scope of while as well as for loop is the next statement following the loop. Means as shown in the slide in both the cases statement1 belongs to the while as well as for loop. In case of while, statement1 will keep on getting executed so long as \( i <= 10 \). But in while loop \( i++ \) never gets a chance to get work because scope of while is only statement1 that keeps on getting repeated. \( i \) never gets to exceed 10. Hence, we fall in an infinite loop. This would not happen in a for loop. Because after initialization and testing control goes to statement1 and after executing statement1 control goes to \( i++ \). So this will get repeated until \( i <= 10 \). When \( i \) becomes 11, condition fails and control goes to the statement2 and statement3 after
the loop. So the for loop is in finite loop. If we put a pair of braces around these loops then both the loops will work in exactly same way.

Comparison – II

Once again we write two code snippets using while and for loop. We also put a pair of braces around the while and the for loop. But the culprit here is that we put a ; at the end of both while and for loop. Once again the code snippet which uses while loop runs in an infinite loop and the code snippet which uses for loop run in a finite loop. This is because ; is a null statement. In case of while after initialization when i <= 10 is done ; would get executed and the control again goes to i <= 10 and this get repeated infinite number of times. This is so because i does not get a chance to get incremented. Infinitely ; is get executed and i never get incremented. So the statements in the while loop i.e. statement1, statement2, statement3 and i++ does not get executed. So while loop falls in an infinite loop. In case of for loop, after initialization, testing is done and after testing ; is get executed. After execution of ; control falls to i++ and i gets incremented. This will get repeated until i <= 10. When i becomes 11, 11 <= 10 condition fails and then statement1, statement2 and statement3 would get executed. This indicates that for loop falls in a finite loop. Giving ; at the end of the while or for loop is a fault but if we give then the behavior of two loops become different.

Comparison – III

Again we write two code snippets which uses while and for loop. In both while and for loop, the pair of parenthesis are empty. In for loop, in parenthesis we put ; which are necessary because of syntax. If we write loops in the manner given in the slide, then with while loop we get an error whereas in case of for loop, it is an infinite loop. The for loop which has two semicolons inside it is considered as an infinite loop whereas in case of while, it is not allowed to put a pair of parenthesis empty. It reports an error. So in case of while loop, we get an error and in case of a for loop, we get an infinite loop.

Print Nos. From 1 to 10

Suppose we want to print numbers from 1 to 10 then create a class PrintNumbers inside which there is a main( ). Then in for loop we say for ( i = 1 ; i <= 10 ; i++ ) and then print the value of i. This printing is get repeated till i <= 10. In place of i++ we can say i = i + 1 or i++ or ++i.

Dropping Init./Incr./Cond.

In for loop, we can drop the initialization part. Because while defining i, we can initialize it to 1 there itself. If we already initialize i = 1 then in for loop, initialization is not necessary but ; is necessary. Rest of the for loop is remain as it is, i.e. testing and incrementation and finally printing of values of i is done which prints the numbers 1 to 10. From this it is clear that, it is not mandatory to initialize the value of a variable in the for but ; at the beginning is necessary. We can also drop the incrementation part in the for loop. But still two ; are necessary. If we drop incrementation in the for loop, then we can do it in the body of the for loop. In the body of the for loop we can print the value of i and then do i++. So the numbers from 1 to 10 get printed. Finally we can even drop the condition in the for loop but two ; are necessary. We had drop the condition so we have to make a provision for that in the body of the for loop. This can be done using if ( i > 10 ) break ; i.e. get outside the for loop. break can also work with the for loop. Means in this form, initialization is done outside for, then testing and incrementation is done in the body of for loop. We can combine the printing and incrementation part into one using System.out.println ( i++ ). In place of combining i++ with println( ), we can combine it with if using the condition if ( ++i > 10 ). So we can either put i++ with println( ) or with the condition. Both are correct.
More For Loops

In this lecture you will understand:

* Nested for loops
* Usage of break and continue
* Multiple initializations in for loop
Combinations

We would try to write a program which will generate different combinations of 1, 2, and 3 for us. The different combinations that we want to generate are shown in the slide. It means with 1 in the 1st position we can generate 9 different combinations. Likewise when 2 is in the 1st position we can generate 9 different combinations and with 3 in the 1st position we can generate 9 different combinations. So we can generate 27 different combinations. We will try to write a program to generate all these combinations.

Starting Off…

Class Combinations contains main( ). It contains variables like i, j and k. For the 1st time i is 1. It will change to 2 for the next 9 combinations and 3 for the next 9 combinations. Also j will change to 2 and 3 for the different combinations. Everytime k ranges from 1 to 3. This is done using a for loop. Now begin with the k loop. Inside main( ), we say i = 1, j = 1 and k ranges from 1 to 3 and each time through this loop we will print the values of i, j and k. When we execute this, it will generate the first three combinations of the combinations shown in the slide. But we want 27 different combinations. So we had generated 1st three combinations. If we want to generate 1st 9 combinations then j must not be always 1. Because j is 1 for 1st 3 times, 2 for next 3 times and 3 for still next 3 times. To get is done, we will use the next slide.

Adding One More Loop

Keep rest of the program i.e. declaration of variables, initialization of i as it is. We do not keep the value of j as it is i.e.1 but we make it to vary from 1 to 2 and then from 2 to 3 using the for loop for j. Then put the statements within the j loop in the pair of braces. Add for loop for k within the for loop for j. After it is done, 9 different combinations would get generated. Because for each value of j, k changes 3 times. When j = 1, k takes value like 1 to 3, when j = 2, k takes value like 1 to 3 and again when j = 3, k takes value like 1 to 3. Means for each value of j, k takes 3 values. So finally the 9 different combinations as shown in the slide get printed out. The loop which changes earlier should always be implemented in the inner loop. Out of j and k, k changes much earlier than j. So k is kept in the inner loop and j as the outer loop. But we want 27 different combinations. To get is done, we will use the next slide.

Finishing Off…

Keep the program as done in the previous slide as it is. Now change i from 1 to 3. So when i changes from 1 to 3 and for each value of i, j changes from 1 to 3 and for each value of j, k changes from 1 to 3. In all the 27 combinations, i changes latest. So i is the outermost loop. Out of j and k, j changes later than k so j is the outer loop as compared to k and k changes fastest. Hence k is the innermost loop. So these 27 different combinations would now get generated. However we really do not want 27. We want to reject the 1st combination because in the 1st combination 1 is repeated 3 times. 1, 1, 1 we also want to eliminate because 1 is repeated here. 2, 1, 1 we also want to eliminate because 1 is repeated here. It means we do not want non-unique combinations. We want only unique combinations to get printed. For that we need to make some changes in the program as shown in the next slide.

Unique Combinations

If we want to print unique combinations, then there are 6 unique combinations as shown in the slide. So we will generate all 27 combinations but will print only the 6 unique combinations. So the rest of the program as shown in the previous slide will remain as it is. Inside the k loop before printing out i, j and k, we should figure out whether i is same as j and j is same as k. If we do so using if ( i != j != k ) then it will report an error because i != j will give the value false and that true / false cannot be compared with k since k is integer and the result of i != j is boolean. So this is not the correct way. So we will write multiple conditions and connect them using && operator like using a condition if ( i !=
j && j != k && k != i ) means within i, j and k there is no repetition. If there is no repetition means the condition gets satisfied then we will print the values of i, j and k. Only if the combinations are unique when we will print the combinations otherwise we will move for the next combination. So when we go through these loops, 27 different combinations get generated but we will print only 6 unique combinations.

**One More Way**

The generation of unique combinations is achieved using one more way. The for loops are as it is. Instead of checking if ( i != j && j != k && k != i ) we are trying to do the same thing using one more way using if ( i == j || j == k || k == i ). If this is the condition or if this is the situation then we do not want to print any combination. So we will do the printing in the else block. If all the three conditions fail then only the control lands in the else block. When all the conditions fail, it means that all the combinations are unique so the control goes in the else block and print the unique combinations. We are printing i, j and k in the else block. So if we found i == j || j == k || k == i then we will put break in the if block. break will take the control outside the k loop because break is right now working for the k loop. If we begin with the outermost for loop, i = 1 then it satisfies the condition 1 <= 3 and enters the inner loop. j takes the value 1, 1 <= 3 is true and then enters in the innermost loop. Now k = 1 and 1 <= 3 will satisfy the condition hence reach inside the k loop. Here i == j is true and since it is true, break will work and takes the control outside the k loop. Now j = 2 and 2 <= 3 is true and since it is true it would begin with k = 1 and k <= 3 i.e. 1 <= 3 would satisfy the condition. Now the control lands at if. Now i = 1, j = 2 and k = 1. As the condition gets satisfied break goes to work which would take the control outside the k loop. Now j becomes 3. The moment j becomes 3, we would realize that we had committed a mistake. Because the 1st unique combination that we expected to get printed out is 1 2 3. Now j has become 3 but 1 2 3 has not printed out. This indicates something is wrong. As j becomes 3, we had now lost the chance to print the 1st unique combination 1 2 3. If this is so then the program is wrong. When we execute this program, out of 6 unique combinations only 2 is get printed out. Hence we can conclude that break cannot be used here.

**The Correct Way**

The program is same as the previous slide. The condition is also same as the previous slide except that instead of using break keyword, we are using continue keyword. So now i, j and k each would begin with value 1. k <= 3 would satisfy the condition and the control reaches the if. The condition i == j || j == k || k == i is true so continue is get executed. break takes the control out of the k loop when we had break instead of continue. But we want to remain in the k loop and continue with the k loop by using next value of k. This is done by the continue keyword for us. When the control lands at continue, k++ would get executed. Now k becomes 2 and check 2 <= 3. The control lands at if. Once again the condition is true and the control goes to continue which brings the control to k++ and k becomes 3. Then 3 <= 3 is true and control falls in the if condition. Again condition gets satisfied and continue gets executed which brings control to k++ and k becomes 4 but the condition 4 <= 3 fails. The moment condition fails, the control goes outside the k loop. Now control goes to j++ and j becomes 2. The condition 2 <= 3 gets satisfied. So once again the control goes into the k loop and initialize k = 1. k <= 3 i.e. 1 <= 3 is true and the control reaches if. The condition becomes true and continue will work which brings the control to k++ and k becomes 2. 2 <= 3 is true and the control reaches to if. Once again the condition becomes true and continue will work which brings the control to k++ and k becomes 3. 3 <= 3 is true and the control reaches to if. At this time all the three conditions would fail and the control goes in the else block. In the else block the values of i, j and k would get printed. So the 1st unique combination i.e. 1 2 3 would get printed out. When we go through all the iterations then all the 6 unique combinations would get printed out. So break will always terminate the loop, it will always bring the control outside the loop. It will execute the next statement after the loop. Whereas continue will keep the control within the loop rather than taking the
control outside the loop. continue will cycle us to the next iteration of the loop whereas break will always terminate the execution of the loop.

Alternate Way

The same thing can be done in one more way. Program is same. The only difference is that instead of continue, we would write a ;. We get the same effect when we write ; instead of continue. If i == j || j == k || k == i, and if we put ; instead of continue, if any one of the condition is true and ; which is a null statement goes to work. Having executed the ;, else block will be skipped. So we would reach the end of k loop and would send the control to k++. Same thing is happen when we use continue in place of ;. Now in this case even if we replace continue with the ;, execution would not change. We would still reach at k++. But do not think that wherever there is continue, we can replace it with ;. continue abandons rest of the instructions in the loop and goes to the next iteration or next cycle through the loop. Suppose we write a for loop like for ( .. ; .. ; k++ ) inside this for loop we write if ( some condition ) and if the condition is true, we would put continue in the if block otherwise in the else block we would do a = 2. Then we have one more statement which prints Hello and a closing brace. Now imagine that while executing the for loop if the condition turns out to be true then the continue would go to work and will take the control to k++ without executing the rest of the statements. Means rest of the execution is abandoned and goes to the next cycle through the loop. Hence we come in the next iteration of the loop. If we put ; instead of continue and when the condition turns out to be true ; would get executed. When ; would get executed control will not come to k++ instead it skips else block and comes in System.out.println ( "Hello" ). ; does not abandon the execution of rest of the loop. So, continue can be replaced with ; provided beyond the else block, we have no other statements. In the unique combinations example, there is no such statement. So continue is replace with ;. In the example as shown in the box in a slide, there is a statement after else block. So in that case we cannot replace continue with ;.

Multiple Initializations

Suppose we have variables like i, j and k and then we initialize them as i = 1 and j = 3. Then run a k loop saying for ( k = 1 ; k <= 5 ; k++ ). Inside the for loop, we have some statements beyond which we increment i and also j. i is incrementing in steps of 2 whereas j is incrementing in multiples of 4. k = 1 would get executed only once whereas k <= 5 and k++ would continue to get executed so long as the test condition is true. Again i = 1 and j = 3 is done only once since they are outside the for loop. So i = 1, j = 3 and k = 1 i.e. all the initializations is done only once and if it so then it is not necessary to initialize them outside the for loop. We can initialize them where we are initializing k using for ( i = 1, j = 3, k = 1 ; ; ). Whatever is before the 1st ; in the for loop is get executed only once. We have done incrementation of i and j done together in the incrementation part of for loop. This concludes that multiple initializations in a for loop are ok and multiple incrementations are also ok provided multiple initializations are separated from one another using , and multiple incrementations are also separated using , as shown in the slide. Multiple conditions in the for loop are also ok provided they are connected using && and || operators. Do not separate multiple conditions using , likewise done in multiple initializations and multiple incrementations. In this example we had dropped k++ but that is perfectly ok. It is not necessary that since we are initializing 3 variables, we have to test all three and increment all three.
Do While Loop

In this lecture you will understand:

* Flexible declarations
* Sophisticated break and continue
* do-while Loop
* Effects of break / continue on different loops
* Repeating unknown number of times
Flexible Declarations

Take a class Myscope. Inside main( ), we run a loop for ( int  i = 0 ; i < 3 ; i++ ). Inside the for loop, print the value of i. Once outside the loop, we will print the latest value of i. Normally we define the int i outside the for loop and above it. However in this case we are defining i right inside the for loop where we are trying to write the initialization part of the for loop. In Java it is allowed to define a variable wherever we wish but certainly not beyond its point of usage. Means we cannot use i and then 10 statements down the lines try to define it which would be unacceptable. However we are not force to define the variable at the beginning of the function unlike many languages like a C programming. Java permits to define variables wherever we wish them but before the point of usage of that variable. Before we use the variable its type should be defined. Usually it’s done at the beginning of the function but that is not a rule. Once outside the for loop we are trying to print out the value of i which is impossible because the scope of that i is now limited only to the for loop. As soon as control goes outside that for loop, i is bound to die and hence i cannot any longer print the value of i.

Sophisticated break

Create a class Mybreak inside which we have a main( ) and then we are writing a for loop. Inside the outermost for loop, there is another for loop and inside that there is yet another for loop. We have deeply nested for loops. Then within the k loop, we write some instructions following which we check a condition if ( i + j + k > 21 % 4 ). If it is so then we want to execute the break. Now suppose during course of execution of these loops, the condition if ( i + j + k > 21 % 4 ) turns out to be true then break would go to work and when break goes to work it would take the control outside the k loop. Once we are outside the k loop, we would hit closing brace of j loop and that would send the control to j++. Means we would go to the next iteration of the j loop. For an incremented value of j, once again k would go from 0 to 5 in steps of 1. break is going to only terminate the k loop, control would remain within the j loop and the control would go to the next iteration of the j loop. But we may not want this. If this is precisely what we want then fine but if we do not want this, then the requirement might be such that if the condition in the innermost for loop turns out to be true then the control should go outside the outermost for loop. If that is the requirement then we can meet that requirement by saying a for loop like first: for ( int i = 0 ; i < 3 ; i ++ ). Means just before that for loop we write first and a :. So to say we are giving a name to that for loop. Once we are able to give a name to a for loop then if the condition ( i + j + k > 21 % 4 ) turns out to be true then instead of breaking outside the k loop, we may say break first and the moment we say break first control is going to outside the i loop. Means anywhere we want to get outside the outermost loop from deeply nested loops we have to give a name to a particular loop out of which we want to break if the condition in the innermost loop gets satisfied and then say break followed by name of the loop. So this is the sophisticated way of breaking outside the outermost loop.

Sophisticated continue

Like sophisticated break, there also exists a sophisticated continue. If in place of break, we use a continue, that continue will take the control to k++ means k will get incremented by 1. If we do not want that i.e. if we want that this continue will straightway take the control to j++ or i++, we need to give names to the i loop or the j loop as we did in the first case. When we had a sophisticated break that time also we had given the i loop a name called first. The grammar to do so is to say first : and then for loop. Then in place of continue we have to say continue first. So if the condition becomes true, continue first would take the control straightway to i++. i would get incremented and if ( i < 3 ) is true then once again j loop and k loop would be subsequently executed. This is another sophisticated continue available the there was a sophisticated break.

Type of Loops

Three different types of loop exist. We had seen a while loop, a for loop. We will not learn the third repetition control instruction i.e. do-while loop.
The do-while Loop

Create a class Loop inside which main( ) is present and using this program we intend to print numbers from 1 to 10. In main( ), we had written do-while loop. do is a keyword like while and for. do is not followed by any pair of parenthesis. We have to open a brace after do and inside this brace i which enjoys a value 1, we may try to print that out by saying System.out.println ( i ). After that we can increment i because we want after 1, 2, 3, 4 etc. get printed out and then write a closing brace and say while ( i <= 10 );. The pair of braces that we are using for a do-while loop are utmost necessary. Even if we combine the println( ) and i++ by saying System.out.println ( i++ ), still we cannot ignore pair of braces of do-while loop. Even if there is a single statement to get executed in a do-while loop, we cannot get rid of a pair of braces. The ; that we are writing after while is utmost necessary. It is necessary part of the grammar of the do-while loop. Execution begins with i = 1 and when it reaches to do, there is no condition that is being checked at the entry point of this loop. So control would straightway land at System.out.println( ) without checking any condition. At that time i = 1, so 1 get printed out. i++ would increment the value of i to 2 and then we reach the while loop. When we reach while, i is already 2 so 2 <= 10 is checked. Since the condition is true control will go to do. Again no condition there, so it would reach System.out.println( ). It will print out a value 2. Once again i++ will make value of i as 3. 3 <= 10 is true, yet again control goes to do. This will keep on happening till the time i <= 10. Last time around println( ) would print 10, i++ would make i as 11, 11 <= 10 would fails. The fact that it fails control would not any more go back to do. So this way it would print numbers from 1 to 10.

Compare

Suppose we write a same code snippet using while loop, for loop and a do-while loop as shown in the slide. If we try to execute the while loop, then it will not be able to produce any output at all because the condition that we had written 4 < 1 is an absurd condition. If the condition is going to fail first time itself, the body of the loop will not get executed even a single time. Same thing happen with a for loop. It reaches initialization part, initialization part is empty. So it reaches the condition. When the control reaches the condition 4 < 1, it is once again false. Since it is false, System.out.println( ) does not go to work and control jumps outside the for loop without producing any output at all. Unlike the while and the for, in case of a do-while loop, execution begins with do. There is no condition being checked at the entry point. So System.out.println ( "Hi" ) would print out Hi. Then it would reach the while. In the while the condition is false, 4 is never less than 1. Hence the control will not go back to do. But by that time Hi has been indeed printed once. With a for and a while, statements within the loop would be executed once or more number of times based on whether first time around the condition is true or false. If it is false, statements in the loop would not go to work even a single time. Unlike that in case of a do-while loop, no matter what absurd condition we write such as 4 < 1, even in such a case statement within a do-while loop, the body of the do-while loop would indeed get executed at least once. Statements in a do-while loop are going to get executed at least once.

Effects of break & continue

Suppose we write a while as shown in the slide. Inside which we have a if ( cond1 ) break and then if ( cond2 ) continue. If cond1 is true, break will take the control outside the while whereas if cond2 is true, in that case continue will take the control to the beginning of the loop. Same can be done using a for loop. If we have a for loop, within which we write if ( cond1 ) break and if ( cond2 ) continue. break is going to take the control to outside the for whereas continue is going to have a slightly different effect. continue will not take the control to the condition part as we did in case of a while loop, it would take the control to the incrementation part. After doing the incrementation then only the control would go to the condition part. In that sense working of continue is slightly different. In case of a while loop, continue cannot take the control to incrementation because there is no fixed place for a incrementation in a while. We can do it anywhere within the body of the loop. Hence, continue cannot take the control to the incrementation part in while. Hence, in case of a while loop whenever we use a continue, it will take the control to the condition part of the while loop and continue will take the control to the incrementation part in case of a for loop. Now we will see the effect of break and continue with the do-while loop. Once again same
conditions are used. if ( cond1 ) goes to work break would straight way take the control outside the do-while. continue will bring the control wherever we write the condition because again there is a fixed place where we write a condition in a do-while. break and continue would work with all three types of loops. Suppose we write if ( cond ) { .... break ; ... }. Similarly if ( cond ) { .... continue; ... }. We have to remember is, if in both cases there is not a loop at all. if and loop are different. So if we attempt to use break and continue in this manner, it is a major error that we are trying to use break and continue out of context. Context for a break and continue is a loop. Unless there is a loop, there is no question of break and continue. if is a decision control instruction and not a loop or a repetition control instruction at all. break is used to terminate the execution of a loop abruptly. Now we are writing a break and a continue with a proper context in a for loop. Within a for loop we can make use of break and continue. If we use break and continue in the manner as shown in the slide in the last two boxes at the bottom, even though grammatically it is correct, logically it is wrong. Because in the 1st for loop which uses break, if we run that for loop as for ( i = 1 ; i <= 10 ; i++ ) then for i = 1, statements within the for loop before break would get executed and when the control reaches to break, the control comes outside the for loop. So statements within for loop got executed only once. The statements which are written after break, those statements even would not get any chance to get executed. Same is true with continue. Everytime we run through this loop, continue will take the control to the incrementation part. So whatever we are going to write after the continue that stands no chance of ever getting executed. Because continue abundance the rest of the execution and goes to the next iteration of the loop. So statements beyond the continue have no chance of getting executed.

Unknown No. of Times

In this program we want to execute a set of instructions an unknown number of times i.e. an unpredictable number of times. If that is the requirement then when we write a loop we would say that the logic that is to be executed, put that within a while. Once that logic has been executed then ask the user himself /herself whether he / she wishes to continue or not and whatever he / she supplies receive that value in ch. Means 'y' or 'n' that is supplied try to read the value of 'y' or 'n' into ch. After that put the closing brace and then check while ( ch == 'y' ). If ch == 'y' then user has typed a 'y' for that ch. If 'y' has gone into ch, 'y' == 'y' would be true, the logic would be executed one more time. At the end of it, again ask the user does he / she want to continue or not. If he / she says yes go back to while, once again ch holds a 'y', 'y' == 'y' is true again statements within the loop would be executed next time. After having executed this 5 to 10 times, suppose now we want to stop that time when the message comes "Continue y / n", say we do not want to continue by typing 'n'. So 'n' would now gets stored in ch. When we go back to while ch holds 'n', 'n' is being compared with 'y'. 'n' == 'y' is false. Since it is false control will go outside the while. So long as user keeps typing 'y' for the message "Continue y / n", till that time the loop would run. The moment he / she types something other than 'y', it is considered to be no. In that case ch == 'y' would fail and then loop would come to an end. To ensure that the statements in the loop get executed at least first time around, we must make sure that ch == 'y' is evaluated to truth first time. For that we have to initialize ch to a initial value of 'y' such that first time around 'y' == 'y' would be true. Since it is true at least one execution is guaranteed. Whatever we have done using a while loop that is also achieved using a for loop and a do-while loop. In place of ch == 'y' if we want to make it a foolproof, we can say ch == 'y' || ch == 'Y' such that even if a capital 'Y' is supplied then we would still be able to consider that yes user wishes to continue the loop.
Case Control Instruction

In this lecture you will understand:

* Case Control Instruction
* Nuances of case Control Instruction
* Variations of case Control Instruction
Control Instructions

The control instruction in Java are Sequence control instruction. We saw Decision control instruction and Repetition control instruction. There are three types of repetition control instructions i.e. while loop, for loop and a do-while loop. While learning these repetition control instructions, we saw keywords like break and continue. We also learnt few operators like ++ and -- for incrementation and decrementation. We saw the difference between pre-incrementation and post-incrementation. We also learnt compound assignment operators. Now we will learn case control instruction.

Slide 4

We create a class Example which contains main( ). Inside main( ) we ask the user to Enter number between 1 and 3 and we receive the value of n. Then check if ( n == 1 ) and if it is true then display the message You entered 1. if ( n == 1 ) is false then the control go to the else block where we want to write several statements so open a brace. Inside this check if ( n == 2 ) and if it does then display message You entered 2. If this condition also fails then control falls to the else block within which we again proposed to write several statements. Once again we will open a brace and now check if ( n == 3 ) and if it does then display message You entered 3 otherwise control lands in the else block and display a message Wrong choice. Working wise this program is ok. But this is not the best way of writing the program. There are three problems associated with this program.

- First is there are too many ifs and elses to be matched.
- Second is in each if or the else, we have a single statement. If there are a multiple statements then it demands a pair of braces. So matching those pair of braces is going to be difficult.
- Thirdly, as the number of conditions goes on increasing, the indentation level also goes on increasing.

These three problems we had encountered earlier and we had solved those using Logical operators. Here to eliminate these three problems we cannot think of using logical operators like && and || because we suggested that && and || to be used whenever there are ranges to be checked and whenever there is a yes / no problem. This program is neither of the two. We are checking for specific values of n. So we will make use of case control instruction.

Slide 5

Upto receiving the value of variable n, everything is same as previous program. Then we would write a switch which is a keyword. Within the parenthesis associated with switch, we write the variable whose value we want to check. After switch open a brace. Irrespective of how many statements we write within switch, a pair of braces is necessary. Inside switch we write case 1: where case is a keyword. case 1 checks whether n == 1 or not. Every case is always followed by a :. If n == 1 i.e. case 1 is satisfied then we display a message You Entered 1. Then we would write case 2 means we are checking whether n == 2 or not. If it does then we would display a message You Entered 2. Then we you would write case 3, You Entered 3. Suppose if n contains a value 4 then case 1 would fail, control would jump to case 2; case 2 would also fail, control would jump to case 3 and case 3 would also fail. In such a case we would say else Wrong choice. This appears to be ok that if all cases to be failed then control would go to the else block and print out Wrong choice. But when we execute this we find that this reports an error because this has become a hanging else. Unless and until we have an if, there is no question of using an else block. So we need to rectify this situation.

Slide 6

The program will remain same as the previous program till three cases. After the three cases we should write default followed by colon where default is a keyword. Usually whenever we use a
switch then there would be a cases and there would be a default case. In the default case, we can say Wrong Choice. Space between case and 1 is necessary. So follow the rule for rest of cases. We cannot write case default, it is only default. For example, suppose to the readLine( ) function we supply value 2 i.e. n will hold 2. When we do switch ( n ), since n contains 2, it does not match with case 1 so case 1 fails. The moment case 1 fails, the control bounds to get to case 2. case 2 gets satisfied and You entered 2 will be printed out. Then case 3 would not be satisfied, it would jump to default. default will not also get executed because case 2 has been executed. Finally execution would come to an end. So any time a case fails control is bound to jump to the next case of the switch.

Slide 7

When we actually execute the program given in the slide, we find that the output for 2 as the input does not turned out to be You entered 2. It not only prints You entered 2, in addition to that it also prints You entered 3 and Wrong choice. Means the output that is shown in the previous slide is really wrong. When we actually execute this we get these statements to be printed out. We have to understand why do we get this output and then we subsequently see how to avoid this output. Again if we begin with value of n = 2 then case 1 when it is checked that case fails and control jumps to the case 2. n contains a value 2, so case 2 gets satisfied and You entered 2 get printed out. Following that control goes to case 3 and there onwards it will ignore case 3. It will print out You entered 3 that will go to default and it will ignore default and will print out Wrong choice. It is so because the way the switch has been implemented by the Java compiler. Anytime a case fails, control is made to jump to the next case which we had seen in the previous slides. But any time a case gets satisfied then every single statement below that case upto the closing brace of switch gets executed ignoring any more cases and default that may come across. Means case 2 gets satisfied and You entered 2 get printed out; case 3 is ignored, You entered 3 is printed out; default is ignored and Wrong choice is printed out.

The Solution

The solution to avoid the problems in the previous slides is as follows. We need to add a break statement at the end of case 2 i.e. at the end of System.out.println ("You entered 2"). break takes the control outside the loop. Likewise in this case, break terminates the switch and takes the control outside the switch. We also have to add the break in case 1. If case 1 is satisfied, You entered 1 should get printed and then switch should be immediately terminated; break will do it. Likewise we need to have a break in case 3 as well. Even though in case 1 we have System.out.println ("You entered 1") and a break as two different statements, we still do not need a pair of braces around them. Now this program works as per our expectation and we can see that this is a much better program than we had written earlier using several ifs and elses.

What If continue

Now we write a same program. Here we can observe that in case 2 after having printed out You entered 2, we are not writing a break. In place of that we are writing a continue. For example, suppose to readLine( ) we provide a number 2. n will hold a value 2. case 1 would fail control would jump to case 2. case 2 would gets satisfied, You entered 2 would get printed out. Then continue would go to work which will take the control to switch ( n ). If it reaches n, again case 1 is not satisfied. case 2 get satisfied You entered 2 gets printed and once again continue goes to work. Means we have stuck up in an infinite loop. Using a keyword continue inside switch as shown in the slide reports an error. The compiler reports that continue is being used out of context. Hence, continue should never been used inside a switch. So we can now conclude that break can be used within loops. break can also be used within a switch. However, continue can be used only with loops never with a switch. So in this program usage of continue is definitely illegal.

Slide 10
Within this program, we have written case 2 before case 1 and then we have followed case 1 with case 3. This is perfectly acceptable. We are free to write the cases in any order which we think is the best order. In short we can say that order in which we write the cases does not matter. We can write them in any order, we will find that there is no absolutely a change as far as the execution of the program is considered. Suppose to readLine( ), if we supply a number 2, case 2 will be satisfied, You entered 2 will be printed break would take the control outside the switch. If we had written case 2 after case 1 then if n contains 2, case 1 would have fail, control would have jump to case 2. Within case 2 once again You entered 2 would have been printed and break would have then taken the control outside switch. Means outputwise, functionalitywise there would have been absolutely no change at all. So we can conclude that order in which we write the cases does not matter.

Slide 11

Suppose we write a program and a switch as shown in the slide. We see that the default as the very first case. Take an example and figure it out. For example, suppose we supply a number 4 to readLine( ), so n contains 4. default case will not go to work immediately. It will first check out all the cases. If none of the cases are satisfied then only control will go to the default case. For n = 4, case 2 would failed, control would jump to case 1; case 1 would also failed, control would jump to case 3; case 3 would also failed. So control would go to the default case. In the default case Wrong choice would be printed out and once it has been printed out, there is going to be a problem. After having printed Wrong choice, case 2 will be ignored, You entered 2 would be printed. break would then terminate the switch taking the control out of it. If we do not want this then immediately after println ("Wrong choice"), we should say break. This break would take the control outside the switch terminating it. Means whenever a case gets satisfied, control is going to fall through till the time when the control either reaches end of switch or it encounters break. default case can have a break. So default can be the first case.

Slide 12

In this program, we have written all the cases. In the last case we have dropped the break. We know that is feasible. We have not written the default case because default case happens to be an optional case. Even if we do not write the default case, the compiler will in no way flag an error saying that we have not written the default case. In such a case, if value of n is 4 then none of the three cases i.e. case 1, case 2 and case 3 would not get satisfied and since we are not written the default case control would go out of the switch without printing any thing at all. If we do not want to do anything we are free to drop default case. If we supply valid case for n, then obviously one of the cases would get satisfied but if we do not i.e. if we provide a number like 4, 5 or 10 to n, in that case nothing would be printed out through this program because default case is not there. None of the cases get satisfied, control goes outside the switch. If n = 4, then we reach closing brace of main( ), execution comes to an end without printing out anything at all.
In this lecture you will understand:

* Nuances of case Control Instruction
* General form
* What data types can be checked
* Menu driven programs
Slide 3

Suppose create a class Example inside which we have main( ) and within main( ) we try to receive a character into the variable ch. We will first print a message Enter alphabet between A and C. Then receive the value of ch. There is no entity like Character.parseChar( ). Hence we have to use a function called charAt( ). When we do str.charAt ( 0 ), it means from the string that has been received in str using readLine( ) give the character that is present at index number or position number 0. So if we supply character B then B in str would be at character position 0. Hence charAt ( 0 ) will fetch that B which we can then store in ch. Once that has been done, now we can do a switch ( ch ) and as we do switch ( ch ), we would try to test whether ch contains a character A or not . We do so by using switch ( ch ) { case 'A': }. If ch contains A then we would print a message You entered A and get outside the switch using the break statement. On similar line we can have case 'B' where we say You entered B following with break. Likewise we can case 'C', You entered C once again following with break. We are proposing not to have any default case. Any alphabet between A to C that is supplied would be properly deal with using the suitable case from this switch. Problem is going to occur if somebody supplies a, b or c. Suppose 'a' is supplied then none of the three cases case A or case B or case C would be satisfied, control would go outside the switch and then execution of main( ) come to an end.

Slide 4

The program till receiving the value of a character remains same as the previous program. Now in switch we can say case 'A' || 'a'. If the value entered in ch is anyone of these two then we would say You entered A and then follow it with break. Same we will do for other cases, case 'B' || 'b'. Similarly case 'C' || 'c'. Now if A is supplied case 'A' would get satisfied, You entered A would get printed out and break would terminate the switch. If 'a' is supplied, the case is case 'A' || 'a' so we have 'a' in ch. So case 'a' would get satisfied, You entered A would get printed out and break would then terminate the switch. But it is what we think but the compiler is thinking a little differently. All these three cases have gone wrong. We can never say case 'A' || 'a' in this manner because || operator is a logical OR operator and the logical || operator can be used only to combine two boolean values. 'A' is not going to give a boolean value, 'A' is going to give a ASCII value of A which is 65 whereas 'a' ASCII value as is 97. 65 || 97 would certainly not work. 65 and 97 are not boolean values. 65 and 97 are integer values. So unless boolean values are combined using ||, || would refuse to work. So the compiler rejects this right at the compilation stage.

Slide 5

The program till receiving the value of a character remains same as the previous program. Now we write the switch as shown in the slide. So if A is supplied, the case 'A' would get satisfied, Entered A get printed out and break would then terminate the switch. Problem is not with 'A', problem is if we supply 'a'. For 'a', we want that the case 'A' would get satisfied, means all the statements that belong to case 'A' should get executed even if 'a' is supplied. One way to do so is write case 'A' and statements for that and then repeat the same set of statements for case 'a'. But this is not ok. Again before this comparison is done we may try to convert the 'a' into 'A' and then only enter the switch ( ch ). But this will not work if in place of 'A' or 'a', we have multiple cases where we check for numbers rather than alphabets. For example, if we want to check for case 5 or case 10 then we cannot convert these numbers into letters. So this can be done by using write case 'a' above case 'A'. When we do so as shown in the slide, if 'a' is supplied, case 'a' would get satisfied, case 'A' will be ignored; control will fall through and print Entered A and then break would terminate the switch. If 'A' is supplied then case 'a' would failed. As soon as this case gets failed, control jumps to the next case i.e. case 'A', this case get satisfied and Entered A is printed out and break will terminate the switch. So same can be done for case 'B' and case 'b' as well as for case 'C' and case 'c'. So if we want to check multiple numbers rather than alphabets then same can be done. For example, we can have case 23,
case 4 and case -55. For these cases if we want that a common set of statements to get executed so this is possible now.

Would This Work

In this example we write switch ( n ) and inside switch we write case a, case b and case 3 where n = 2. a has value 1 and b has value 2. Then we are writing case a, a is 1 so it becomes case 1. b is 2 so it is case 2 and then we have case 3. case 1, case 2, case 3 we would agree are unique cases. There is no conflict between these cases. So no matter which of these three, n takes which value, suitable case seems to be going to work. However, when we try to compile this, we find that it reports an error saying that case a or case b cannot be accepted because a and b are variables. Once they are variables we are agreeing that their values are likely to be changed during course of execution of the program. So to begin with cases might be unique but during course of execution who knows 'a' may get a value 3. It can take any value. So during course of execution 'a' which was originally 1 becomes 3 and then control lands into switch. Now we have case 1, case 3 and case 3. So there are two cases which are trying to check the value 3. Means these are non-unique cases. Non-unique cases are not acceptable because then the compiler will not be able to decide if n really enjoys a value 3. It would not know whether to take the control to first case 3 or to the second case 3.

General Form

Take a look at the grammar that is written for each case as shown in the slide. We are saying case constant expression whereas in switch we are saying expression. When we say expression, it might be a variable expression or it might be a constant expression. For example, n + 3 / a + 2 is a variable expression whereas 4 + 3 / 5 + 2 is a constant expression. Any one of these expressions can be written within a switch. However, when its time to write cases we cannot think of writing like a + b % c because it is a variable expression. However, if we say case 3 + 2 % 5 which is perfectly ok because it is a constant expression. Means at the compilation stage the compiler must be able to figure out what is the exact value of each case. Unless and until it knows that value, it will not be able to decide whether the cases are unique or cases are not unique and unless that gets decided it cannot really pass the switch.

Checking switch

So far we have checked two data types an int and a char using switch. So if int can be checked using a switch, long int can also be able to check using a switch and there is no reason why a float and double would not be checked using a switch. However, we can never check a float or a double using a switch. If we attempt to do so, it would immediately report an error. If we want to check a float, we have make use of if and else. So if we want to check if a float a enjoys a value 0.1, 0.2 or 0.3, do not try to make use of case 0.1, case 0.2 and case 0.3. Instead we have to say if ( a == 0.1f ) else if ( a == 0.2f ) else if ( a == 0.3f ) because we have reached the switch only after rejecting if and else. But if floats and doubles cannot be checked using a switch we have no other alternative but to go back to the if and else. if and else can check all these 5 data types without any problem. Suppose we have a question that is switch is a replacement for an if? On one count we can definitely say that it is not a replacement because now we know that floats and doubles cannot be checked using a switch. So if that is what we want to check, we cannot think of using a switch. However, in all cases other than that can we think of using a switch in place of if? We cannot because if at all switch is a replacement, it is a replacement in a very limited sense. In the sense that whenever we want to check the value of a variable on an expression to be equal to one value or another value or some other value or yet another value, if specific values are to be checked then nothing like a switch. But trying to use a switch, to find out whether a variable’s value or expression’s value lies between a range 1 to 5 or 6 to 10 or 11 to 15 for that switch cannot serve. Because switch was never invented for checking ranges. Likewise if we are required to solve the yes / no problem, again switch cannot be answer for that. switch can be a replacement for if and else only if we are going to check specific values rather than ranges rather
than solving a yes / no problem. Another question is that would if work faster or would switch work faster? Switch would work faster because whenever we use a switch internally a jump table is maintained and in the jump table it is mentioned that, if this is the value of expression jump the control to such and such instruction, if this is the value take the control to such and such instruction. Working with the jump table is faster rather than trying to evaluate the expressions at the execution time which would have to happen if we decide to make use of several ifs and elses. So in that sense if we can write a switch instead of nested if and elses, go for it because it’s going to make the program more efficient.

Menu Management

Within the class Menu, inside main( ) first we would try to display menu. For displaying menu put a println( ) to print 4 menu items. Exactly the same way we can ask for choice to be made and receive the choice using readLine( ) and then convert the string that is read using readLine( ) into a suitable integer by the method of Integer.parseInt( ).

Slide 10

Till receiving the choice, the program will remain same as the previous program. The menu gets displayed and choice is received. Once we have the relevant value 1, 2, 3 or 0 in choice. Then we are supposed to check whether choice contains 1 or 2 or 3 or 0 and then execute the appropriate logic. Then we would employ switch ( choice ) as shown in the slide. In case 1, we would write had put a comment saying that here we are going to add odd / even logic, following with break. Then in case 2, we will add leap year logic following with break, in case 3 prime number logic followed by break and in case 0 just a break. We can also say default where we say System.out.println ( "Wrong choice" ). Then close braces of switch, main( ) and the class Menu. Here we have to actually write the odd / even logic or leap year logic or the prime number logic in the three different cases. We should open the previous program and then Copy ( Ctrl + C ) and then Paste ( Ctrl + V ) only the logic or relevant logic here rather than writing. We have displayed menu, received the choice and then we have done switch ( choice ). Now suppose choice is received as 1 then when we do switch ( choice ), case 1 is bound to get satisfied. When case 1 gets satisfied, the logic would try to ask the user to Enter a number. Whatever number user enters then it would be reported as odd or even. Following that break would go to work and break would take the control outside the switch. Once the control goes outside the switch we are going to hit the closing brace of main( ) where upon the execution of the program is going to an end.

No while, No Menu

We expect that once odd / even program is executed and break terminates the switch, we should go back to displaying the menu because we want the menu to reappear. We want menu should get reappearing again and again and if that is so then we are essentially talking about repeating this logic. To repeat this logic, we have to write a infinite loop while ( true ). Once we say while ( true ) then this logic would continue to get executed again and again. But writing an infinite loop is not going to serve the purpose because if case 0 is selected then break would take the control outside the switch, we would reach closing brace of while and which would take us back to while ( true ). Means we will not be able to come out to the while loop at all. So we will fall into an infinite loop. But we do not want that to happen. Hence, in place of break we should say is System.Exit ( 1 ). Using the Exit( ) method we are trying to terminate the execution of the program itself. break terminates the loop or the switch whereas Exit( ) terminates the execution of the program itself. We are saying Exit ( 1 ) because there might be multiple exit points in a program from where the control actually exited that we wish to figure out then we may use Exit ( 1 ) at one exit point, Exit ( 2 ), Exit ( 3 ) etc. So we are passing a number to the Exit( ) function. In menu program, two things are important, one is an infinite loop and a switch. Because the menu has to appear again and again so we make use of infinite while loop whereas to made a choice switch is used.
Conclusion

No matter that how complicated logic that we have, that logic would always remain programmable using the four control instructions i.e. Sequence, Decision, Repetition and Case control instruction. These four instructions form the basic building block for creating any kind of logic for implementing any kind of functionality.
In this lecture you will understand:

* What are functions
* Program with multiple functions
* How to define functions
* How to call functions
* Different types of function calls
Functions

Suppose we write program whose class name is Functions. Inside class Functions, we have main( ) and inside main( ) we have a System.out.println( ) which prints the message saying I am in main. In addition to that we also have public static void bombay( ). The way main( ) was a function, we are now for the first time trying to define one more function within the same class. To give it some name we have called bombay( ). Inside bombay( ), we will print out some message I am in Bombay and then we have one more public static void kanpur( ) in which we decide to print out message I am in Kanpur. Lastly we put closing brace of Functions class. We call main( ), bombay( ) or kanpur( ) as functions because of the pair of parenthesis that are associated with these three words. main( ), bombay( ) and kanpur( ) all are followed by a pair of parenthesis so called as functions. Function is a word followed by a pair of parenthesis. println( ), print( ) and printf( ) are all functions because of the same reason. In addition to that we have used System.Exit( ). Exit( ) is also followed by a pair of parenthesis. So Exit( ) would also called as function. But for, while, if and switch are not functions even though they are followed by a pair of parenthesis because for, while, if and switch are keywords. So correct rule about the functions is, if a word is followed by pair of parenthesis that word is a function, provided it is not a keyword. We may think that in this program the first function that we have is main( ) and as usual main( ) would get executed, I am in main gets printed out. Subsequently to execution of main( ), bombay( ) would get executed; I am in Bombay would get printed out, following that there is a definition of kanpur( ) function. A kanpur( ) function would get executed and I am in Kanpur would get printed out. Means we expect that this program goes to work, 3 messages namely I am in main, I am in Bombay and I am in Kanpur should really be printed on the screen. But when we actually execute this, the output we obtained is only I am in main. This is so because execution of any program comes to an end after executing closing brace of main( ).

Calling Functions

We have the three functions main( ), bombay( ) and kanpur( ). Before we reach closing brace of main( ), we would say bombay( ) ;. Likewise kanpur( ) ;. Now after having printed out I am in main, we are not immediately hitting closing brace of main( ). Before we reach that closing brace, there are two words here bombay( ) and kanpur( ). We not only get I am in main, we also get I am in Bombay so also I am in Kanpur. It is so because after having printed I am in main, the moment we say bombay( ) ;, it is considered to be a function call. So bombay( ) is being called subsequent to that kanpur( ) is called. The moment we call bombay( ), control would go outside main( ) and jumps to the definition of bombay( ). When we said public static void bombay( ), we were indeed defining the function bombay( ). If the function is not followed by the ; then it is a definition whereas if the function is followed by ; then it is a function call. During the course of execution of the program whenever there is a function call, control jumps to wherever the function has been defined, executes all the instructions present within that definition then returns the control back to the place from where the call had originated. So in this program, execution begins with main( ), I am in main would get printed out. After that there is a call to bombay( ), control would now jump to bombay( ). Within bombay( ) we have printed I am in Bombay so that message will printed out. We reach closing brace of bombay( ) that returns the control back to main( ) to the immediately next statement after call to bombay( ). Immediately next statement after a call to a bombay( ) is a call to the kanpur( ) function. So control would now jump to the kanpur( ) function, executes the kanpur( ) function. During the course of which I am in Kanpur would get printed out, we reach closing brace of kanpur, control would go back in to main( ); subsequent of kanpur( ) we reach closing brace of main( ). As a result of which execution would come to an end. By that time all these three messages would get printed out.

Types of Functions

There are two types of functions that exits, one is standard library functions and second is user-defined functions. Library functions means functions which are available readymade. When we
install the Java compiler, these functions become available to us. On the other hand, we can also define the functions that we want. For example, bombay( ) and kanpur( ) we defined the way we wanted whereas println( ), printf( ), print( ) all of these are library functions. We are just using them, we are just calling them but we never defined them. So all these are standard library functions whereas kanpur( ), bombay( ), main( ), etc. are user defined functions.

Tips
- A Java program can contain one or more classes.
- Each class can contain multiple functions.
- Out of the several classes in a Java program, one of the classes has to be marked as Main class.
- Execution always begins from the main( ) function of the Main class.

Nobody is Nobody’s Boss
In the Functions class, we have main( ). Within main( ), we display a message and then call bombay( ) and then call kanpur( ). After that define bombay( ) and within bombay( ) we not only display a message but also make a call to kanpur( ). Then define kanpur( ) within which display a message and make a call to bombay( ). Execution would begin with main( ). Since there is only one class, this must be the Main class. Main class’s main( ) function is the first function to get called whenever a Java program executes. So execution would begin with main( ), I am in main would get printed out. Following that there is a call to bombay( ). So the control would land into bombay( ) function. When bombay( ) executes, I am in Bombay would get printed out. Then there is a call to kanpur( ). The moment there is a call to kanpur( ) control would leave bombay( ) and would jump to wherever kanpur( ) is defined. In the definition of kanpur( ), we have printed out I am in Kanpur. So that message would get printed on the screen, following which there is a call to bombay( ). Call to bombay( ) will bring the control to bombay( ), I am in Bombay would get printed out, now there is a call to kanpur( ). The control would jump to kanpur( ), I am in Kanpur would be printed following that there is a call to bombay( ). Means bombay( ) keeps calling kanpur( ), kanpur( ) keeps calling bombay( ). This will stuck in an infinite loop. There are no hierarchies about functions in a Java program. Any function is free to call any other function. In that sense nobody is nobody's boss. All functions are equal except for that execution begins with main( ). main( ) also does not enjoy any special privileges. We can call main( ) from bombay( ) and kanpur( ). Any function can call any other function.

Order, Order!
In the same class Functions, we have defined bombay( ) before main( ). It would be acceptable because we know that execution begins with main( ), no matter where is it defined. Whether it is a first function in the class or last function in the class, execution would still begin with main( ). Since execution always begins with main( ), the order in which we defined the function does not matter. The order in which we call the function that is indeed important. So order of definitions of functions is not important, order of calling the function from main( ) is indeed very important.

More Calls, More Bills
In the 1st program, we have a class Functions. Inside the class Functions, we have main( ) where we are running a for loop 20 times. Each time through the loop, we are trying to make a call to the bombay( ) function. We can make call to bombay( ) function 20 different times. So in this case bombay( ) is called 20 different times. The another program has the same class name Functions, inside which there is a main( ) and inside main( ), bombay( ) is called only once. Once we land into bombay( ), we are printing the message I am in Bombay 20 times. Output of both the programs
would be exactly same i.e. I am in Bombay would be printed out 20 different times. When we compare the performance of these two programs, we would realize that the second program is a better program because in the second program bombay() gets called only once whereas in the first case bombay() is being called 20 times. This difference in the performance of the program is because whenever we make a function call, some time is definitely spent in passing the control to the function and then returning the control back once the execution of that function is over. So passing of control and returning of control is indeed going to consume some amount of time may be only fraction of a millisecond but certainly some time would be devoted to transfer the control to the function and return the control back from the function. If we are required to do this 20 times, performance is going to suffer. As against this in the second program, we land into bombay() only ones. We do all the printing that we want and then go back once and forever. So we can say more the calls we make, slower becomes the speed of execution of the program. In short we can remember, more the calls we make, more would be the bill.

Types of Calls

There are varieties of calls that we can really make for different functions. For example, in program pr1.java which has main( ) and the function bombay( ). From main( ), we are calling bombay( ). From main( ), we are calling main( ). We have pr2.java, in which we had defined function fun1( ). The program pr3.java contains the function fun2( ) but pr3.java is not on the same machine on which pr1.java and pr2.java are present. Suppose pr3.java is present on completely different machine name Alpha which is connected to 1st machine in a network. From main( ), when we call bombay( ) that is known as Happy Hours calling. So within the file when we make the call that is a Happy Hours call i.e. calling the function in the same file in which main( ) is present. Another type of call is a Local Call. Means from within the body of main( ), we call main( ). Whenever we do so that function is known as a recursive function. So in this case main( ) becomes a recursive function because within body of main( ) there is a call to main( ). The process of function calling itself is known as a recursion. Thirdly, from pr1.java, if we try to call a function which is present in pr2.java we would call that as a STD call which costs more in terms of time. STD calls are more costly in the sense whenever we are going to cross the file boundary to make a call, it is going to be more time consuming and if we are required to cross network boundary then it is going to be a more costly call namely to be an ISD call. So whenever we make a call to a function which is residing physically on a different machine that is going to be certainly an ISD call.
Communication Using Functions

In this lecture you will understand:

* How to communicate between functions
* Passing values and control
* Returning value and control
* Returning multiple values from a function
Communication

Suppose we create a class Communication inside which we have a main( ). Within main( ) we define a, b and c as integer type three variables enjoying values like 10, 20 and 30. Then we called a function calsum( ). The calsum( ) function should be able to calculate the sum of the values which the variables a, b and c holds. Then we print the calculated sum. Inside the definition of calsum( ), we would try to add the values of a, b and c. We are storing the result of a + b + c in variable s which is a integer type variable defined in main( ) and then print s by System.out.println ( s ). The program seems to be ok but when we compile this program, we got some errors indicating that a, b, c and s are undefined variables in function calsum( ). This is so because a, b, c and s in main( ) are treated as different than a, b, c and s in calsum( ). So define a, b, c and s in calsum( ). When execution begins, from main( ) there is a call to calsum( ), control jumps to definition of calsum( ). Within calsum( ), we do s = a + b + c and print the value of s. When we print the value of s, we get the result as 0 in both the cases. This is so because when control lands to calsum( ), we have different set of a, b and c which are not initialized so hold the value 0 each. Thus the result is 0. When we go back to main( ) and try to print value of s which belongs to main( ), since s here is not initialized that will also print the value as 0. Means we are able to carry out a call to calsum( ) which transfers the control to calsum( ). However, communication between main( ) and calsum( ) is not happening.

Passing Values

Till the initialization of variables a, b and c in main( ), the program remains same as the previous program. Then when we made a call to calsum( ) from main( ), we should pass the values of a, b and c within the pair of parenthesis. Means during this call to calsum( ) from main( ), control is going to pass to calsum( ) and values of a, b and c would also been passed to calsum( ). When we pass values of a, b and c to calsum( ), the definition of calsum( ) must be able to collect these values. So in the definition of calsum( ), in the parenthesis, we say int x, int y and int z. Then we can add these x, y, z and store the value in s. Whatever values we pass to calsum( ), that value x, y, z will enjoy i.e. 10, 20 and 30. When we add s = x + y + z and print value of 6, it is turned out to be 60. So in calsum( ) function definition we get the value of s as 60. But when we go back from calsum( ) to main( ) and in main( ) when we try to print the value of s, we get the output as 0 because the value calculated as 60 and stored in s was stored in s which belongs to calsum( ). s that belongs to main( ) that continues to hold the value 0 since it has not been initialized. If we want to avoid this, then we must make sure that the way we pass values of a, b, c to calsum( ) because calsum( ) did not know about a, b, c likewise 60 that is calculated in calsum( ), main( ) does not know about it; we must return that back to main( ). a, b, c are known as actual arguments whereas x, y, z are known as formal arguments. Actual and formal arguments must match in number, order and type. Between actual and formal arguments, there must be a one to one correspondence.

Returning Values

Till the initialization of variables a, b and c in main( ), the program remains same as the previous program. When we make a call to calsum ( a, b, c ), control goes to the definition of calsum( ). We are collecting values of a, b, c in variables x, y, z. In calsum( ) we are defining another variable ss to hold the sum i.e. 60. This 60 is returned back to main( ) by using return ( ss ); where return is a keyword, its not a function. When we say return ( ss ), control goes back to main( ) along with that the value of ss is also going to back. When we return the value from the calsum( ), main( ) should be able to collect the returned value in some variable. For that we should say s = calsum ( a, b, c );. Means 60 that is returned from calsum( ) is now assigned to s which is an integer. Now when we print the value of s, it will be printed as 60. Instead of saying return ( ss );, return ( 60 ); is ok and return ( x + y + z ); is also perfect. With return, we can use either a variable like ss or a constant like 60 or an expression like x + y + z. If we want to return only the control and not the value to the calling function, in that case we say return ;. When we do return ;, only the control will go back but
the value would not go back with it. We would never be able to return only the value. Means any time we do return, control is always going to go back.

Are These Calls OK?

The same calsum( ) function is used here. Now we say \( x = \text{calsum} \left( a, 25, d \right) \). This function call is ok because values of \( a \) and \( d \) are collected in \( x \) and \( z \) and 25 is collected in \( y \). Now calsum \( \left( 10 + 2, 25 \% 3, d \right) \). Even this call is ok. \( 10 + 2 \) i.e. 12 is collected in \( x \), \( 25 \% 3 \) i.e. 1 is collected in \( y \) and value of \( d \) is collected in \( z \). So, actual arguments can be constants, variables or expressions. However, formal arguments like \( x, y, z \), they have to be necessarily variables because when we collect values, we have to collect them only in variables. We can never collect them in constants and expressions because values can never be assigned to constants and expressions. We cannot remove int \( x \), int \( y \), and int \( z \) and combine it with int \( ss \) because we know that actual and formal arguments have to match in numbers. Likewise we cannot add int \( ss \) to the formal argument list of calsum( ) function because of the same reason. Any local variable that we need for further calculations, we have to define those variables within the body of calsum( ). As shown in the slide, in the second function call, value returned by calsum( ) is not collected in any variable which is acceptable. When we pass the values, we have to collect them but when we return value from a function then we have a choice whether we want to collect it or not. The call calsum \( \left( a, \text{calsum} \left( 25, 10, 4 \right), d \right) \) is perfectly ok. In this call nested calls are used and nested calls are legal. In this, the inner call would be resolved first. Whatever value is returned, that would be collected in some temporary variable. That temporary variable is passed along with \( a \) and \( d \) for the outer call to calsum( ). Defining of one function within another is not possible. Function calls can be nested but function definitions can never be nested. The call \( d = \text{calsum} \left( a, 25, d \right) * \text{calsum} \left( a, 25, d \right) + 23 \) is perfectly ok. The result of calsum \( \left( a, 25, d \right) \) is collected in a temporary variable. Similarly when we make a 2\(^{nd}\) call to calsum \( \left( a, 25, d \right) \), whatever is result is collected in another temporary variable. Those two temporary variables would then multiplied with one another and then 23 is added to it and the result is assigned to \( d \). Calls within an expression are also legal.

Returning More Than 1 Value

Till now we had seen that the calsum( ) function returns only one value. But we also want that the calsum( ) should return not only the sum but also the product. So we write a function which returns sum as well as the product. We create a class ReturnTwo and inside main( ) we initialize \( a, b, c \) with values 10, 20 and 30. After that we make a call to function sumprod( ) which receives values of \( a, b \) and \( c \) and collect the values of \( a, b, c \) in \( x, y, z \) in sumprod( ). Then calculate sum using \( ss = x + y + z \) and product using \( pp = x * y * z \). Then we return sum as well as the product. The program shown in the slide seems to be ok but there is problem in main( ). Because the values of \( ss \) and \( pp \) that we are returning cannot collect in two variables by saying \( s, p \) since on the left hand side of =, only one variable has to exists. return \( \left( ss, pp \right) \) is itself is wrong because from a function we are allowed to return only one value at a time but here we are trying to return two values at a time sum and product.

One More Try

In main( ) we have \( a, b, c \). Then we will make two calls to sumprod( ), first time to obtain the sum and second time to obtain the product and then we would print \( s \) and \( p \). While defining sumprod( ), \( a, b, c \) that are passed are collected in \( x, y, z \). Then we calculate the sum and product using \( ss = x + y + z \) and \( pp = x * y * z \). Then we say return \( \left( ss \right) ; \) return \( \left( pp \right) \). This seems to be all right but there is a problem. When first time we make a call to sumprod( ), \( a, b, c \) that are 10, 20 and 30 would be passed, collected in \( x, y, z \), get the sum as \( ss = 60 \) and product as \( pp = 6000 \). Then return \( ss \) will go to work and it will return 60 back and will get assigned to \( s \). When we make a second call to sumprod( ), that time also return \( ss \) would go to work. So second time also \( ss \) which is 60 would get returned. Means when we print sum and product we get 60 and 60. So returning control like
return (ss) and return (pp) cannot work. return (pp) has become redundant because it does not get any chance to ever get executed.

**The Only Way Out**

Till the initialization of variables a, b and c the program remains same. The function sumprod() cannot determine whether its time to return sum or product. So when we make a first call to sumprod() in addition to a, b, c we should also pass 1 like sumprod (a, b, c, 1) and when we make the second call to sumprod(), we should say sumprod (a, b, c, 2). The 1 and 2 that are passed are collected in some variable code. Within sumprod(), we can check the value of code using if (code == 1). If it is 1 then we will return sum otherwise we will return product. So first time, we make a call, code is 1, so sum i.e. 60 would be returned which is assigned to s. Second time when we call sumprod(), code is 2, so product i.e. 6000 is returned and assigned to p. When we print them we get the output as 60 and 6000. Now we can overcome the limitation of a function that it can return only one value back. It would return indeed return only one value back but at different lines we can use it to return different values.

**A Better Way**

A more compact sumprod() function is as shown in the slide. We say code == 1 ? return (x + y + z) : return (x * y * z). But this line which makes the code compact does not work. If we want to make it work then we say return (code == 1 ? x + y + z : x * y * z).
More About Functions

In this lecture you will understand:

* Examples of functions
* Returning a non-integer value
* Different return types
* Beyond basic Arithmetic
Roman Equivalent

The program is used to figure out the roman equivalent of a given number. Begin with the class RomanEqu inside which we have a main( ). Inside main( ), we would ask the user to Enter year. Then we would receive the value of year in a variable y which is an integer. Decimal number 1 in Roman is written as i, 5 is written as v, 10 is written as x, 50 as l, 100 as c, 500 as d and 1000 is written as m. For example, if we take year i.e. y = 1998. To find out roman equivalent of 1998, there is one 1000 in 1998. So we first write m. Then 998 remains be to represent. Then there is one 500 in it. So we write d. Then 498 remains to represent. Similarly we can go through the number and the roman equivalent obtained for 1998 is mdcccclxxxxviii. To obtain the roman equivalent of a year, we would call a function romanize( ) in main( ) and pass the year to it like romanize ( y ). The year which is passed is received in variable yy. Inside this function, we do n = yy / 1000. n = 1 since yy is 1998. So we can print m. But if the year is 2998 then n = 2 be the result. In that case, we want to print two m. So to print that we will make use of a for loop. We run the loop for ( i = 1 ; i <= n ; i ++ ) and each time through the loop, print m. In the function romanize( ), define n and i as integers. But this function is inadequate because using this function we have represented only 1000 i.e. m but still 998 is remaining to be represented.

General Call

In the romanize( ) function, we are now saying n = yy / j instead of n = yy / 1000. Because using n = yy / 1000, we are making commitment to find out how many 1000s are present in yy. If we want to find out number of 500s or 100s in 998 then yy / 1000 would not work. For 1998, j should be 1000; for 998 j should be 500. Because of this reason, when we make a call to romanize( ), we should pass 1000 along with the y which can be collected in j. Also in the romanize( ) function, in the for loop which runs from 1 to n, we are not printing 'm' but we are printing ch. This is so because 'm' would be specific for 1000. To make romanize( ) more general, we should pass 'm' to romanize( ) along with y and 1000. So 'm' that we are passing to romanize( ) in main( ), is collected in variable char ch. So we are passing three arguments and collecting them in three formal arguments. Now whatever present in ch can be printed using System.out.println ( ch ). For example, if y = 1998 then the call to romanize( ) will be romanize ( 1998, 1000, 'm' ). So 1998 is collected in yy, 1000 is collected in j and 'm' is collected in ch. Then n = yy / j gives 1. If n = 1 then loop runs from 1 to 1 and only one 'm' gets printed out. Now we want to return from this function that 998 remains to be represent out of 1998 and we can achieve that by saying return ( yy % j ) i.e. 1998 % 1000 i.e. 998 will be returned. If we return this 998 and then want to make use of it for further calculation then in main( ) we should collect this 998. This can be done by y = romanize ( y, 1000, 'm' ). We are returning the remaining value from romanize( ). So in place of void romanize( ), we should say int romanize( ) indicating that it is going to return an integer.

Slide 5

In this we first make a call to romanize( ) using y = romanize ( y, 1000, 'm' ). In romanize( ), we calculate n and then print the number of m’s i.e. in this case is 1. When we do ( yy % j ), 998 would be returned in y. So y now contains 998. Now we will call romanize ( y, 500, 'd' ). Once again control goes to romanize( ) function and values are collected in yy, j and ch. So n = yy / j gives 1. So loop runs from 1 to 1 and one 'd' would get printed out. Again 998 % 500 i.e. 498 is returned and it get collected in y. Again romanize ( y, 100, 'c' ) is called. Again they are collected in yy, j and ch. Again n is calculated and this time value of n is 4. So loop runs 4 times and each time through loop value of ch i.e. 'c' is printed out. Again 498 % 100 i.e. 98 is returned in main( ) and collected in y. So in this way we can call romanize( ) to find out number of 50s, number of 10s, number of 5s and number of 1s. In the last call to romanize( ) the value that is returned from romanize( ) is not collected which is acceptable. Finally roman equivalent of 1998 is obtained as mdcccclxxxxviii. In this way we can find out roman equivalent of any given year.
Returning a Non-Int

Suppose we want to calculate the square of a number for that we will create a class SquareNum inside which main( ) is present. Within main( ), we would try to get square of 2.0f then 2.5f and lastly square of 1.5f. The square( ) function would return the square of a number and we would collect that in a, b, c and define a, b and c as float. Then we can print out a, b and c. In the square( ) function, we should do y = x * x and then return the value of y by using return ( y ) where y is a float. When we execute this, we get output as 4.0, 6.25 and 2.25. Here we are saying public static float square( ). Means whichever type we want to return, we have to precede the name of the function with the particular data type wherever we define the function within the class. So, return type of a function can be anything. It might be void, float, int, long int, double. Whatever is the return type, we have to clearly mention it while defining the function. In main( ) we are doing function call and outside main( ) there is a function definition. At times main( ) is also known as a calling function whereas square( ) is known as a called function.

Return Types

As shown in the slide, for example, suppose there are four functions. When we say void main( ) means this function is not going to return anything back. If we write int main( ) in place of void main( ) and then attempt to return 10 then it is not acceptable. main( ) has to be a void main( ). There is no choice for main( ). In some language, if we define the function and do not mention the return type, then the return type is assumed to be an integer. But Java does not make any such assumption. Unless and until we mention the return type, the compiler does not parse that function. In the fourth function, as shown in the slide, we are saying that this is a void fun( ) and within its body we are returning an integer value 20. The compiler would immediately report this as an error.

Output?

Suppose we have a class PrePost inside which we have a main( ). Inside main( ), we have a variable i of an integer type with value 3. Then we are printing the values of i by saying two printf( ) i.e. System.out.printf ( "%d %d", ++i, i ) and System.out.printf ( "%d %d", i, ++i ). When 1st printf( ) goes to work, i = 3, so ++i will make it 4. By the time we pass i to the 1st printf( ), i is already 4. So it will print the values 4 4. In the 2nd printf( ) when we attempt to pass i, that time i is 4 but when we attempt to pass ++i, i is already 5. So it attempts to print the output 4 5. Means arguments that are passed to function, are always passed left to right. So the output obtained is 4 4 and 4 5. In languages like C, arguments are passed from right to left. In that case the output would be different.

Beyond Basic Arithmetic

- java.lang package contains a Math class.
- Math class contains different functions and constants.
- The different types of methods that exist are
  - Basic methods which allows to get the round of a number or a floor of a number or ceiling of a number, etc.
  - Exponential and logarithmic functions
  - Trigonometric functions

Basic Arithmetic Functions

Create a class MathFunctions which contains main( ) and inside main( ) declare variables a and b as double with values a = -792.735 and b = 43.74. Then declare variables i and j as integers with initial values 15 and 45 respectively. Then we say System.out.println ( Math.abs ( a ) ). The abs( ) function
returns the absolute value of a number a. We want the ceiling value of b that can be obtained using System.out.println ( Math.ceil ( b ) ). The ceil( ) function returns the smallest integer greater than or equal to the value supplied i.e. b. For example, ceiling of 3.5 gives 4 but ceiling of -3.5 gives -3. In similar way we can obtain floor value of b using System.out.println ( Math.floor ( b ) ). The floor( ) function always give the largest integer which is less than or equal to the argument. We want to round of to the next integer value of b. So we use System.out.println ( Math.rint ( b ) ). All these functions return double. To find out maximum out of i and j, System.out.println ( Math.max ( i, j ) ) is used and to find out minimum out of i and j, use System.out.println ( Math.min ( i, j ) ). The output obtained for each is shown in the box of the slide.

Exp and Log Functions

Within main( ) of ExpLogFunctions class, we declare two doubles a = 3 and b = 2. Then we try to print out System.out.println ( Math.E ), System.out.println ( Math.exp ( a ) ), System.out.println ( Math.log ( a ) ), System.out.println ( Math.pow ( a, b ) ) and System.out.println ( Math.sqrt ( a ) ). When we do Math.E, we get the constant which is exponential with a value 2.718281828459045. There is another constant PI = 3.141592653589793 in Math class. When we do Math.exp( ) we get base of natural logarithm i.e. E raised to the argument i.e. a. In other words we are trying to find E

\[ E^a \]

which is 20.085536923187668. When we do Math.log( ), we will get the natural logarithm of the argument. a = 3, so natural log of 3 is 1.0986122886681098. Then pow( ) function gives 32 i.e. 9. Then sqrt( ) function gives square root of 3 i.e. 1.7320508075688772.

Trignometric Functions

Within main( ) we have a double d = 45.0. When we say Math.toRad ( d ), we are trying to convert the degrees to radians and whatever value is obtained is stored in r. Then we print Math.PI which gives 3.141592653589793. Then we obtained sin of r using Math.sin ( r ) which is 0.7071067811865475. Similarly we obtained Math.cos ( r ) which is 0.7071067811865476 and Math.tan ( r ) which is 1. We are trying to find out \( \sin^{-1} \) of Math.sin ( r ) and whatever result of that is converted into degrees using Math.toDegrees( ) which is 45.0. Similarly we obtain \( \cos^{-1} \) of Math.cos ( r ) and result is converted into degrees using Math.toDegrees( ) which is again 45.0. Likewise we obtain tan\(^{-1} \) of Math.tan ( r ) and result is converted into degrees using Math.toDegrees( ) which is 45.0. Math.toDegrees( ) converts radians to degrees.

Random Numbers

Random( ) function generates random numbers. We are saying ( int ) ( Math.Random( ) * 20 ) i.e. whatever is the result of Math.Random( ) * 20 is converted into an integer and result is stored in num. Then print the value of num. Similarly we can say 50 + ( int ) ( Math.Random( ) * 10 ) and store the result in num and print it out. Random( ) function will return the number between 0 and 1.0. So when we say Math.Random( ), it will generate some random number between 0 to 1.0 and when we multiply it by 20, we get some number between 1 to 20 whereas when we multiply the number generated by Math.Random( ) by 10 we get some number between 1 to 10 and then add 50 to it. Then we would be able to generate some random number between 50 to 60. So the output is as shown in the slide. You may get some different output.
In this lecture you will understand:

* What is Recursion
* Programs using Recursion
* Pros and Cons of using Recursion
* When to use it
Recursion – Different Forms

Suppose within class Recursion we have a main( ) function. Within main( ) we try to call a function fun( ). Inside fun( ) function, we first print the message Hi using System.out.printf( "Hi" ) and then we try to make a call to function fun( ). From within the body of fun( ), we are trying to make a call to fun( ) which makes it a recursive function because that leads to an infinite loop. So any function when it calls itself makes it a recursive function and the process of calling itself is known as recursion. We should put recursion in some better use rather than putting it in an infinite loop. When the function fun( ) keeps calling itself, some elements get created at a place in memory called Stack. So the recursion will go on till the time the stack does not become full. The moment stack becomes full, the program will stop.

More General

In the class SumOfDigits, we have a main( ). Within main( ) we should ask the user for the number and the number is received in num. After that we try to make a call to function sumdig( ). Pass num to the function sumdig( ). This function finds sum of digits of a number and return the result i.e. sum in the variable sum in main( ). After collecting the sum, we can print it. num and sum are defined as integers. Within the sumdig( ) function, we are receiving the number so we can collect it in int n. sumdig( ) function should be capable of finding sum of any digit number. So in sumdig( ), we say d = n % 10 i.e. extract the last digit in d and after doing that reduce the number by saying n = n / 10. These two instructions we want to repeat because we do not know what is the length of a number i.e. 3 digit, 4 digit or 5 digit. So we put these two instructions in a while loop using while ( n != 0 ). So after putting these two instructions into while loop, for the first time when we traverse the loop we obtain some value in d. When two again traverse through the loop then earlier value of d would be lost. For example, if the number is 485, then for 1st time d will contain 5 and when the 2nd time loop runs d will contain 8 but 5 gets lost. To avoid this loss, we can keep running sum of digits by saying s = s + d. Finally the latest value of sum is returned back to main( ). For example, if the number supplied is 327 then the sum obtained is 12.

Slide 5

The main( ) function of the program remains same as the previous program only the function name is changed i.e. rsum( ). In the box as shown in the slide, we are trying to calculate 4! i.e. 4 * 3 * 2 * 1. But this can also be expressed as 4 * 3!. Then 3! can be expressed as 3 * 2! then 2! as 2 * 1! and 1! as 1 * 0!. 0! is 1. Likewise sum of digits is a similar logic. Sum of digits of 327 we can say 7 + sum of digits of 32. Then sum of digits of 32 we can say 2 + sum of digits of 32. So the way factorial of a number can be expressed in the form of itself, sum of digits logic can also be expressed in the form of itself. So we can program it using recursion. Within rsum( ) function we should say d = n % 10 to extract the digit and n = n / 10 to reduce the number. After that we should say s = d + rsum ( n ). If n = 327 then d = 7 and n gets reduced to 32. Then we say s = 7 + rsum ( 32 ). When we say s = d + rsum ( n ) then we are in an infinite loop. We must make a provision to get outside this loop. We can do so by putting these three statements within if ( n != 0 ). If n = 0 then we can return 0. Finally we should do return ( s ). Here we are not initialized s to 0 because we are not calculation running sum here using s = s + d. If from main( ), we supply a number 0. So call is made using rsum ( 0 ) and n = 0. The condition in the function fails at the first time so the control falls in the else block and return 0. Whenever we make a call to function, a completely fresh set of local variables get created and whenever control returns from the function that time the variables would die.

Slide 6

The slide shows different copies of function rsum( ). From main( ) we had supplied 327 and we passed it to rsum( ). So first copy of rsum( ) will start with value of n = 327. So when we reach if
(327 ! = 0) control lands within if. Then d = 327 % 10 = 7 and n = 327 / 10 = 32 and s = 7 + rsum(32) is done. When we say rsum(32), then the control will go back to rsum(). So we say that control goes to another copy of rsum(). Now second time rsum() gets called with value 32. Again control reaches to if (32 != 0) is true. Then d = 32 % 10 = 2 and n = 32 / 10 = 3 and s = 2 + rsum(3) is done. We are again making a call to rsum() with value 3. So in the third copy of rsum(), rsum() is called with value 3. In rsum(3) when control goes to if (3 != 0), the condition is true. Then d = 3 % 10 = 3, n = 3 / 10 = 0 and s = 3 + rsum(0) is done. Now 0 is passed to n and when n becomes 0, if (0 !=) failed and control goes to else block. When it goes to else block, we do return 0 to the place from where the call had materialized. Call was materialized from the previous copy of rsum() function i.e. at s = 3 + rsum(0). So now s = 3 + 0 = 3 gets stored in s. Once s = 3 then if block is completely executed. Hence, else is skipped and we would reach return (s) i.e. we will return 3 to the place from where call was materialized i.e. s = 2 + rsum(3). So s = 2 + 3 = 5 is stored in s. Once again if block is completed. Hence, else is skipped and then we would reach return (s) i.e. we would return 5. Now s = 7 + 5 = 12 is done. So s will contain 12. Again if block is completed, else is skipped and return (s) i.e. 12 is returned to main(). We had created multiple copies of rsum() for the sake of understanding how the call gets materialized and how the control flows. Compiler creates only one copy of the function.

**Factorial Value**

In main() we ask the user to enter a number. The number is received in a variable num whose factorial we want to find. Then we made a call to a function factorial() and pass the number to it. This function returns the factorial of a number which we collect in a variable fact and then print the value. In the factorial() function we receive the number in n. Then inside the function we will a while loop by saying while (n != 0). Factorial is a running product so we say p = p * n. Each time through the loop, we keep on reducing n using n--. Then finally return the value of p. We must initialize p = 1 since we are trying to find out running product.

**Recursive Fact.**

The main() function of the program remains same as the previous program. Only the name of the function is changed to refact(). In the refact() function, we receive the number in n. So to calculate the factorial we say p = n * refact(n - 1). So if n = 4 then it becomes p = 4 * refact(3) i.e. 4 * 3!. But this is an infinite loop. So we will put this statement in an if block saying if (n != 0). We will calculate the factorial till the time n = 0. 0! is 1. Then lastly we will return the latest value of p. p is an integer type variable.

**Recursion Tips**

Following tips must be understood while dealing with recursion.

- Always make a recursive call within an if.
- else block forms a escape route.
- else would always have the end condition logic.
- return may not be present.
- Whenever there are too many calls, the Stack is likely to become full and this happens when there is something wrong in the logic.
- Recursion is not easy and not even fast because any time we make a call we have to pass value, pass control, return value and return control. More function calls slower becomes the execution. Recursion is difficult to understand, difficult to explain, difficult to write about and difficult to maintain. Maintenance of a program means the changes that are required over a period of time to making those changes in the existing program.
Slide 10

In the Towers of Hanoi example, 3 pegs A, B and C are given. On the first peg, 3 discs are mounted, each has different diameter. The objective is to move these discs to peg C. While doing so, we can use peg B as an intermediate peg. While shifting the discs from peg A to peg C, we have to follow two rules. First rule is, at a time we can move only 1 disc. Second rule is, on a smaller diameter disc we cannot put bigger diameter disc. So the first move is Shift disc from peg A to peg C which is shown in the 2nd picture. Next move is Shift disc from peg A to peg B which is shown in the 3rd picture. Now we cannot place disc from peg A on top of peg B or peg C. So we will move disc from peg C to peg B as shown in the 4th picture. Then in the next move we will shift disc from peg A to peg C as shown in the 5th picture. Then as shown in the slide, we have to shift disc from peg B to peg A. Then shift disc from peg B to peg C and finally shift disc from peg A to peg C. Finally all the discs from peg A to peg C in 3! + 1 moves i.e. 7 steps. While implementing this logic in program, it will need recursion.
Object Oriented Programming

In this lecture you will understand:

* What is object-oriented programming
* How OO model applies to programming
* Difference between class and object
* How to create classes and objects
* How to use objects
Real-World Model

There is a picture shown in the slide. Somebody says that it is a vehicle; somebody says that it’s my motorcycle, it’s my bike. When we say, it’s my bike then we are talking about which is very specific and whenever we are talking which is very specific then we are talking about an object whereas when we are talking about something generic like a vehicle then we are talking about a class because vehicle can be any vehicle. It might be a motorcycle, might be a truck, might be a truck, etc. When we talk about a specific vehicle, then we talk about an object. For example, if we say bird, then bird is a class. But if we talk about a sparrow then it’s an object. One object may contain several other small little objects. For example, this bike has an engine, gearbox, brake and lights. All these are objects. So these are objects within a bigger object called my motorcycle. Typically any object would have functions and data. For example, data for the Engine object is temperature of the oil, density of the oil, revolutions / minute of that engine, timing details of the crank shaft, type of fuel used in the engine. All this is the data about that engine. We also need functions which can work upon this data. For example, there might be some functionality which igniting the fuel of the engine, to move the piston, to open the valves, to eject the fuel that has already been burnt or to rotate the crank shaft. Talking only about functions is not enough because we need data on which the functions can work upon. Similarly just talking about data would not be enough because we need functions which can use this data. Functions and data co-exist. They go together. Whenever they work together means we are talking about an object. Functions in object oriented programming are also known as procedures whereas data is commonly known as properties.

Difference – Class & Object

Class realizes a blueprint or a template. Whenever we talk about something generic then it’s a class. When we talk about a specific person, then we have to talk about specific data and how this specific data can be utilized by the functions. In general, whatever is generic is a class and whatever is specific is an object. Suppose we have a class Employee, then an employee has properties like name, age, salary, etc. Then we have some functions which let us low to set the value of name, age, salary and there might be some other function which displays name, age and salary on the screen. Means we have data like name, age, salary and we have functions like set( ) and show( ). If we want to talk about an object then we can say that we have some object like e1 or e2 of the Employee class. Suppose name is Sameer, age is 30 of the object e2 and name is Sanjay and age is 30 of object e1. We have functions which let us allow to set this data or to display this data on the screen. Now we can clearly see that the data that is in object e1 is different than the data that is in object e2. Means two different objects can hold two different sets of data and then there can be functions associated with each of these objects which allows to either manipulate the data or access the data. Object e1 and e2 enjoys specific data. Hence, infact they are objects whereas when we just says name or age or salary, that is not specific data. Objects are unique.

More Classes and Objects

Suppose we have a DateTime class. The data that is maintained by the DateTime class might be day, month and year; likewise hour, minute and second. Then we have functions which can either access the data or manipulate the data. For example, getdate( ) and gettime( ) allows to have an access to the data whereas setdate( ) and settime( ) allows to manipulate the data. If we talk about object d1 then we can say 13/12/2007 is a specific date and 12:25:30 is a specific time and we have setdate( ) and getdate( ), similarly settime( ) and gettime( ) which can allow to access the date and time or manipulate the date and time. From the same class we can create multiple objects. We can also create object d2 enjoying some other value of day, month and year and some other value of hour, minute and second and a set of functions which allows to access or manipulate the data.

Working Of Stack
Stack is used in recursion. It can also be used in some other places like while evaluation some arithmetic expression that time internally stack is used. Data structures means way of arranging the data in memory. Stack is one of the data structures which is popularly used for doing variety of things in memory. Stack is a LIFO list means Last-In-First-Out list. Whatever items gets added to the list of items at the end, is the first one that goes out of the stack whenever its time for the element to go out. The last one that came in, is the first one that go out. We must keep track of where is the last item and to keep track of that a pointer known as Top is used. So Top will always keep information about the top of the stack i.e. where is the last item added. Because wherever it is added, that is where it is go out from. As shown in the slide, on the Stack, right now there are no elements. So we are saying that the Stack is empty. If we want, we can add an element 10 on the Stack. When we add element 10 to the Stack the Top will start pointing to the 10 because right now 10 is Top of the Stack. Adding a element on the Stack is known as a Push operation. So we can say Push 10 on the Stack. Then we push one more element 20 on the Stack and now Top has started to point to 20. Removing the element from the Stack is known as the Pop operation. If we do Pop, then the topmost element will be popped out of the stack. So 20 is popped out, only 10 will be left. If again we say Pop then 10 is popped and Stack becomes empty. Now Top will have some value indicating that there is no element in the Stack right now. If the Stack is implemented in the form of an Array then Top has given a value -1 because there is no element at position -1 within the array. Stack can also be implemented using a Linked List.

Stack Class & Stack Objects

If we implement the Stack as a class then the Stack class will hold data whichever we push on the stack as a collection of elements and this collection of elements in programming commonly known as an Array of elements and we also need top of the stack. To be able to work with the Array and the Top of the stack, we need operations like Push( ) and Pop( ) and one more method Printall( ) which will print all the elements on the stack. All these functions and data together form a Stack class. Then there might be object s1 of the Stack class which has 5 elements on the stack and Top is right now enjoying a value 4. There might be another object s2 which has 7 elements and Top has value 6. This is so because we have to start counting on the stack from 0. All these are specific things so are objects. To work with these things we need functions. So Push( ), Pop( ) and Printall( ) are part of object s1 and also be part of object s2.

Creation of Classes & Objects

We have a package classdemo. Within the package we can have multiple classes. One of the class is ClassDemo which has main( ). Within main( ), we say Sample a, b. Sample is a class which is user-defined. So we will first define the Sample class which contains int i and float j. This program has two classes, Sample and ClassDemo. a and b are references to objects of the type Sample. References hold addresses of the objects. Addresses of objects means the place where these objects are present in memory, at what location number. That location number would be the address of the object. To create an object, we use a keyword new. So we say a = new Sample( ). When we say new Sample( ), a new object would get created in memory which is a nameless object. The way to identify this object is using its address. When we say new Sample( ), we get the address of the object where the object got created in memory which is collected in a. So a becomes a reference to the object which does not have name. Even if it does not have a name, we can access the data present in it because we know the address of the object. In memory, a new object has created which contains an int i and a float j. a is pointing to that object which is indicated by an arrow as shown in the slide. Means a is holding the address of the nameless object that has been created. Then we can say a.i = 10 and a.j = 3.14f. When we do so, i and j belonging to the object to which a is pointing, 10 and 3.14 would be set up. So now we can do same for multiple objects. We can print values of a.i and a.j. Then we say b = new Sample( ). So another object would be created in memory which is also a nameless object but its address
wherever it is created in memory is collected in b. Then we say b.i = 20 and b.j = 6.28f and finally print those values.

**Better Way…**

In the same ClassDemo class, we have main( ) and inside main( ) we say Sample a, b i.e. declare the references. Then we say a = new Sample( ) and then call a function SetData( ) using a. Rather than directly approaching to i and j, we can approach i and j that belongs to the object to which a is pointing through a function SetData( ) using a.SetData( ) and pass 10 and 3.14f to it. So also to print the values rather that straightway accessing them, we can use function ShowData( ) using a.ShowData( ). Likewise we create b = new Sample( ), set the values using b.SetData ( 20, 6.28f ) and then print the data using b.ShowData( ). Now we will define the Sample class. It contains not only int i and float j but also we declare them as private variables which is a keyword. When we declare i and j as private means from outside the class Sample nobody can ever have an access to i and j. Whosoever wants to have an access to i and j, they will have an approach some public function inside the class Sample through which they can access i and j. i and j cannot be accessed directly. By doing so we prevent i and j from getting manipulated outside the class. So they become safe. We will write SetData( ) function which is public. Only public things can be accessed outside the class. If we do not declare i and j as private then they would be treated as public. The values that are passed to SetData( ) from main( ) are collected in ii and jj. Then within SetData( ), we can access i and j and set them up with ii and jj. Since SetData( ) is within class Sample, we can access i and j there. In the same way we write ShowData( ) function and print the values of i and j inside it. private and public are access specifiers. Default access specifier is public. This is a safer and better way of organizing the class.

**Which Is Better**

The slide contains two programs. In class Sample, we have public int age within it. From a Demo class we try to access this class. For that first we create some object s1 = new Sample( ). Then we will receive the value of x and then we say s1.age = x. Likewise we create one more object s2 = new Sample( ), receive x and then set that value to age by saying s2.age = x. Instead of doing it in this way, we can try it to do as shown in the second program. We declare age as private and write a public function setdata( ) to set the value of age. In main( ), we receive value of x and then we create same two objects s1 and s2 by saying s1 = new Sample( ) and s2 = new Sample( ). Then we say s1.setdata ( x ) and s2.setdat ( x ). If we want that age should never be less than 0 for that in 1st program we check if( x > 0 ) and if it is so then set up the value in age. In this program we are creating only two objects, but situation might come that we want to create multiple objects. Then everytime we have to check this condition. So instead of doing this checking in this way, if we do that the way which 2nd program uses then we can put the condition in the setdata( ) function. The way the 2nd program uses is a better way of doing it because we are keeping age safer from getting manipulated outside the class and also we are validated the data only once before setting it up.
Classes And Objects

In this lecture you will understand:

* Handling Complex Numbers
* Difference between primitive and object
* Where objects are created
* Stack versus Heap
* The this Reference
Handling Complex Numbers

Each complex number would have a real and imaginary part. For example, there is one complex number $2.1 + 2.3i$ in which $2.1$ is the real part and $2.3$ is the imaginary part. Another complex number is $1.8 + 1.1i$ where $1.8$ is the real and $1.1$ is the imaginary part. If we want to deal with the complex numbers, then we may want to add these complex numbers and after addition the resultant complex number would be $3.9 + 3.4i$. On subtraction we get the resultant complex number as $0.3 + 1.2i$. In the representation of the complex number, we may think of the real part and the imaginary part and functions like Add( ) and Sub( ) which can work on this real and imaginary parts. All these are put in a class Complex. Then we can create objects of this class which will actually holds specific values like $2.1$ and $2.3$. Likewise another object can hold $1.8$ and $1.1$. Both objects contain Add( ) and Sub( ) functions which can work upon values present in this object. Complex class is generic whereas objects that we had created are unique.

Implementation

In main( ) inside a class ClassDemo, we say Complex a, b. Then we say a = new Complex( ). a acts as a reference to the new complex object that has been created. Then in the complex object that has been referred by a, we store the values $2.1f$ and $2.3f$ using the function SetData( ) which are passed as arguments to the SetData( ) method. The values that are passed to the SetData( ) function are collected in some local variables like rr and ii in the function definition of SetData( ) in the class Complex. Then these values are assigned to the variables of the class i.e. $r$ and $i$. Then we call a function a.ShowData( ) to display the data. After that we create another Complex object by using b = new Complex( ), call b.SetData ( 1.8f, 1.1f ) which can set these values within the Complex object and again display the values using b.ShowData( ). In the class Complex, we declare private float r, i. then we declare SetData ( float rr, float ii ) as public. Whatever values we pass to SetData( ) from main( ) are collected in rr and ii and then these values are assigned to $r$ and $i$ using $r = rr$ and $i = ii$ in the SetData( ) function. Then we declare the ShowData( ) function which is also public and used to display the values of $r$ and $i$. private is inaccessible from outside the class. Anything which is public can be accessed from outside the class. Output is $2.1$ $2.3$ for the $1^{st}$ object and $1.8$ $1.1$ for the $2^{nd}$ object.

How Many Copies

Suppose we have two Complex objects having real and imaginary parts. We have two references which are pointing to them. We also have SetData( ) and ShowData( ) functions. Now a question is that how many copies of data and function would get created if multiple objects were created? It is obvious that each object would have its copy of data members since values of data members in each object are likely to be different i.e. data part of each object is different whereas functions are made common to both the objects. Conceptually there can be many copies of SetData( ) and ShowData( ), one associated with each object. But practically if there are many copies then it is wastage of memory. So in practice, only one common set of functions is maintained in memory and the same functions are used to manipulate and access data present inside different objects. So, only one copy of functions is maintained in memory for member functions of an object.

Difference – Object and BuiltIn

Suppose we say

```java
Complex a ;
a = new Complex( ) ;
```

On the Stack reference a would get created whereas the actual object, the nameless object, Complex object would get created another place in memory i.e. Heap. If we have int i ; float j ; then these i and j would always get created only on the Stack. Actual object always reside in the Heap whereas the
reference that refers to that object is always created in the Stack. Primitive variables like, i, j, k etc. would always reside only on the Stack.

Stack Vs Heap

Suppose we have some memory. Whenever we execute a program, some chunk of memory is reserved for that program’s stack. Into the Stack, different local variables are created. So if there are several programs running in memory then there are different Stacks for each of the program because each program is going to use different set of variables. We also have Heap in memory. In the Heap different objects are created. For example, there might be object1 of program1, object2 of program1, object3 of program1. If we have three primitive variables for program1 then they would always get created within the adjacent locations in the Stack. No such guarantee is given about the objects that are created in the Heap even though all the objects belong to one program. It is not mandatory that these objects would always been created in adjacent locations within the Heap. Unlike that, in case of a Stack, it is guaranteed that if a function uses five variables these are bound to be in adjacent locations within the Stack. Several other objects might also be there i.e. object2 for program2, object1 for program2 in non-adjacent locations. Similarly for program3 and program4 there are objects which are present in non-adjacent locations. Properties of Stack are as follows.

- Stack is used for storing not only local variables but also in Stack addresses of different functions goes.
- Whatever goes into Stack works very fast i.e. recycling of elements can happen very fast in Stack.
- Contents of the Stack can be kept in Cache memory which is a high speed memory.
- Caching adds to the efficiency of the Stack because we can access elements on the Stack faster.
- Whatever goes into Stack, actually occupies only as much memory as it deserves to occupy.
- Creation of elements in the Stack is very fast. There is special register called Extended Base Pointer (EBP) which always points to the base of the Stack. From there onwards calculate the offset in which a new variable would get stored.

Properties of Heap are as follows.

- Objects are always created in the Heap. The nameless object that we would create using new always goes into Heap
- There is no size barrier to how big an object we create. As much as Heap memory is available to us that big an object we can easily create.
- Heap is slow because recycling of elements in the Heap is take time. The objects that have gone to the Heap have to be physically eliminated from memory whenever we give up the object. Whenever we do not want to use the object, the object is removed from the Heap.
- Cashing is done in Heap but it was not as effective as it was in case of Stack because objects in a Heap are not guaranteed to be in adjacent locations.
- Whatever we create on Heap will always occupy more space than it deserves to occupy. Whenever we create an object on Heap, on the Stack we have to create the reference to that object i.e. if the object needs 50 bytes then the reference would need at least 4 bytes in the Stack. So an object occupies more space.
- Creation of object always happens at execution time or at runtime, it will take some amount of time. So Heap works slower.
If we observe the slide, then Heap does not seem to be a good idea. But we should not get rid of Heap because Stack is limited in size. So, big objects are never created on the Stack. So in Java, whatever is small i.e. primitive data types are created in Stack such that they can be accessed faster whereas objects that are big in size are always kept on the Heap.

The this Reference

Suppose we have an ex class which has a private int i and a private float a. Then we have a class Sample within which we have a main( ) inside which we create some objects of the type ex. By doing new ex( ), an object would be created and its reference would be available in e1. Likewise e2 would be pointing to another object that would be created using new ex( ). Then we say e1.SetData( 5, 5.5f ) and e2.SetData( 10, 10.5f ). We called SetData( ) twice and pass int and float values to this. Suppose e1 will occupy some space in the Stack which is a reference and object will occupy some space in the Heap. Likewise another object which is being referred to by e2 would also be created on the Heap whereas e2 reference would be created on the Stack. Whatever we pass to SetData( ) would be collected in some int and float i.e. ii and aa which can then be assigned to i and a. When we call SetData( ) using e1, it appears that we are passing two parameters to it. But in addition to 5 and 5.5f, e1 is also passed to SetData( ) and e1 contains reference. SetData( ) has to collect this reference. SetData( ) collects this in a reference whose name is this. Its type is ex and name is this. final indicates that whatever is present in this, that is something which can never change till the time we do not go back from SetData( ). When we go back from SetData( ), the ii and aa are going to get die. But during the execution of SetData( ), this is unchangeable. This can be achieved by declaring this as final means constant. final is a keyword. Internally SetData( ) would access i and a by saying this.i and this.a. First time when we make a call to SetData( ), at that time this would contain the address that is stored in e1 which is the address of the first object. Means e1 at that time would hold the address 400. This would be passed and this would hold 400. When we say, this.i and this.a in SetData( ), we are using 400 to access this i and a. Means when we say i = ii and a = aa, there is no confusion as to which i and a we are talking about. Even if we say i = ii and a = aa, externally the compiler would convert it to this.i = ii and this.a = aa. So this = 400 for the first time. So i and a for the first time will be setup with values 5 and 5.5. When we say e2.SetData( ), in addition to 10 and 10.5 this time e2 reference would be passed. So this would contain address of the 2nd object i.e. 500. If this contains 500, then when we say this.i and this.a then in SetData( ) we are referring to the object that begins with 500. So whatever reference we pass is present in this for execution of the SetData( ) function. We do not have to specifically say this.i and this.a. In that case it is optional. this pointer helps us to resolve, whether we are talking about i and a belonging to first object or i and a belonging to the second object. this can never be modified as it is final.

Is this Necessary

In a particular situation, there is no other chance other than to use this. For example, in the program shown in the slide, we are defined i and a as the private int and private float and in SetData( ) function we are using the same variables i and a. So if we just say, i = i and a = a, we are referring to the local variables inside SetData( ) function. We do not want that so we can resolve the conflict between two i’s by saying this.i = i and this.a = a. When we say this.i and this.a, we are referring to private i and a and when we say = i and = a means we are referring to the i and a which are arguments to the SetData( ) function i.e. local i and a. In this case alone, we would be required to make use of this reference specifically. Otherwise as shown in the previous slide, usage of this is optional.
Member Functions

In this lecture you will understand:

* What are constructors
* What are overloaded functions
* Types of calls – By value, By reference
* Garbage collection
Recap…

The quick recap of whatever we had done regarding classes and objects so far is as follows.

- Classes are user-defined data types.
- Objects are instances of a user-defined type i.e. a class.
- Multiple objects enjoy different data but always make use of a common set of functions between them.
- Data is usually private.
- Member functions are usually public.
- Objects are nameless.
- Objects are created using new.
- Objects are referred using references.
- References are created on stack.
- Objects are created on heap.
- this reference is passed to member functions.

Constructors

Suppose we have a class called Sample inside which there is main( ) and within main( ) we are creating new ex object. Now while creating new ex object we say new ex (10, 3.14f). So e1 would point to the object and inside the object we are storing 10 and 3.14. This can be achieved through a constructor function. In the ex class apart from private int i and private float j, we have a public function ex( ). This function would receive 10 and 3.14 into variables ii and jj. Then we can set up these ii and jj into the variables i and j. ex( ) is a public function so can be accessed from outside the class. This function would get called anytime when we create an object of the class. Hence this function which enjoys the same name as class is known as a constructor function. It is called as constructor because it let us allow to construct the object. Since it aids in construction of an object so we called it as a construction function. Now we are doing ex e2 = new ex( ) but this time we are not putting any value in it. Hence to construct this object, we are going to have one more constructor in which name is ex and within it we wish to set values of i and j to 0 and 0.0f. So we have one 0-argument constructor and one 2-argument constructor. This indicates that there can two functions in a class which can enjoy exactly the same name. In such a case those functions are known as overloaded functions. Overloaded means the same function has given the additional work. Any other member function within a class can be overloaded. Whenever we create an object, it’s a two step process. Firstly the space for an object is allocated in memory and in the space that has been allocated the constructor function should fill in some relevant values. Constructor name is slightly confusing because the word constructor gives the signal that this function allows to construct the object from scratch. But it is wrong. By the time we land in the constructor, object already stands created means memory already stands allocated for the object. We reach constructor function only to fill values inside the space that has already been allocated. Constructor will never allocate a space for an object. When we land in the constructor function, when we say i = ii and j = jj that really stands this.i = ii ad this.j = jj. Means this pointer already contains the address of the object i.e. object already stands created so contains address of the object. We land in the constructor function to assign the values in the allocated space.

Constructor Tips
• Name of constructor must be same as name of class.
• Constructor is invoked only using new.
• Constructor is a function.
• Constructor never returns any type.
• Constructor can be overloaded.
• If the class is empty, compiler inserts a Constructor.
• Either I do it or you do it policy. Means if we create a class ex and beside to provide a 2-argument constructor within the class ex, the Java compiler says that if we know how to put a 2-argument constructor then should also the responsibility of creating an 0-argument constructor within the class. We cannot say that we will provide a 2-argument constructor and will expect the Java compiler to provide the 0-argument constructor. This would never happen. Hence the compiler says that either you do it or let it do it.

Fun. Overloading

In a class Sample within main( ), we call a set( ) function and we are calling it 5 different times. Within the same Sample class, we provide all the versions of set( ) function. The last two set( ) functions are not ok. When we say set ( 10 ), the set( ) function which receives an integer get called. When we call set ( 10, 3 ), the set( ) function which receives 2 integers get called. When we call set ( 10, 3.14f ), the set( ) function version which receives integer and float get called. But when we call set ( 3.14f, 10 ), then it does not know the set( ) function which receives float and integer which is the fourth function or the set( ) function which receives float and integer which is the fifth function should be called. Means if we have two different functions bearing the same name they would be considered to be overloaded if we can actually distinguish between them. In the fourth and fifth version of set( ) function, we are unable to distinguish between them. Arguments of overloaded functions have to differ at least in number or in the order of argument or at least in the type of the argument i.e. arguments in the overloaded function must differ in number, order or type. There must be some way to distinguish the overloaded functions. Different return types are not sufficient or not enough for overloading. Because whenever a value is returned from a function, it is never binding upon a caller to collect the return value. When we make a call, we may collect the return value or may not. If we decide not to collect the value, then 4th as well as 5th version of set( ) serve the purpose.

Call By value

In the Sample class within main( ) suppose we have int i = 1 ; float f = 3.1f. Now we are calling function fun( ) to which we are passing value 1 and 3.1f i.e. fun ( i, f ). When we come back from function fun( ), we intend to print the values of i and f. Whatever is passed to fun( ) are collected in integer ii and float ff. Then we try to manipulate ii and ff by multiplying each with value 2. If we try to print the values of ii an d ff, then the manipulated values would get printed out. So the output would be 2 and 6.2. when we go back from function fun( ) and print out values of i and f in main( ), we do not get the manipulated values but instead we get the original values i.e. 1 and 3.1. Means the change that happen in fun( ) would not be reflected back in main( ). ii and ff are formal arguments whereas i and f are actual arguments. If we make changes in the formal arguments, it’s not going to ever change the values of the actual arguments because when we pass i and f, we merely pass the copy of whatever is present in i and f into ii and ff. ii and ff are the Xerox copies of i and f. We manipulated ii and ff means we manipulated only the Xerox copies and if we manipulated the Xerox copies then it is guaranteed that original copies remains same. So that result is obtained.

Call By Reference
We have a class Sample, inside it we have main( ) and within main( ) we create a new object by using
ex e = new ex( ). Then call a function Set( ) using reference e and pass arguments to it i.e. e.Set ( 1, 2.5f ). Then call the function Display( ) which is defined in the ex class. So a new object has been created and e is referring to it. Into the object, we have set up 1 and 2.5. Set( ) function is defined in the ex class. Within the ex class, we define two private variables private int i and private float f. Whatever is passed to Set( ) function is collected in x and y and set the values in i and f. In the Display( ) function, we would display the values of i and f. In the first call to Display( ), 1 and 2.5 would be printed out. Subsequent to this, we call a function fun( ) within the Sample class and reference e is passed to it i.e. fun ( e ). Following it we would again call the Display( ) function. In the function fun( ), the reference e that is passed is collected in p. e contains address of the nameless object. Same address is collected in p means p has started pointing to the same nameless object. Using p, we call the Set( ) function i.e. p.Set ( 3, 8.5f ). When we call p.Set( ), we pass 3 and 8.5 to the Set( ) function along with the reference p. When we say i = x and f = y in Set( ), actually it is this.i = x and this.f = y where this contains the reference value present in p. Since p is referring to the same object to which e is referring, the moment we say this.i = x and this.f = y in Set( ) function, that this pointer is going to overwrite the 1 and 2.5 with 3 and 8.5. So when we go back from fun( ) and call e.Display( ) from main( ), we get 3 and 8.5 as the output. Means whatever changes we made in the Set( ) function through the function fun( ) indeed become effective in the original object. So objects are being passed by reference and the reference got passed by value because references are created on the Stack. So when we pass e, whatever is present in e got copied in p. p also got created on the Stack. e and p both referring to the same object hence, when p tried to manipulate the object, the original object itself changed which did not happen when we try to make use of call by value.

References

If we say Complex c1, c2. Then c1 and c2 to begin with will hold a null value means these are null references i.e. references which are pointing to no object at all. When we do c1 = new Complex( ), c1 would start pointing to some Complex object having i and f. Then when we do c2 = c1, whatever is present in c1, same thing is stored in c2. c1 contains reference of the nameless object. So c2 starts containing the same reference. Hence c2 starts pointing to exactly the same object. This is known as shallow copy because the actual object did not get copied, its reference alone got copied. Then if we do c1 = null, c1 would stop pointing to the nameless object because the reference that is stored in c1 is now overwritten with null. We also do c2 = null. When we do so, c2 will also start containing null means c2 will also stop pointing to the nameless object. Now nobody is pointing to the nameless object. Then nobody ever be able to reach the nameless object because there is no variable in the program which contains address of the object. In that sense, it appears that this has become an inaccessible object. So whenever Java compiler finds that there is a nameless object in memory which is inaccessible to anybody because there is no reference that is pointing to it. In that case, the runtime invokes a program Garbage Collector. Garbage collector claims that the memory that is occupied by this object. Means it wipes this object from memory. Whatever space is occupied by this object is recycled for some other usage. This GC is invoked again periodically. Whenever it is invoked, it finds what all objects are present in memory which is not being pointed to by anyone. All such objects it would wipe and reclaim the memory that has been occupied by those objects. So in this way, objects are destroyed. We do not have to take any special effort to get rid of the object. When we want to kill the object, ensure that within the program, there is no reference which is pointing to that object. Then we can use the GC to eliminate the object from the memory. Means our job to eliminate the object from memory is restricted only to setting all the references that were pointing to that object to a value null. Then the GC take care of wiping out the object from memory, in this way objects are destroyed.
In this lecture you will understand:

* How to share data amongst objects
* Static data and static functions
* What is a Singleton class
* Need for a Singleton class
* finalize( ) method
Shared Data

Suppose we have a class Ex and inside class Ex we have private int i and then we have private static int count = 0. static is a keyword. Within the Sample class, we have main( ) and inside main( ) we would try to create objects of the Ex class using Ex e1 = new Ex( ) where e1 is the reference of the type Ex. When we do so, the 0-argument constructor of the class Ex would bind to get called. So we can either leave it to the compiler to create the constructor or we can create this constructor ourselves. In this case we are going to define the constructor ourselves and inside this constructor we are initializing i = 0 and we are also going to increment count by 1 using count += 1. Since I is a private variable, each object that is created from the Ex class is going to have its own i. Unlike that count is defined as static. The moment we define count as static, this is going to be shared amongst multiple objects of the Ex class. Static variables are defined right where we have defined them. Everytime a new object is constructed; control would land into the 0-argument constructor and as soon as control reaches there, increment the value of the count by 1. i is initialized to 0 each time a constructor would get called while creating a new object whereas count is common for all objects would be incremented by 1. After that we say Ex.ShowCount( ). Then the ShowCount( ) function would be called and within ShowCount( ) we can print count. The way we are calling function ShowCount( ) is used only for static functions. So we had declared ShowCount( ) as a static function. Means data as well as function can be static. The correct syntax for calling a static function is classname.function-name. When control lands into ShowCount( ), current value of count will be printed. Now count enjoys value 1 so 1 will get printed out. Then we do Ex e2 = new Ex( ), the control would once again land into the 0-argument constructor. Here count would be incremented to 2 whereas i for the new object is initialized to 0. If we again say Ex.ShowCount( ), the control lands into the ShowCount( ) function and the value of count is printed as 2. The two objects pointed to by e1 and e2 enjoy their own i, each one of which is initialized to 0. But count is not part of any of these objects. count is shared amongst multiple objects. By declaring count as static, we are able to keep track of how many objects have been created from this class. So far two objects have been created from this class Ex. Static functions can access only the static data, they can never access instance data. Static functions cannot call instance functions.

Problem

Now we wish to create a one single object from the particular class. For that we need to declare a constructor within the class as private and then we create a static member function to create an object of that class. This class is known as Singleton class. Singleton class is a class from which only one object can be created. This is a specific requirement and whenever there are such specific requirements, it’s known as a specific pattern known as a design pattern.

Singleton Class

Suppose we have a class Sample and from within main( ) of the Sample class we say Singleton s1 where Singleton is a class and s1 is a reference to that class. Then we say s1 = new Singleton( ). In the Singleton class, we declare private Singleton( ) { } then in main( ) s1 = new Singleton( ) cannot work because as per our purpose, we declare constructor of Singleton class as private. The moment we make the constructor as private, we would no longer be able to call that from outside the Singleton class. So s1 = new Singleton( ) cannot work any more. Then we may say that s1 = Singleton.Create( ). In this we are calling the Create( ) function using classname rather than the reference. Means Create( ) must be the static function inside the Singleton class. So we define the static function Create( ) in class Singleton as static Singleton Create( ) { }. Within the static function, we check if ( p == null ) and if it is null then we say p = new Singleton( ). At that time constructor get called because within the class any other function is accessible. The object would be created and its reference would be stored in p. Now p has started pointing to the Singleton object. Now we will define p as Singleton p. Then in Create( ) we say return p. Means from the Create( ) function, we are
returning a Singleton reference which is collected in s1 in main( ). So s1 would now start pointing to this object. But Singleton p ; is not ok because if we do p = new Singleton( ), p is going to be an instance data and instance data can never be accessible within the static function. So we declare p as static Singleton p; Now if we try to say Singleton s2 ; and then s2 = Singleton.Create( ). Then same Create( ) function would be called but now if ( p == null ) would fail because p is already pointing to the singleton object that we created first time around. Hence the condition fails and control reaches to return p. p contains address of the 1st Singleton object that we have created. So same address is returned and collected in s2. Means s2 would also start pointing to exactly the same object. Here onwards no matter how many times we call Singleton.Create( ), newer objects would never get created. Address of the 1st object would keep getting returned from the Create( ) function. So we achieved that any user can create only one single object.

Circle Class

Suppose we wish to create a Circle class and then the Circle object that we create from the Circle class should have radius of the circle and the color in which the circle is going to be drawn. This data we wish to maintain in each Circle object. Then we create a function Area( ) which helps in calculating area of the circle. We also want to count, how many objects of the Circle class have been created so far. So if we create multiple Circle objects, each one will have some color may be the 1st object has Red color, 2nd has Green and 3rd has Blue color and then each object has some radius 1.5, 1.1. and 1.4. We want to keep track of how many objects of the Circle class have been created means we will need a static count. There are three objects created. So count’s value is 3. While calculating the area, we are using the standard formula i.e. area = \( \pi R^2 \). The PI remains fixed and constant all time i.e. PI = 3.14. count and PI would be common for all the objects that we create from the Circle class.

Implementation

In the Circle class, we have int color, float rad and static int count. All are private variables. We say final float PI = 3.14f. static variables can be changed whereas final variables can never be changed. Even though both of them shared among multiple objects, final can never change whereas static can always change. Within the Sample class inside main( ), we create three references c1, c2, c3 each of the type Circle. Then we say c1 = new Circle ( 1.5, 'R' ). Similarly we say c2 = new Circle ( 1.1, 'G' ) and c3 = new Circle ( 1.4, 'B' ). Circle constructor we have to define. We want to calculate the area so by doing c1.Area( ) we can find it and finally we want to display the count of how many Circle objects have been created by saying Circle.ShowCount( ). In the constructor we are receiving the radius and the color in rr and cc and assign it to color and rad variable of the class and increment the value of count by 1 each time. count is a static data. Instance function like constructor can access static as well as non-static data. Then we write the function Area( ) and calculate area using PI * rad * rad. Finally we will define static void ShowCount( ) function and in that display the count.

Summary

Function can be either an instance function or a static function. An instance function can access private data as well as functions. An instance function is a function inside the class. Within the class anybody is accessible to anybody. Hence private data is also accessible to the instance function. Instance function is within the class scope and it is invoked using an object. Static functions are also present in the class. Static function can also access private data provided that data is static. Static function can never access instance data. Static functions are also within the class. Static functions are never invoked using object means a this reference is never passed to them. We always invoke the static function by classname.functionname. Static functions can be called even before creating an object.

Static Block
Suppose in a Sample class inside main( ) we say Ex.fun( ). Ex must be some class inside which we have static int x, y, z and then we have a static block having x = 10 ; y = 20. We also initialize z = x * 2 + y * 5. Then we say static void fun( ) inside which we print the values of x, y and z. The output obtained is 10 20 and 120. The statements within the static block are executed before fun( ) could be called. Means static block gets executed exactly once when the class is loaded. At that time x was initialized to 10, y was initialized to 20 and z was initialized to 120. So when we land in the static function, x, y and z are already set up. Means if we want that as soon as the class gets loaded some initialization is done at that stage, then we have to put that initialization in the static block. The syntax for static block is

```java
static
{
    /* some initialization */
}
```

**finalize( ) Method**

- finalize( ) method is called whenever an object is about to be destroyed.
- The actual destruction of the object is not done by the finalize( ) method, it will be done by the Garbage Collector.
- The finalize( ) method is used to free non-Java resources like file handle of a font.
- The finalize( ) method is called by Java-Runtime just before the garbage collection and it is defined as
  ```java
  protected void finalize()
  {
      //finalization code
  }
  ```
- The finalize( ) method will not be called just because when an object goes out of scope. It is called just before the Garbage Collector destroys that object.
Arrays - I

In this lecture you will understand:

* What are Arrays
* Need for Arrays
* Properties of Arrays
* Array Initialization
* Variable Sized Arrays
How Much Java

There are 48 keywords available in Java. We began with data types like int, char, float, double, long, short, byte, boolean. All of these are also keywords. We saw different control instructions like if, else, for, while, do. We also saw break, continue and then we saw switch, case and default. In addition to this, we are also aware of keyword like class, throws, private, public, final, import, package, new, static, this, void, return. All of these are keywords. All are almost 30 keywords. So we had done 30 keywords out of 48.

Storing Multiple Values

Suppose we wish to store multiple values of variables. For example, if we have a class Array inside which we have main( ) and inside that we ask the user to enter marks obtained by a student in three different subjects. We receive those values in m1, m2 and m3. Then we will get the percentage done and then we will try to print that out. If this is the requirement then it is perfectly served by this program. But if we are required to get percentage marks for 10 different students instead of only 1 student then we will have to repeat these set of instructions once for each student. This can be done by putting these set of instructions within the loop and define variable i as integer. But again besides going to calculate and print the percentage; before printing them we want to arrange them in ascending order and then we want to print them. This is not possible using this program because each time through the loop when we calculate a new percentage, the earlier percentage is going to be eliminated i.e. going to be overwritten with the newer percentage. Means if we go outside the for loop and print the value of per, we will get only the last student’s percentage. So we will not be able to arrange the percentage in ascending order since we do not have all the percentage. To overcome this problem, we have two possibilities. One is, create 10 different variables each holding one value. Another possibility is use only one variable and make it capable of holding all 10 values. But the 1st option is not best while 2nd is better. Whenever we create a variable which holds multiple values that variable is known as an Array variable. So array is a variable capable of holding several values simultaneously.

Arrays

Within the class Array inside main( ) we will define m1, m2, m3 and i as normal integer variables or ordinary variables. We are also keeping for loop same. Then we say int per[ ] = new int[ 10 ]. Means per is now going to be an array of 10 integers. When we say new int[ 10 ], the array of 10 integers would be created in the Heap, its reference or address will be returned which we will collect in per where per is a reference to the array which will be created in the Stack. Then we will print all the 10 percentages through another for loop. Once we go outside the for loop and try to print all percentages, all 10 percentages are available to us. Now we have the capability to make available all 10 values simultaneously. Now we can use this to arrange the numbers in ascending order and then print them. In the 1st for loop, we say per[ i ] = ( m1 + m2 + m3 ) / 3. When i = 0, then the 1st student’s percentage will get set up in per[ 0 ]. This will go on happening for rest of the students. Counting of array elements begins with 0 and stops at 9. So, 10 elements are numbered from 0 to 9. We have defined int per[ ] = new int[ 10 ]. We can define it also as

```
int per[ ];
per = new int[ 10 ];
```

We can also do it as

```
int[ ] per;
per = new int[ 10 ];
```

So there are three different ways in which we can define the array. per is a reference which is created on the Stack whereas the actual 10 element array is created on the Heap.
Array Initialization

Suppose we say int i and then i = 2 in two different lines instead of that we can combine them and say int i = 2. This can also be done for an array apart from normal variables. So we can initialize the array like int a[ ] = { 7, 6, 11, -2, 26 } ;. In this case we are not mentioning the size of the array. Because we can count that there are 5 elements in the array. In the same way compiler also can count that there are 5 elements in the array. So compiler would be able to allocate space for these 5 integers. Now we say int b[ ] = new int[ 10 ] ;, in this case we have to mention the size because if we do not mention the size, the compiler would not know how much space to allocate for this array. Suppose we decide to print a[ 0 ] and b[ 0 ] then the output obtained is 7 0. If we do not initialize an array, it will hold the value 0 like normal or ordinary variables. Now we are defining the array c using int c[ 10 ] = { 16, 13, -8, -7, 25 } ;. In this array c, we are defining the array as well as also mentioning the size of the array. Mentioning the size of the array is optional if we define the array at the same place. But in some cases even if we are defining the array at the same place, we may still want to mention the size of the array. Now we are receiving the elements c[ 7 ], c[ 8 ] and c[ 9 ] from the keyboard and also calculate c[ 5 ] = 3 + 7 % 2 ; and c[ 6 ] = c[ 1 ] + c[ 3 ] / 16 ;. In this case it was necessary that the size of the array c must be greater as we had done. 1st we had given size of array c as 10 and initialize only 5 elements in it. But later on we had fill rest of the elements. Arrays can be initialized like normal variables. The 5 elements initialized in array c are always necessarily being the leading elements of the array. Array elements can be scanned from the keyboard. Array elements can also be calculated and arithmetic on array elements is perfectly accessible. Array elements are different than normal elements in two ways. 1st is they can hold multiple values simultaneously which a normal element cannot hold and 2nd is that the normal elements are always created on the Stack whereas an array is always created on the Heap. The reference that points to the array is present on the Stack.

Tips About Arrays

• Array is always a collection of similar elements. For example, in the array int a[ ] = { 2, 1.4, 'A', 6 } ; elements are not similar. Means dissimilar elements we can never put within the array. In this array, 2 and 6 are stored and 1 is truncated to 1 and A is stored with value 65.

• Elements of an array are always stored in adjacent memory locations. As shown in the slide, we have an array a and it would be laid out in the memory as shown in the figure. Each integer occupies 4-bytes. So suppose 30 will begin with some address 402, 20 will begin at location 406.

• Different types of arrays are possible. So we can create an array of integers or we can even create an array of floats. Whatever way array of integers is created, exactly same way array of floats is created. All the arrays are created on the heap.

More Tips…

• In arrays, bound checking is never the programmer’s responsibility. For example, as shown in the slide we have an array a and then we are printing the elements of the array using for loop like for ( int i = 0 ; i <= 40 ; i++ ). If we do so, immediately an exception is thrown telling that we are out of the bounds of the array. Bounds of the array is from 0 to 4 but we are using 0 to 40.

• Instead of array of characters, we can also create a string. For example, as shown in the slide, if we create an array of characters where each element of the array is an individual character. We can access this by saying for ( int i = 0 ; i < str1.length ; i++ ) and then print the characters using System.out.println ( str1[ i ] ) ;. str1.length gives the length of the array means gives how many elements are present inside that array. Alternately we can say String str2 = "Hello" and then print out str2.
Variable Sized Arrays

Suppose we have `int n = 4` and then we say `int a[] = new int[n];`. This is perfectly accessible. Means the size of the array that we are mentioning need not be a constant. It is indeed possible to create variable sized arrays. The slide shows the way array is created in memory. On the Heap there would be 4 integers and on the Stack there would be a reference to this array. Dimension of the arrays can be a variable. We can also scan the value of `n` from the keyboard instead of initializing it.
Arrays - II

In this lecture you will understand:

* Selection Sort Algorithm
* Bubble Sort Algorithm
* Reallocation of Arrays
* Deallocation of Arrays
* Passing Arrays elements / entire Array
* Returning an Array
Selection Sort

Suppose we take a set of numbers 17, 6, 13, 12 and 2. To arrange them in ascending order, we will first compare 17 with 6 i.e. compare 0th number with 1st number. 17 > 6 so exchange 17 with 6. Then we will try to compare 0th number with 2nd. 6 < 13 so it will remain as it is. Then compare 0th with 3rd. 6 < 12 so will remain as it is. Then compare 0th with 4th. 6 > 2 so exchange them. After this comparison the smallest number i.e. 2 has gone to the beginning. Now we will keep 2 unchanged and then we will apply the same logic on the balance 4 numbers i.e. 17, 13, 12 and 6. After that comparison is done the smallest number out of 4 numbers i.e. 6 has gone to the beginning. Now we can keep 2 and 6 unchanged and apply the same logic again for 3 numbers i.e. 13, 12 and 17. After the comparison is done, the smallest number out of 3 numbers i.e. 12 has gone to the beginning. Now we keep 2, 6 and 12 unchanged and apply the same logic again for 2 numbers i.e. 17 and 13. When we compare 17 and 13, 17 > 13 so exchange them. At the end of this, we find that numbers have gone in ascending order i.e. 2, 6, 12, 13, 17. When we want to implement this programmatically then we have to generate the pairs of i and j as shown in the slide. For that we can implement a for loop which runs from 0 to 3 in steps of 1 for the value of i. Similarly for other values we can run a for loop using a variable j. So we have a class SelectionSort and inside main( ) we will define an array a as int a[ ] = { 17, 6, 13, 12, 2 }. Then to generate those pairs of numbers, we will run two loops. i runs from 0 to 3 and inside which j goes from i + 1 upto 4. Each value of j begins with 1 more than the value of i. After that we want to compare a[ i ] with a[ j ]. So we use if ( a[ i ] > a[ j ] ) and if the condition is true then we carry out the exchange of numbers using t = a[ i ] ; a[ i ] = a[ j ] ; a[ j ] = t. After that if we want to check that all the elements had gone to the ascending order or not, for that we can print all the elements of the array using the for loop. When we print the elements we get the output as 2 6 12 13 17. If we change the condition to if ( a[ i ] < a[ j ] ) then the number gets printed in descending order. So putting the number in ascending or descending order is known as sorting and the procedure implemented in this program is known as Selection Sorting Algorithm.

Bubble Sort

We begin with the set of numbers as 17, 6, 13, 12 and 2 and once again we compare 0th number with 1st number i.e. 17 with 6. 17 > 6 so we exchange them. Now we will compare 1st number with 2nd number i.e. 17 with 13. 17 > 13 so exchange them. Once again we compare 2nd with 3rd i.e. 17 with 12. 17 > 12 so exchange them and then we will compare 3rd with 4th i.e. 17 with 2. 17 > 2 so exchange them. After this comparison the biggest number out of all the numbers i.e. 17 has gone to the end. So we will keep 17 unchanged and apply the same logic on 1st 4 numbers i.e. 6, 13, 12 and 2. After the comparison is done, the biggest number i.e. 13 has gone to the end. Now we will keep 13 and 17 unchanged and apply the same logic again on the remaining 3 numbers i.e. 6, 12 and 2. After this comparison is done the biggest number i.e. 12 has gone to the end. Now we will keep 12, 13 and 17 unchanged and apply the same logic for remaining two numbers i.e. 6 and 2. After this comparison i.e. 6 > 2 we will exchange them. So at the end of this we will find that all the numbers have gone in the ascending order i.e. 2 6 12 13 17. Once again to implement this logic we have to generate the pairs of i and j as shown in the slide. In the BubbleSort class inside main( ), we define an array as int a[ ] = { 17, 6, 13, 12, 2 }. Then we will run an outer for loop from for ( j = 0 ; j <= 3 ; j++ ) and the inner for loop from for ( i = 0 ; i <= 3 - j ; i++ ). Inside the inner for loop, we check the condition that if ( a[ i ] > a[ i + 1 ] ) and if it is true then we can exchange the numbers using t = a[ i ] ; a[ i ] = a[ i + 1 ] ; a[ i + 1 ] = t. We are doing i <= 3 – j because in the 1st set i is beginning from 0 and ending at 3. But for the 2nd set we are going from 0 to 2. For the 3rd set we are going from 0 to 1 and for the 4th set we are going from 0 to 0. So we get the requisite number of comparisons get done. Finally to check whether the numbers are correctly sorted in ascending order or not, we will print all the numbers using for loop. We get the output as 2 6 12 13 17. In this way we can implement the Bubble Sort logic.
Sorting Procedures

There are many other sorting procedures as well. There also exists a sorting procedure known as Shell sort, Shuttle sort, Heap sort, Merge sort, Radix sort and Quick sort. But question arises that why do we have so many sorting procedures? This is because sorting is one of the primary activities for which we use the Computer. Hence, sorting has to be done as efficiently as it can get. Sorting can be used in so many places. For example, if a bank may want to sort the transaction that it does in chronological order by date. A bookshop may want to sort the names of the books in alphabetical order. Selection sort and Bubble sort are not efficient as Quick sort. Infact, Quick sort is one of the fastest sorting algorithms as compared to other sorting procedures available to us. Ultimately in any sorting procedures, we carry out comparisons between pairs of numbers. In Selection sort and Bubble sort, they make use of comparisons to the tune of \(n^2\) comparisons. Compare to that Quick sort make use of only \(\log_2 n\) number of comparisons to sort the numbers out. Quick sort incidentally makes use of recursion. As Quick sort uses recursion but we even say that it is fastest because whatever time Quick sort looses by being required to use recursion, it more than makes up for that lost time by being required to do less number of comparisons. If \(n = 10\) where \(n\) is the number of elements that we wish to sort then with the help of Bubble sort or Selection sort we are required out to carry out comparisons to the order of 100 comparisons. As against that when we use Quick sort, we can get the same thing done by much lesser number of comparisons i.e. comparisons as lesser as to the order of 3. Since Quick sort requires less number of comparisons, it is a fastest comparison algorithm.

Reallocation & Deallocation

Suppose we have two arrays \(a\) and \(b\) out of which we have initialized \(a\) to \(\{1, 2, 3, 4, 5\}\) whereas \(b\) is not initialized; we have only declared it as \(\text{int} \ b[\ ]\). If we try to print out \(a\) then we will observer that it will print some odd number like 548743452. This is the contents of the variable \(a\). \(a\) is a reference which holds address of the array wherever the array has been created in the Heap. \(a\) indeed contains the address, it does not contain the array itself. Hence, when we do \(b = a\), we are not assigning an array itself to \(b\); we are assigning the reference that is stored in \(a\) to the reference \(b\). All arrays are always dynamically allocated i.e. always created on the Heap. Then if we say \(a[\ ] = \text{new int}[10]\), now \(a\) will no longer will point to the original array; it will start pointing to the completely new array of 10 integers. In this way we are resizing the array. To resize the array we are allocating another array of 10 elements and into this array now we can copy the 5 elements from the original array into the new array. We can refer to the original array using \(b\) because whatever a was referring to, now \(b\) is referring to the same array. So \(b\) is now pointing to the array containing 1, 2, 3, 4, 5. So, 1st five elements from array \(b\), now we are copying into array \(a\). After doing that the balance 5 elements in the new array i.e. \(a\), we are setting up them with value 20. After that we say \(a = \text{null}\). If this is done, now array \(a\) has been deallocated. Means \(a\) is no longer pointing to this new array of 10 integers. Now if we try to print \(a[0]\), we get the exception saying that we are already get rid of the array in memory by setting \(a = \text{null}\). Array \(a\) is garbage collected by the Garbage Collector. Memory utilized by array \(a\) is recycled by the Garbage Collector. Hence, we have no longer access to \(a[0]\). So deallocation is possible by assigning null to the array reference whereas reallocation is possible by using \(a = \text{new int}[10]\) and then copying original 5 elements from array \(b\) into the new array \(a\). If we print \(b[0]\), we get the output as 1.

Problem

Now we want to find out what we can do if we are required to create an array by receiving its size and its values. Then what we need to do to print the values. Suppose we define \(\text{int} \ n\) and then we ask the user to enter the size of the array. Once the user enters the size, receive it in \(n\) and then we say \(\text{int}[\ ] \ a = \text{new int}[\ n]\). If \(n = 5\) then a new array of 5 integers would get created. \(a\) would then start to pointing to that array. \(a\) is a reference to the array and created on the Heap. If we want to print the elements of this array then we say for \((i = 0; i < a.\text{length}; i++)\). Within the loop, we receive the
value, store it in the array and print that out. After that if we wish to increase the size of the array by 3 elements and then receive all the elements once again then we say a = new int[ n + 3 ]. Means a new array would be created and in this array, there would be n + 3 elements. So whatever we had supplied the original size of the array, we are expanding that array by 3 elements. Once again we run the same for loop, receive n and store it in the array and then printed out. a.length gives the number of elements present in that array or size of the array. Whenever we use a.length, we are leaving it to the system to figure out what is the length of the array and it’s a good idea. In memory, suppose initially there would be an array of 4 integers. a would then point to this array and suppose we had supplied numbers like 11, 4, 2 and 9 to them. When we do a = new int[ n + 3 ], a would not no longer point to this array; it would start pointing to another array of n + 3 integers. Into this new array, we are receiving a completely new set of values i.e. 5, 4, 6, 1, 0, 7 and 9. So the Garbage Collector wipes out the original array and recycles the memory.
Arrays - III

In this lecture you will understand:

* Passing array elements / array
* Returning an array
* Two-Dimensional Arrays
* Initialization of 2D Array
* Passing 2D Array
* Returning 2D Array
Passing Array Elements

Suppose we have a class Array inside main( ) we have an integer array a holding values 7, 9, 16, -2 and 8. We know how the array is laid out in memory i.e. similar elements in adjacent memory locations as shown in the slide. Array would be created in the Heap; the reference a would be created on the Stack. The reference holds the address i.e. in this case it is 102. Then we would call a display() function and to the display( ) function, we would try to pass individual elements of the array by using display (a[0], a[1], a[2], a[3], a[4]). Then we would write the display( ) function. In the display( ) function these a[0], a[1], a[2], a[3] and a[4] are collected in variables like i, j, k, l and m. After that we are printing their values in the display( ) function. In place of doing it in this way, we can also run a loop for (i = 0; i < a.length; i++) and each time through the loop, we can call a function display1( ) to which value of a[i] is passed i.e. display1 (a[i]). In the display1( ) function, whatever value is passed is collected in variable n and its value is printed in the display1( ) function. In this program, the display( ) function seems to be better because we are required to call display( ) only once whereas display1( ) is called at least 5 times which consumes some amount of time. However, display( ) suffers from a limitation that now there are 5 elements, hence we are using a[0], a[1], a[2], a[3], a[4]. But if numbers of elements become 500 then passing those many number of elements become very tedious. Means even though display( ) is fast, its call is not compact whereas display1( ) is slow but the call that we are making to display1( ) is very compact. But we want best of both i.e. we want the call to be compact at the same time we want the call to be fast. So we will pass entire array itself to the function.

Passing Entire Array

In the PassingArray class inside main( ), we first define the array a as int a[] = { 7, 9, 16, -2, 8 } and int b[] = { 1, 3, 6, -2, 18, 1, 1, 7, 1, 1 }. Then to the display( ) function we pass the entire array by just passing a i.e. display (a). Inside the display( ) function, we can receive this in some variable p which would also be the reference to the exactly same array. Because a is containing the base address of the array 102, same base address is collected in p and p also becomes the reference to the array of integers. Now in the display( ) function, accessing elements is easy as p is reference to the same array of integers. So using loop for (i = 0; i < p.length; i++), we will print the value of p[i] each time through the loop. Then in main( ), we pass array b to the display( ) function using display (b). When we pass b, once again we collect the reference to the new array in p and then again using for loop, print all the elements of the array. So in this way display( ) function becomes generic, no matter there are 5 elements or 10 elements, we can print the elements using the same display( ) function. Means any time we want to communicate with an array, then communicate its reference. Do not pass individual elements because it’s a time consuming.

Returning An Array

In main( ), we say int p[] = fun( ). Then from fun( ), we are going to return the array a. In the definition of fun( ) we say public static int[] fun( ). The int[] indicates that fun() is going to return an array. Inside this function fun( ) we are initializing array a by saying int a[] = { 7, 9, 16, -2, 8 } and then we are returning an array by using return (a). The a that is returned is collected in a variable or in array p. Once it is collected in p, we can access the elements of the array. To print the elements of the array we use for loop, for (i = 0; i < p.length; i++ ) and then print the elements using p[i]. When we do p[i], we are going to refer to exactly same elements of the array a which is created within the function fun().

2-D Array

Suppose within main( ) we have set of elements as shown in the slide. So rather than creating 4 single dimensional arrays, we can create a 2-dimensional array. For 2-dimensional array, we use two pairs
of brackets and put one more pair of braces. We must put , at the end of each of the set of elements. , is compulsory. Braces around one set of elements are also compulsory. Once the array is initialized, now if we want to print out 21 then we say System.out.println ( a[ 2 ][ 4 ] ) because rows and columns are always counted from 0 onwards. The first value in the bracket is always stands for the row number and next value is going to stand for the column number. So 2 is the row number and 4 is the column number. But if we wish to print all the elements of the array, then we need two loops; one for managing i and one for managing j. So we would run the outer loop as for ( i = 0 ; i < a.length ; i+ + ) and inner loop as for ( j = 0 ; j < a[ i ].length ; j++ ) and then inside the inner loop we say System.out.print ( a[ i ][ j ] + " " ). A 2-D array is an array of several 1-D arrays. When i = 0, a[ 0 ].length gives length of the 0th 1-D array i.e. 5. So, j loop runs from 0 to j < 5. In this way we can access all the elements of the 0th row. When all the iterations are completed, elements of the 2-D array get printed out row after row. To ensure that the elements of the array would get printed out in the form of a table, so after coming out of the j loop we simply execute System.out.println( ) such that elements of the next row would start getting printed on the next of the screen. 0th element of the array is 2, 6, 1, 8, 4 and not only 2 because 2-D array is a collection of several 1-D arrays.

Find Biggest…

Suppose we have a 2-D array as shown in the slide. We wish to find out the biggest element of this array which is 50. We begin by saying 7 is the biggest element this is our assumption. Once we assumed that 7 is the biggest number then we can search the array for a number which is greater than 7. The moment we come across 8, we will forget 7 and memorize 8. Once we memorize 8 then we search for a number which is bigger than 8. The moment we come across 9, we will forget 8 and memorize 9. Once we memorize 9 then we search for a number which is bigger than 9. The moment we come across 50, we will forget 9 and memorize 50. Then there is no number which is bigger than 50. So we know that 50 is the biggest number. Inside main( ) 7 is accessed by using a[ 0 ][ 0 ]. So we assume big = a[ 0 ][ 0 ]. To search for an element we have to go through each and every element of the array. So we run two loops. The outer loop is for ( i = 0 ; i < a.length ; i++ ) and the inner loop is for ( j = 0 ; j < a[ i ].length ; j++ ). Inside the inner loop, we check if ( a[ i ][ j ] > big ) and if it is true then we assign that number into big by saying big = a[ i ][ j ]. Outside the for loop, we will print the value of big. In addition to this if we want to find the biggest number’s position we begin it by saying r = 0 and c = 0. Inside for loop, after checking the condition the way we assign the number a[ i ][ j ] in big, likewise we assign i and j in r and c i.e. r = i and c = j. After coming outside the for loop, we can print the position of the biggest number by printing values of r and c.

Initialization of 2-D Arrays

We can initialize a 2-D array by saying int a[][] = {{ 7, 2, 6, 1 }, { 9, 3, 4, 5 }, { 10, 12, 16, 18 }} ;. In memory array will always be laid out row after row after row. Means firstly 7, 2, 6, 1 will be arranged. Immediately after that in the next memory locations 9, 3, 4, 5 will be arranged and immediately after that 10, 12, 16, 18 will be arranged. In the Stack, there would be only reference to this array. This way of arranging elements in memory is known as Row major arrangement. If 7 is stored at location 402 then the next number i.e. 2 will be at location 406. Hence, 9 is stored at location 418. Next row begin at 434. We can also define the array as int a[][] = new int[3][4] indicating that a is going to refer to an array which has 3 rows and 4 columns and then we can initialize individual element as shown in the slide like a[0][0] = 7 ; etc. We can also define an array using the syntax int[][] a = new int[3][4] where a is a reference to the 2-D array of integers which has 3 rows and 4 columns.

Passing 2D Array

Suppose inside class TwoD within main( ) we have a 2-D array int a[][] = { { 1, 2, 3 }, { 4, 5, 6 } }. Then we say sum = fun ( a ). The function fun( ) performs the sum of all elements get done, return that sum and collect it is variable sum and inside main( ) we can print that. In fun( ) when we pass a,
we have collected it in b[ ][ ] where b is a reference to a 2-D array. After that we can run two loops inside fun( ), one for row elements and another for column elements. The loops are for ( i = 0 ; i < b.length ; i++ ) and the inner loop is for ( j = 0 ; j < b[ i ].length ; j++ ) and inside that we can keep getting a running sum by saying s = s + b[ i ][ j ]. After coming outside the for loops, we can return the sum by using return ( s ) which is collected in variable sum in main( ). The sum would be turn out to be 21.

Returning 2D Array

Within class TwoD we have main( ). Inside main( ), we have a reference to a 2-D array i.e. int b[ ][ ]. Then we say b = fun( ) which indicates that fun( ) to return a 2-D array reference which is collected in b. Then to check that we have a reference to a 2-D array using which we can access the elements of the 2-D array, we are printing b[ 0 ][ 0 ] and b[ 1 ][ 1 ]. We define the fun( ) function as public static int[ ][ ] fun( ) which indicates that fun( ) is going to return a reference to a 2-D array. We initialize the array b by saying int b[ ][ ] = { { 1, 2, 3 }, { 4, 5, 6 } }. Then we run two for loops; for ( i = 0 ; i < b.length ; i++ ) and for ( j = 0 ; j < b[ i ].length ; j++ ). Inside the inner loop we say b[ i ][ j ] += 20 which adds 20 to each 2-D array element. Finally outside the for loop we will return the array b by using return ( b ). The b we are returning is collected in main( ) once again in reference b and then we print values of b[ 0 ][ 0 ] and b[ 1 ][ 1 ].
Arrays - IV

In this lecture you will understand:

* 3D Arrays
* Array of References
* Jagged Arrays
* Passing Jagged Array
* Returning Jagged Array
* 3D Jagged Array
3D Arrays

3D arrays would also get created in the Heap. A 3D array is a collection of several 2D arrays. So, one 2D array contains elements like 2, 7, 8 and 1, 4, 3. These are 2 rows and 3 columns of a 2D array. There can be one more 2D array behind this array. Then there are still two 2D arrays behind it as shown in the slide. 2D array is a collection of several 1D arrays. Tough we imagine that these array are stored one array behind another array behind another array and so on but still in memory there are stored adjacently or in a linear fashion i.e. in a Row major fashion. Means 2, 7, 8 will be set in memory, just besides that 1, 4, 3 and so on. A reference a is pointing to these collection of elements which is created in the Stack. So the first number’s i.e. 2’s base address is stored in reference a. The way we have initialized 2D arrays, we can initialize 3D arrays. After initialization, we want to print out the element a[3][1][2]. In this 3 stands for the 3rd 2D array, 1 stands for 1st row and 2 stands for 2nd column. When we print the value of a[3][1][2], we get the output as 6. Then we also want to print a.length, a[0].length and a[0][1].length. When we do a.length, we get the output as 4 since there are four 2D arrays are present in a 3D array. When we do a[0].length, we get the output as 2 since there are 2 elements in the 0th 2D array. When we do a[0][1].length, we get the output as 3 since in the 0th 2D array, in the 1st row we have 3 elements.

Accessing

We had taken the same array as it was in the previous case. A 3D array having a collection of four 2D arrays. If we want to access and print all the elements of a 3D array we run the outmost loop as for (int i = 0; i < a.length; i++). Now in each 2D array we have to go all the rows present in it. So we run the inner loop as for (int j = 0; j < a[i].length; j++) and the innermost loop runs as for (int k = 0; k < a[i][j].length; k++), inside that we print the elements by saying System.out.print (a[i][j][k]). To go to the next row on the screen after printing the elements of the one row we say System.out.println() which would work only when control goes outside the k loop. This is one way of accessing elements of a 3D array. There exists one more way using which we can do the same functionality. We say for (int arr2d[][] : a). Inside this for loop, we say for (int arr1d[] : arr2d) and inside this we say for (int item : arr1d). Inside this for loop, we say System.out.print (item). Once outside the for loop, to go to the next line on the screen say System.out.println(). We are going to print int values so item is an int. item is going to take the values that are fed to it by arr1d. arr1d gives values that it gets from arr2d. arr2d gives all the values that it gets from a. Means from reference a, we get 4 values saying that there are 4 2D arrays. arr1d will get each of the row of every 2D array and item would get each of the integer that is present in each 1D array. So we can access all the elements of the array using this syntax also.

Array Of References

Suppose on the Heap, we have an array of several Sample references where Sample must be the class. On the Stack, there would be references which points to this array of Sample references where each reference will be reference to some or other Sample objects. If we want to create an array of such Sample references where each reference points to some or other Sample objects, we can say Sample s[] indicating that s is an array of references. We can initiate that array by saying s = new Sample[5] and then we can say for (int i = 0; i <= 4; i++) and within the loop we say s[i] = new Sample(). As each new Sample object gets created, its reference goes into the array and the reference s which is there on the Stack is pointing to the array of Sample references. This is also a thought of an array of Sample objects. Ultimately objects are created in Heap and whatever is created in Heap is never in the adjacent locations. Hence the Sample objects that we create, their references we have to gather together into an array of Sample references.
As shown in the slide, it's an Auditorium. We want to write a program which represents the working of an Auditorium. May be we want to do ticketing or some other job on the auditorium indicating that how much collection we can have from this auditorium. If we want to repair it then how do we track, what chairs are to be repaired in the auditorium, so on. Basically we need a good way to represent each and every seat which is in this auditorium. Auditorium seats are arranged in different rows and columns. So we can use a 2D array. But problem is that each row of this auditorium has never the same number of seats. Auditorium is always a trapezoidal in shape wherein number of seats in the front row is exactly not equal to the number of seats in the last row. The number of seats always reduced from last row to first row. Means if number of seats in each row are different then we are talking about an array where it’s a collection of several 1D arrays but then each 1D array is not similar. Whenever we need that representation, a normal 2D array cannot be used. Because in a normal 2D array, its assumed that every single 1D array within the 2D array is exactly similar. So the 2D arrays that we had seen and used are known as Rectangular arrays. So, rectangular arrays not going to serve the purpose. We need Jagged arrays. If we use Jagged arrays then it gives the provision that number of columns in each row may or may not be same.

Jagged Arrays

There are three rows in a Jagged arrays where 1st row has 4 elements, 2nd row has 3 elements and 3rd row has 2 elements. Reference of each one of these rows is being stored in another array and to this array of references, reference a is pointing which is on the Stack. a is a reference to an array of references and a[0], a[1], a[2] is reference to 1D array. To create this array, we have to say int a[ ][ ] = new int[3][ ]. 3 indicates that in this 2D array, there are going to be 3 rows. The number of columns in each row is we have to assign separately by saying a[0] = new int[4] ; a[1] = new int[3] ; a[2] = new int[2]. Means in the 0th row there are 4 columns, in the 1st row there are 3 columns and in the 2nd row there are 2 columns. Then we assign the values to each of the slots present in 1D array by saying a[0][0] = 7 ; a[0][1] = 2 ; etc. as shown in the slide. Then to access and print all the elements of this Jagged array, we once again run a loop, for ( int i = 0 ; i < a.length ; i++ ) and the inner loop runs as for ( int j = 0 ; j < a[ i ].length ; j++ ). Inside the inner loop, we can print the elements of the Jagged array as System.out.print ( a[ i ][ j ] ) ; and once outside the j loop, we can put the cursor next line by saying System.out.println( ).

Passing Jagged Arrays

Suppose the program contains a Jagged array as shown in the slide which we wish to pass to a function. Inside main( ), first we will create the Jagged array using int a[ ][ ] = new int[2][ ]; a[0] = new int[4] ; a[1] = new int[3]. In the 0th row we are creating 4 elements and in the 1st row we are creating 3 elements. Then as shown in the slide, we are setting up values in each of the rows. Then we want the sum of all the elements present in the Jagged array and we can do so by using function fun( ). So we say sum = fun ( a ). The calculated sum of elements is returned in variable sum and finally we will print the sum obtained. We write fun( ) as public static int fun ( int b[ ][ ] ). We are collecting whatever we pass to fun( ) in int b[ ][ ] indicating that b is a reference to a 2D array. Within the fun( ), we do running sum for that we run a loop. The outer loop runs as for ( i = 0 ; i < b.length ; i++ ) and the inner loop runs as for ( j = 0 ; j < b[ i ].length ; j++ ) and inside the inner loop the sum is calculated as s = s + b[ i ][ j ]. Inside fun( ), set s = 0. Finally outside the loop, we will return sum by saying return ( s ) which is collected in main( ) in variable sum and finally print the sum.

Returning Jag. Arr.

Within main( ) we define an array int b[ ][ ] saying that it’s a reference to a 2D array. Then we say b = fun( ). We will expect that fun( ) should return us a Jagged array. Then we print out the elements whatever we received through the set of for loops. We define fun( ) as public static int[ ][ ] fun( ) saying that its going to return a reference to a 2D array. Inside fun( ) we create the Jagged array using
Arrays - IV

Java Programming

```java
int a[][] = new int[2][ ]; a[0] = new int[4]; a[1] = new int[3]. 0\textsuperscript{th} row contains 4 elements whereas 1\textsuperscript{st} row contains 3 elements. Then we initiate values into these rows as shown in the slide. Then we return a. In this way we can return a 2D Jagged array back from a function. There is no difference in the way we treat a 2D rectangular array or a 2D Jagged array. The difference is only in the way we create these arrays. Once the array is created, the way to access them, the way to pass them and the way to return them are exactly same.

3D Jagged Arrays

A 3D Jagged array would be a collection of several 2D Jagged arrays and a 2D Jagged array would be a collection of several rows where each row has different number of elements. Suppose we wish to create 3D Jagged array where there are two 2D arrays in the 3D Jagged array. The 0\textsuperscript{th} 2D array is going to have 3 columns in one row and 5 columns in another row whereas the 1\textsuperscript{st} 2D Jagged array is going to have 3 rows containing 2 columns in 0\textsuperscript{th} row, 4 columns in 1\textsuperscript{st} row and 3 columns in 2\textsuperscript{nd} row. Means in the Heap, we can imagine it in the way as shown in the slide i.e. there are 2 rows one having 3 columns and another having 5 columns and the references to these would be stored in some other array. Then the slide shows another array having 2 columns in 0\textsuperscript{th} row, 4 columns in 1\textsuperscript{st} row and 3 columns in the 2\textsuperscript{nd} row. References of these different rows we have created will be stored in another array. So we have two 2D Jagged arrays. The references of these array of references that we have created would be stored in yet another array. Reference of this array of array of references would be in the Stack. So arr would be pointing to this array of two elements which in itself is a array of references. But these array of references, each element of it does not point to some integer; it points to two different arrays of references which in turn point to 2 different 1D arrays. If we want to create such 3D Jagged array then we say int arr[][][] = new int[2][][]. Then we say

```java
arr[0] = new int[2][];  
arr[1] = new int[3][];  
arr[0][0] = new int[3];  
arr[0][1] = new int[5];  
arr[1][0] = new int[2];  
arr[1][1] = new int[4];  
arr[1][2] = new int[3];
```

In this way we can construct a 3D Jagged array. arr is a reference to an array of references where each reference in this array in itself points to another array of references and each one of those arrays point to several 1D arrays. So arr is a reference to array of array of references whereas arr[0] is a reference to an array of references and arr[0][0] is a reference.
Using NetBeans Effectively

In this lecture you will understand:

* What are Edit Hints
* How to Navigate
* What is Code completion feature
* What is code Refactoring
* How to achieve Refactoring in NetBeans
Creating New Project

Whenever we start NetBeans, we start by creating a new project. Start new Java Application and give the name of the project. In addition to that, we can also give the name of the Main class. In the slide, the name of Main class is Main. Name of the project is UsingNetBeans and usingnetbeans.Main is the Main class name. We can give any other name for the Main class of the project. Within that class the main( ) method would be written. Name of the package is usingnetbeans. Then click on Finish.

NetBeans Views

When we click on Finish, we get the screen as shown in the slide. Name of Main class is Main. NetBeans allows to view a project in two different ways; one is a Project View and another is a File View. In the slide, UsingNetBeans is being shown in the Project Views tab along with different source packages, names of the classes, for example, Main.java is shown here. Click on the Project View, go to the particular project and then if we select Main.java then we can see the code present in Main.java.

Add New Class

In the bottom left corner two things are shown; one is name of the constructor and a method main( ) which receives an array of Strings. We can add a new class to the project. For that go to the project UsingNetBeans. Name of the class is Main. Right click on the name of the project i.e. UsingNetBeans, select New and then select Java Class…. When we do so, we get a dialog as shown in the slide. It requires the name of the class. We give the class name as Utility. Then it requires the name of the package to which this class will belong. We are using the package name as usingnetbeans to which the Main class is belonging. We can create the new class in a completely different package. Then click on Finish.

Utility.java – Add throws

We get a screen as shown in the slide when we click on Finish. We have a statement package usingnetbeans at the top. This indicates within the package in which Main class is present, exactly in the same package the Utility class is also inserted. Within the Utility class, we have a constructor. Suppose we want to add a throws clause to this constructor.

Utility.java

For that we can type th and then space in the constructor. The moment we do so, throws will get added to the Utility constructor. Then we can select throws Exception. When we do so within Utility.java, throw new Exception ( "Not implemented" ) line that code snippet will also get added. This has been done by NetBeans 5.5. At the top, the files are shown with which we are working; one is Main.java and another is Utility.java. Click on Main.java.

Edit Hints – try-catch

If we click on Main.java, we will come to the contents of Main.java file. Suppose we want to add try-catch block to the contents of Main.java which is shown in the slide. We have to type try-catch and press CTRL and hit spacebar.

Edit Hints – try-catch

When we do so, a bulb is shown on the left hand side of the code window. Click on the bulb on the left hand side.

Edit Hints – try-catch
If we click there, it will show two different hints which are known as edit hints. It tells that we are required to either add a throws clause or we can surround this code with try-catch. We select one of these. We are surrounding the code with try-catch. So click on Surround with try-catch.

**Edit Hints – try-catch**

When we do so, the try-catch block has now been inserted in the code. So Utility u = new Utility( ) gone within the try block and catch block says catch ( Exception ex ) and within the catch block we say ex.printStackTrace( ). It is a function of the Exception class and it will print out the stack trace as and when an exception occurs. We get edit hints, the bulb that was shown gave two hints; surround the statement either a try-catch or throw an exception using a throws clause.

**Edit Hints 2 – import**

Within the same code, if we say File f = new File ("hello.txt" ). When we do this, once again the edit hint is shown. File is a class which is present in a package java.io. If we are using a File class, at the top, we have to say import java.io.*. This hint is shown to us through the bulb that is present at the left hand side. If we click on that hint, it will say Add import for java.io.File. Select that hint.

**Edit Hints 2 – import**

If we select that hint then within the code at the top import java.io.File has been added. So in the slide that statement is marked with the rectangle. This indicates that, we do not have to remember that while using a particular class to which particular package that class has belong.

**Navigation – Ctrl + Mouse**

If we position the cursor at any of the class or any of the method that belongs to a class and then depress the CTRL key, we will find that, that particular class name or method name get underlined. In the window as shown in the slide, the word Utility has been underlined. This is because we position the mouse there and then depress the CTRL key. We are working in Main.java and when Utility word gets underlined using Mouse and CTRL key, if we click on the Utility word which is underlined, we are taken to the Utility class wherever that class stands defined in the project. This is also true about the methods, variables, fields, etc. This allows to navigate between different code snippets quite fast.

**Navigation – Ctrl + Mouse**

If we want to go back to Main, then we have to follow the following procedure. Within the Utility class Main is not mentioned. So we cannot use CTRL and Mouse. So at that time, we have to make use Left arrow key (←) as shown in the slide. If we click on that, we will be taken to the place from where we landed into Utility class.

**Code Completion – Step 1**

Suppose as shown in the slide, we have written that much code to main( ). Then we create an object of the File class by File f = new File ("hello.txt" ). We want to call method of the File class. For that we have to say if ( f ). When we do a . (dot) after f, a completion list will appear in the form of a window. This can be compared with the intellisense feature that Microsoft offers with Visual Studio. In NetBeans it is known as a Code Completion feature.

**Code Completion – Step 2**

When we type a f and a ., it indicates that we want to call the method of either the File class or its base class’s method. Hence list of all the methods is shown to us as shown in the slide. It also shows the return types and the parameters of the methods. We can also navigate between them using the
highlighted bar. We can move up and down and then select one of them. While making the selection, even if we do not know the purpose of any of the method then using the highlighted bar we can select the method and at the bottom we can see the documentation for the respective method. So on whichever method we are, corresponding help is shown at the bottom. We can select the method by clicking the mouse or hitting the Enter key. Now we are using the exists( ) method.

**Code Completion – Result**

When we select the exists( ) method, we can see that the code has been suitably completed for us. So there is if ( f.exists( ) ). Then we can add System.out.println( f.getName( ) + " exists" ). We can obtain getName( ) using the code completion feature. It might happen that there are multiple methods which begin with word get. In that case we can continue the typing if we are already aware of these names.

**Code Completion – Multiple**

We can think of making use of the code completion feature in multiple ways. For that we have to type CTRL + spacebar.

**Slide 20**

When we do so, once again suggestions will come. We can make selection. Before making a selection, we can go through the documentation which is at the top. Once we have done that, we would be able to get automatically the code completed within the program.

**Refactoring – Rename**

Suppose we have a word Utility for the class. But after creating this class, we might think that the name should be something else and not Utility. Then we would want to rename this. To change the name of the Utility class, we make use of NetBeans. For that, we position the cursor on Utility and then do right click. When we do a right click, a menu would appear. Select Refactor from the menu that appear. The moment we select Refactor, one more menu would appear. In that menu the 1st choice is Rename… which indicates that we can give a new name to the Utility class. Not only the class but also all the references to this Utility class wherever they have been done throughout the project, they also should change. This is known as Refactoring. So we select Rename… from the menu that appears.

**Slide 22**

When we select Rename…, a dialog will appear which asks for the new name. The present name is Utility. We provide the new name as UtilityNew. Then click on Next.

**Slide 23**

The moment we do so, we will find that wherever there was a word Utility, it will change to UtilityNew when we Do Refactoring. So click on Do Refactoring.

**Slide 24**

When we click on Do Refactoring, we will find that wherever there was a Utility, it has now changed to UtilityNew. So UtilityNew u = new UtilityNew( ) is shown in the slide. Also as shown in the slide, name of the file is also changed to UtilityNew.java. This is refactoring. Every single occurrence even the name of the file is also changed when we make use of refactoring which is a facility to rename a particular entity.

**Configure – Tools | Options**
There are options which allow to configure the data. To do so, go to the Tools menu of the NetBeans. Select Options from there. When we do so, a dialog will appear as shown in the slide. It shows the changes that we would want to make, the configuration that we would want to set up may be for fonts and colors or keymap or the editor. The slide shows, how to do this with Editor. So click on Editor and then go to the Indentation option.

**Configure – Tools | Options**

In the Indentation option, there are several choices given to us. One of the option is Expand Tabs to Spaces. Until we select this option, taking the code from NetBeans into the PowerPoint slide is a headache because within the slide, when we try to put this code, there are space constraints for us. If we use tab, that time the code will indent too much to the right which we cannot accommodate. We can also specify the space that we want. This configuration we need to do. Otherwise we have to take the code to PowerPoint and replace all tabs with spaces physically. Then we also have a option, Add New Line Before Brace. Select that option and we also select Add Space Before Parenthesis option.

**Code Templates**

Then go to the Code Templates tab within the Editor. In this there are shortcuts are given for commonly occurring code. For example, if we want to type out Exception, we just have to type out Ex and hit the spacebar. Immediately it will expand to Exception. If we type th and space, it will expand to throws. If we say wh and a space it will become while. So these are the common standards for commonly occurring code snippets. We can add our own feature using the New option shown in the slide. We can do this for other languages like XML, SQL, etc. We are using Java, so we re selecting language as Java.
In this lecture you will understand:

* Need for using Bitwise operators
* One's complement operator
* Bitwise &, |, ^ operators
* Bitwise <<, >>, >>> operators
Bits etc.

Binary digit is a basic unit of information. Any bit can take a value either 0 or 1. 4 bits taken together are commonly known as nibble, 8 bits together forms a byte whereas 16 bits forms a word and 32 bits form a double word. Binary numbering system has two digits in it i.e. 0 and 1 whereas octal can support as many as 8 digits numbered from 0 to 7. Decimal numbering system has 10 digits numbered from 0 to 9 and Hexadecimal has 16 digits numbered from 0 to 9 and A to F. Frequently we are required from Binary to its Decimal equivalent because Java does not understand Binary; Java understands Decimal, Octal and Hexadecimal. Binary to Decimal is inconvenient. For example, if we take a binary number 0111 0110 then to find out its decimal we have to do 2 + 4 + 16 + 32 + 64. Means each one of these digits we have to multiply with suitable powers of 2 and then add the result to get the decimal equivalent of 0111 0110. Likewise we can find the decimal of 1111 1010 is 2 + 8 + 16 + 32 + 64 + 128. Hence instead of converting from Binary to Decimal it is more convenient to convert from Binary to Hexadecimal. The Hexadecimal equivalent of 0000 is 0, 0001 is 1, etc. as shown in the table in slide. 4 bit nibble form one Hexadecimal digit. For example, the Hexadecimal of 0111 0110 is 76. Similarly the Hexadecimal of 1111 1010 is FA. So we can convert Binary into Hexadecimal in much more convenient way as compared to conversion of a Binary into a Decimal.

Need For Bitwise Operators

Suppose we take int ch = 32. Now ch is a 32 bit number where the bits are numbered from 0 to 31 from right to left. Bit 0 is the rightmost bit or the lowest bit whereas bit 31 is the leftmost bit or the highest bit. We may want to go with these 32 bits and want to check whether a particular bit has value 1 or value 0 or we may want to go to some other bit and set either a value 0 there or set a value 1 there. Smallest entity that we have been able to operate upon in Java is a byte. We could not go anything lower than a byte. We can do that checking for a byte also. All these possibilities exist.

One’s Complement Operator

If we take int ch = 32. In ch, a 32 bit number is stored which is binary equivalent of 32. Binary equivalent of 32 is 100000. All the 0s to the left of 1 are insignificant 0s. If we try to do one’s complement on 32 then one’s complement in Java is represented using ~ ( tilde ) symbol. When we apply ~ operator on int ch = 32, all 0s will be converted into 1s and all 1s will be converted into 0s. So as shown in the slide, in the binary representation, we have only one 1 in binary of 32 that becomes 0 and all others have become 1. We can print ~ch and we will find that it’s a negative number because the leftmost bit in this number is 1. If leftmost bit is 1 then the number is always a negative number. Negative number’s binary is stored as 2’s complement and we get 2’s complement by adding 1 to the 1’s complement. So whatever number we have obtained by applying 1’s complement to 32 if that is treated as a negative number and we wish to reach the original then from these 2’s complement, we must subtract 1 to get the 1’s complement. If we do the subtraction then we get the number as shown in the slide. The number obtained whose equivalent 1’s complement will be 00000000000000000000000000000001 which in decimal turn out to be 33 and since there is 1 in the leftmost bit, it is treated as -33. So if we attempt to print 1’s complement of ch which has value 32 then we get the output as -33. Using 1’s complement, we are able to operate upon individual bit and change its value.

And, Or, Xor

& operator is known as Bitwise And operator. | is known as Bitwise Or Operator and ^ is known as Bitwise Xor operator. At the top left corner in the slide, there is a truth table of the & operator. In the table, vertically as well as horizontally we have 0 1. These bits we are trying to And. The four numbers i.e. 0 0, 0 1 are the results obtained by Anding 0 1 present vertically with 0 1 present horizontally. From the table, if both the bits are 1 then on Anding we get the result as 1. If any one of
them or both of them is 0 then the result is going to get 0. Suppose we have a number as shown in the slide and we want to do & operation between such numbers. When we carry out & operation we get the result as shown in the slide. 1 & 1 only gives 1. So on the original value, we want to apply a & operation on that. With whatever we apply the operation, it is known as a & Mask and the final value get is the New value. Similarly we can use | operator. The truth table for | operator is given. Anything Ored with 1 results into 1. Only both the bits have value 0, then only we get the result as 0. We are doing a Bitwise | operation on a number as shown in the slide. We find that after applying a | mask to the original value the new value obtained is shown. Same can be done for Bitwise ^ operator. The truth table for ^ operator is shown in the slide. The truth table for ^ operator is similar to truth table for | operator except that 1 ^ 1 = 0. Once again we take a number and do ^ mask on it and the result obtained is shown in the slide.

Utility of & and |

Suppose we enter some number through the keyboard. We receive the number may be in the smallest entity i.e. byte n. Suppose n which is a byte is represented as shown in the slide which has 8 bits numbered from 0 to 7 from right to left. Out of all the bits e are interested in bit 3. Now we want check out whether this bit contains a value 0 or does it contain a value 1. We do not know what is stored in all other bits but we are interested in bit 3. We will convert all other bits to 0s. We should & that bits with value 0 whereas bit number 3 we & with a value 1 because we do not know what is the value present in bit 3. If we & it with 1 then in the result if we get 1 then we can conclude that in the original number n also bit number 3 contains a value 1. Final result obtained, its Decimal equivalent is 8 whereas the Mask we are using, its Decimal equivalent is 8. So we can write the condition as if ( n & 8 == 8 ) and if it is true then we can conclude that the 3rd bit in the original number n must be 1. So we can report it by saying System.out.println ( "3rd bit is on" ). So the utility of & operator is to check a particular bit in a number is on or off. Now since the b it is on, we can sure that bit number 3 contains a value 1. We are unsure about the other bits because we are not checked what are the values in the other bits. Now we can try to convert the 1 in the 3rd bit into 0. For that we should carry out & operation once again. To do so, we should and that bit with value 0 i.e. 1 & 0 = 0. While doing so, we want the other bits should remain undisturbed so and them with 1s. For that we say n = n & 0xF7 and then print value of n. Now if ( n & 8 == 8 ) fails then control goes to the else block. In the else block, we wish to put on a particular bit. For example, if 3rd bit already off, we want to set it on. To set it on, we need to OR it with 1. But once again while doing so, other bits should remain undisturbed. So, OR them with 0s. We can do so by saying n = n | 8. The utility of | operator is to put on a particular bit. The utility of & operator is to put off a particular bit and it allows to check us whether a particular bit is on or off. We can also write n = n | 8 as n |= 8. |= is known as Bitwise compound assignment operator. There also exists operators like &=, |=, ^=, <<= and >>=, all are Bitwise compound assignment operator.

Right Shift >>

Suppose we have int ch = 32. Then we say System.out.print ( ch >> 2 ). When we right shift 32 by 2, it will eliminate the rightmost two digits from it. So rightmost two bits are gone and then all other bits to be shifted two positions to the right. When they are shifted two positions to the right then two empty slots are created on the left. These empty slots are filled with 0s. So the resultant number on doing ch >> 2 is 8. When we do ch >> 1 we get 16 and ch >> 2 we get 8. But even after doing so, original value of ch still remains 32 because by right shifting 32 by 2, we are not assigning a value to ch. So ch >> 1 means dividing the number by 2 but if we have some remainder then that remainder will get discarded. Sign of a number not changed by doing a right shift. If we do 35 >> 2 then we get answer as 8.

Left Shift <<
If we take int ch = 32 and then do System.out.print ( ch << 1 ). Once we do that, the leftmost bit will be gone. All other bits should be shifted one position to the left and a 0 will be added in the empty slot created at the right hand side. Resultant number will be 64. Left shifting by 1 means multiplying the number by 2. However, ch would remain 32. If there is 1 in the highest bit and we do left shift by 1 then 1 will be lost. In that case result will not be multiplying the number by 2. Sign is not changed but sign may change in some situations. If the number is 01000000 and we do << 2 then 01 will be shifted out and resultant number will be 0. In this case 1 would be lost. If we take a number 01000000 and do << 1 then the result is 10000000 means the sign of the result has been changed.

Unsigned Shift

Suppose we have int ch = -1 and we do ch >> 1. When we do so, we get the result as -1. Binary of -1 is shown in the topmost box in the slide. When we do ch >> 1, the result is still -1 because the sigh gets extended. In the leftmost bit, there is 1, so it will continue to remain 1 when we do right shift. If we do ch >>> 1 then the result obtained in the leftmost bit, there would be 0. Means the sign will not be extended any more. So the resultant number will be 2147483647. ch is still -1. >>> will not extend the sign. >>> should be used only with integers or long integers. byte and short are anyway promoted to integers.

Utility Of <<, >>

Whenever we attempt to store a file on the disk, we give the name of the file. But the operating system stores the name along with the size of the file, date of creation and time of creation of the file. The date and time are always stored as 2-byte entities. Suppose the date of creation of a particular file is 06/01/99. This appears to be a string of characters. But it never stored as a string of characters, it is always converted into a 2-byte number. The formula to convert is ( 1999 – 1980 ) * 512. So ( Year – 1980 ) * 512 + Month * 32 + Day. When we do this, we will find that the final number is 9766 which is a 2-byte entity and it is stored on the disk. Binary of 9766 is 00100110 0001 00110. In the leftmost 7-bit, year of creation of a file is gone. Middle 4-bits contain the month of creation whereas rightmost 5-bits contain the day of creation of the file. Decimal of 00110 is 2 + 4 = 6 i.e. the day. Decimal of 0001 is 1 i.e. the month whereas decimal of 001011 is 1 + 2 + 16 + the base year ( 1980 ) = 1999 i.e. the year. The way there is a distribution of 7, 4 and 5 for the day, month and year likewise there is a similar distribution for time stored. It takes 5-bits for hour, 6-bits for minutes and 5-bits for seconds.

d, m, y

We can convert the number which is stored as a 2-byte entity back into a day, month and year. In a Date class inside main( ), suppose we have int dt = 9766. We want 9766 to get converted into day, month and year. Binary equivalent of 9766 is 00000000 00000000 010011 0001 00110. We first do >> by 9. If we do so, we get the resultant number as 00000000 00000000 00000000 0010011. Rightmost bits i.e. month and day will be wiped out. Year is shifted to 9 positions to the right. We can carry out this conversion by ( dt >> 9 ) + 1980 and we will convert the result into a short by type casting and assign it to y. Similar procedure we can do for month as well as day. So for month we do ( dt << 23 ) >> 28, convert the result into a short and assign it to m. For day, ( dt << 27 ) >> 27, convert the result into a short and assign it to d. Finally print the value of d, m and y which gives result as 6 1 1999.
Inheritance - I

In this lecture you will understand:

* Cornerstones of OOP
* Reuse Mechanisms
* What is Containership
* What is Inheritance
* Object sizes in Inheritance
Inheritance - I

Java Programming

OO Programming

The three important cornerstones of Object Oriented Programming are Encapsulation, Inheritance and Polymorphism. We had seen Encapsulation. Encapsulation hides the complexity of any operation that we intend to carry out. For example, we have a class Math inside which there are functions which give sin of a number, cos of a number or tan of a number, it gives ceiling of a number or floor of a value or gives the random number. All these complex operations that may be required to carry out before we get the result of a sin of a number or cos of a number, all of these are hidden away from us by putting them inside the class. Class lets us to encapsulate the different functions within it thereby hiding the complexity of any operation from a common user. Inheritance is a mechanism which permits to reuse existing pieces of software. Polymorphism can exist in two different forms, a compile-time polymorphism and a run-time polymorphism. When we do function overloading, we actually make use of polymorphism. Run-time polymorphism generalizes the actions. Each one of the Encapsulation, Inheritance and Polymorphism are facilitated by the classes and objects.

Reuse Mechanisms

Reuse can be done using two different mechanisms, one is Inheritance and another is Containership. Suppose we have a StringList class which is capable of maintaining a list of strings. If it is going to maintain a list of strings then there must be some or the other objects of the String class because each string is represented using a String object. If several such objects are going to be handled then we will need references to these different objects which we will have to maintain in a StringList class. Means we can say, a String object is contained within a StringList i.e. a containership where there is a Has a relationship between StringList and String. StringList has several strings within it. Since it enjoys a Has a relationship, it is known as a Containership. Strings are contained within a StringList. Unlike this, we have a Window and another window like Button. We can say Button is Like a Window. Means between Window and Button, the relationship is that of a Like a relationship or kind of relationship. When we say Button is Like a Window then that is an Inheritance relationship. When we say that it is a Inheritance relationship means whatever properties a Window enjoys, exactly same properties Button will enjoy. Then only we can say that Button is Like a Window. Non-object oriented languages permit Has a relationship. For example, C language permits Has a relationship. Only object oriented languages like Java, C++, C#, etc. which permit not only Has a relationship but also Like a relationship.

Combination of Relationships

Suppose we take three entities, a car, an engine and a vehicle. Then between these entities, if we have to establish a relationship then we can say that a car is like a vehicle. Means between car and vehicle there is a relationship is of Inheritance. As against this, the relationship between a car and an engine will be of Containership because engine is contained within the car. So car and engine enjoys containership relationship whereas car and vehicle enjoys an Inheritance relationship. The slide shows the Window that we get in NetBeans. In this Windows, we see there is a Main window, Project window where all the classes that are present in the project are listed, Error window at the bottom and we also have a Program window where we actually type the program. Different things are present in each of the window. For example, in the Error window, all the compilation steps are shown. In the Project window, different classes are shown. In the Program window, the entire code of the program is shown. But each specific window in itself is a type of a window. All these windows are contained within the Main window. We can say that Error window is Like a window, the Project window is also Like a window, the Program window is also Like a window. Means relationship between these three windows and a Window is a Like a relationship. But then we can say all these four Windows, Error, Program, Project and the Output Window are contained within the Main Window.

Inheritance
Suppose we have a class `Index` and inside class we have private int `count`. Inside class `Program`, we have `main( )`. Inside `main( )`, we say `Index i = new Index( )` i.e. we are trying to create an object of the `Index` class. Having done that the constructor would get invoked. In the constructor, we try to initialize `count = 0`. After that we say `i.display( )`. To make it work, `display( )` function must be present in the `Index` class. We write the function `display( )` in `Index` class and try to print out the latest value of `count`. When we create the first object, `count` is initialized to 0. So when we do `i.display( )` the expected output is 0. Then we call a function `incr( )` using the same reference `i`. We define the `incr( )` function in the class and inside `incr( )`, we increment the count by `count = count + 1`. Now if we do `i.display( )` then the incremented value of `count` i.e. 1 gets printed out. However, if we now want to do `i.decr( )` then the `Index` class that is written, we may not want to disturb. So we will create a new class `index1` by saying class `index1` extends `Index`. Means `index1` class extends the functionality of `Index` class. Right now `Index` class has not the functionality to decrement the value of `count`, we want to decrement the count. Hence, we extend the functionality of `Index` by providing the `decr( )` function. In the `decr( )` function, we decrement the value of `count` by saying `count = count - 1`. Now when we call `decr( )` using `i`, we expect the count should have been decrement to 0. But the program reports error. We cannot let the `int count` to remain private because in the `index1` class, we cannot use `count` since it is private. So, we will make `count` as protected which allows access of `count` to `index1` class but not in `main( )`. In Object Oriented terminology, `Index` is known as base class whereas `index1` is known as derived class. `index1` is derived from `Index` class means all the properties of the `Index` class now become available to the `index1` class. If we say `Index i = new Index( )`, it reports an error because since `i` is a reference to the `Index` class then we cannot say `i.decr()`. So we say `index1 i = new index1( )`. Means we make `i` as a reference to the `index1` class rather than `Index` class. Now since `i` is a reference to the `index1` class, `display( )` would get searched in the `index1` class. But it is not found there. Then it will go in the `Index` class, search the `display( )` function and satisfy the request for it. Same is happen with `i.incr( )`. `incr( )` is searched in the `index1` class and since it is not found there, it is searched in the base class and satisfy the request for it. `decr( )` is itself present in `index1` class and from here it would get executed. When we carry out inheritance between `Index` and `index1`, `display( )` and `incr( )` do not become part of `index1`. They remain in `Index` class. They would be executed from the `Index` class. But when we make the call, they are 1st searched in `index1` class and since they are not found there, searched in the `Index` class.

**Access**

`Index` class has private, public as well as protected data members or member functions. Create an object of the `Index` class by saying `Index j = new Index( )`. Public data members or public member functions of `Index` class would be accessible to this object. Protected and private are never accessible outside the class. Hence those cannot be accessed to this object. But if we create class `Index1` and extend it from `Index` then `Index1` itself has private, public and protected data members and member functions and if we try to create a new object to the `Index1` class by using `Index1 i = new Index1( )` then this object has access to the public data members. `i` cannot access protected and private of `Index1` class. However, all the functions of `Index1` class can have an access to protected as well as public data members and member functions of the base class i.e. `Index` class. Means even private member of the `Index1` class can have an access to protected and public members of the base class. Means `Index1` class can have an access to protected and public members of the base class. Object of `Index1` class can also have an access to public member functions of the base class.

**All Cases**

Base class is having all the three private, protected and public members. der class is extended from base class once again having all the three private, protected and public members. Everything from der class can have an access to protected and public of the base class. Private is never accessible outside the class. So in the class hierarchy, only protected and public are available in the der class. Suppose we have another class `Myclass` where we have reference to the base class and the der class and then
we propose to access private, protected and public of base class and private, protected and public of
der class. Suppose using b, we access private, protected and public of base class. private can never be
accessible whereas protected also will not be accessible because protected is available only in the
class hierarchy. Means only the one that extended from base can have an access to protected
members. MyClass is not extending base. So, base b cannot access the protected member. Only public
is accessible. Using base b, we cannot access private, protected or public members of the der class
because base class has no knowledge about what has been extended from it. Using d, we try to access
these members. private is inaccessible. protected would also been inaccessible whereas public is
accessible. So using der d, we can go to the public members of the base class. private and protected
are inaccessible. Only public is accessible.

Object Sizes

Suppose we have a base class base1 having

```java
private int i;
protected int j;
public int k;
```

and we have a deriv class which extends the base1 class which also has some

```java
private int x;
protected int y;
public int z;
```

Now we say base1 b = new base1( ). Likewise we would create an object of the derived class deriv d
= new deriv( ). When we create these objects, b will point to the base class object. In the base class
there would be i, j and k. In the deriv class object which is being referenced by d, there would be i, j, k
and x, y, z. i is a part of deriv class object. So when we create derived class object, it has base class
data so also the derived class data. i is inaccessible to derived class since it is private but it is a part of
derived class. We may have final within one of these classes. final is going to be shared by default.
static if present within the classes then it is not going to become part of the object.

Inheritance in real life

Suppose as shown in the slide, we have a Grandfather, Father and a child. Grandfather has some
properties i.e. He is a simple minded person, good English orator and very religious. When he gives a
birth to son, who is now Father is simple minded person means he has inherited the property of the
Grandfather, good English orator. So that property is also inherited from Grandfather. However, he is
moderately religious and he has developed good English writing skills. Means some of the properties
he has inherited from the Grandfather as it is, some he has managed to change whereas he has imbibe
new property within himself. Means some properties he has inherited, some he has changed and some
he has acquired newly. The child is also simple minded. This property he has inherited from Father
who inturn inherited it from Grandfather. Child is a good English as well as good Hindi orator. He
has scorn for religion and he is also a good sportsman. One property he has acquired as it is. 2nd
property he has manipulated a bit which is a combination of old and new. He has changed one
property completely and he has acquired something new which neither the Father nor the Grandfather
had. This inheritance permits use to do. Inheritance says that accept some properties as they are. If we
need, we can also change one of the properties. We can add new properties.
In this lecture you will understand:

* What does Inheritance offer us
* Practical application of Inheritance
* How objects are constructed in Inheritance chain
* How to prevent Inheritance
Calls…

Create a class a inside which we have function f1( ) and inside f1( ) we print the message a.f1. In addition to f1( ) within this class we have function f2( ) which prints the message a.f2. From class a, we try to derive a class b by using class b extends a. Inside class b, we have a function f2( ) which prints message b.f2. We also have a function f3( ) which displays the message b.f3. Then we have yet another function f4( ) inside which we print the message b.f4. Within class Sample inside main( ), we create an object of one of these classes. We say b z = new b( ). z becomes the reference of the class b and new b( ) creates the object of class b. Using z, we would try to call different functions by saying z.f1( ), z.f2( ), z.f3( ) and z.f4( ). When we say z.f1( ), f1( ) would be searched 1st in the derived class b and since in b there is no f1( ), control would go to base class b. f1( ) is searched in b and f1( ) is indeed present there and f1( ) would get executed printing a.f1. When we do z.f2( ), f2( ) is searched in the derived class b. Within derived class f2( ) is indeed present hence b.f2 would get printed out. When we do z.f3( ), once again f3( ) is present in the derived class so b.f3 would get printed out. Then we want that z.f4( ) would 1st print a.f2 and then print b.f4. To achieve that, when the control lands to f4( ), within f4( ), we should call f2( ) of the base class b. To be able to call f2( ) of the base class we say super.f2( ). super means the base class. Base class means super class and derived class means subclass. When we do super.f2( ), base class’s f2( ) would be called and when it is called it prints a.f2 and once the control comes back from base class’s f2( ), it prints the message b.f4 on the screen. If in place of super.f2( ), we say only f2( ) then that time derived class’s f2( ) would be called. The different cases that we have tackled here tell us that inheritance facilitates us to implement variety of cases such as use existing once, override the existing functionalities, create new functionality and we can make a combination of functionalities i.e. use some features of the existing functionality and add few more features that we want. These things inheritance offers us.

A Practical Scenario

Suppose a company has created a software which has several classes within it. Then after the product is released in the market, after some time passes and product meet some success then the company does a market survey before trying to create a new version and during that market survey some of the observations come forward as follows.

- A new Auto Update feature should be added to the product.
- Bug was created while saving files whose size is greater than 400 KB.
- Whatever data encryption function is present in the product, it is working very slowly.
- While opening a file, whatever is done right now in addition to that a progress bar should be displayed.
- Rest of the functionality of the product are seems to be at the acceptable level.

A completely new autoupdate( ) feature need to be provided in the product. There need change in the save( ) and encrypt( ) function. We want to provide a combination of existing features of open( ) plus the new feature that we need to introduce in it is displaying the progress of file opening through the progress bar control. Means a combination of existing and new feature is done here. Finally the existing functionality for example, function fun( ) which is existing right now there, we have to use as it is. We do not intend to make any change in fun( ).

Implementation

Suppose we have a class s which already has a function fun( ) and then there is a function save( ) following that there is a function enc( ) which carries out encryption and then there is a function open( ) which let us open a file. Now we create a new class news. So class news extends s means news is derived from class s. Inside the news class, we provide a save( ) function in which we try to
fix the bug which was present in the previous version of save( ). We also going to provide a faster 
eenc( ) function because previous enc( ) function in the class was found to be slow. Then a completely 
nnew autoupdate( ) feature, we wish to provide within this class whereas for open( ), we also add the 
logic for progress bar apart from opening the file. Opening of the file is carried out with base class 
open( ). New feature of progress bar, we provide in the current version of open( ). Within main( ), we 
have news x = new news( ). Once we have reference, we can make calls like x.fun( ) ; x.save( ) ;
xx.enc( ) ; x.autoupdate( ) ; x.open( ). When we do x.fun( ), the control goes to derived class to search
the function fun( ). Since it is not found there, it goes to base class to satisfy the request. So x.fun( )
takes to the base class’s implementation of fun( ). When we say x.save( ), once again save( ) is
searched in the news class and new version of save( ) is present there. So it is called. Same for enc( )
function. The newer version of enc( ) would be called. The earlier slower version would get
overwritten now. When we do x.autoupdate( ), the new feature that has been provided, will now get
invoked. When we do x.open( ), we will reach news class open( ) and from there we go to the base
class implementation of open( ).

**Ctor Calls**

We have a class a inside which there is a constructor a( ) in which we are displaying message a's 0-
arg Ctor. Likewise we also have a 1-argument constructor of the class a and inside which we are
displaying a message a's 1-arg Ctor. Then we create a class b extended from class a. Inside class b,
we provide a 0-argument constructor displaying the message b's 0-arg Ctor. Then we have a class
Myclass within which we have main( ) and inside main( ), we try to construct object of the derived
class by saying b y = new b( ). We are creating a new derived class object; its reference is stored in y.
Then we say b z = new b ( 10 ). When we say new b( ), we are trying to construct an object using 0-
argument constructor whereas when we do new b ( 10 ), we are proposing to construct the object
using the 1-argument constructor. When we execute this, in the output we get a’s 0-arg Ctor, b’s 0-
arg Ctor, a’s 1-arg Ctor, b’s 1-arg Ctor. Means when we try to create derived class object even though
we did not mention explicitly so the base class constructor also got called. Base class’s 0-argument
constructor got called so a’s 0-arg Ctor message was printed out. Following that b’s 0-arg Ctor was
printed out. Means construction proceeded from base towards derived. Firstly the base class
constructor was done then the derived class constructor was executed. When we create a 2nd object
which is being referenced to by z, that time also we expect a’s 1-arg Ctor and b’s 1-arg Ctor get
printed out. For that we have to provide a 1-argument constructor in class b. So we write a
constructor b( ) in class b which receives int x. Within this we say super ( x ) and following that we
print message b’s 1-arg Ctor. When we do super ( x ), the base class’s 1-argument constructor would
get called. Construction always proceeds from base class towards derived class. Call to super( ) must
be the very first statement in the 1-argument constructor. If super( ) is absent, then a’s 0-argument
constructor would get called. super( ) can also be used to call the base class functions other than the
constructor.

**Why Base To Derived**

Suppose we have a class base which has a protected int i. Then we have a constructor for the base
class inside which we are initializing i = 4. Then we have a der class which extends the base class and
in the der class we have int j and a der class constructor. Within the der class constructor we say j = i
* 4. After that suppose within main( ) we say, der d = new der( ). When we do so, the control lands in
the der class constructor where we say j = i * 4. i is available to der class because it is defined as
protected in the base class. Since construction proceeds from base towards derived, i get initialized
with value 4 and it is available to der class constructor. Hence, we get value of j as 16. If construction
does not happen from base towards derived then i is not get initialized with value 4, it is 0 and we get
j = 0. So because of this construction always proceeds from base class towards derived class.

**Conflict...**
Suppose we have a base class in which we have a protected int `i = 13`. Then a derived class extending the base class which also has `i = 9`. Within a `fun()` function, if we try to print out `i` and `super.i`, we will get the output as `9` and `13`. It tells that if there is any conflict between protected `int i` which is available in the derived class with some local variable in the class then the conflict can be resolved by saying `super.i`. `super` is not only work for a function or a constructor but also it work for a data. So, all types of conflicts between base and derived can be resolved using a `super` keyword.

**Prevent Inheritance**

We may create a class `shape` and within it we define a function `final void draw()`. Inside the `circle` class which is extended from `shape` we provide a `void draw()`. When we attempt to do so, the compiler would report an error. It would report an error because in the base class we are defining `draw()` to be a `final` function. Once we define the function as `final`, we cannot override that function in the derived class. Instead of defining the function as `final`, we may define the class itself as `final`. If we do final class `shape` and within it provide a `draw()` function and then try to create a `circle` class which is extended from `shape` and inside `circle` class we again provide a `draw()` function. When we do so, the `circle` class itself gives an error saying that once we declare the class to be `final`, there is no way that we can inherit from that class. Means if the class is declared as `final`, it cannot be inherited i.e. it is the final implementation; no further derivation is allowed. This prevents overriding of individual function or prevents inheritance of the entire class. `final` function cannot be overridden and final class can never be inherited.

**Inheritance Summary**

- Base class is also known as Super class or Parent class.
- Derived class is also known as Sub class or Child class.
- Derived class object contains all base class data.
- Derived class object may not be able to access all base class data.
- In inheritance chain, construction happens from Base towards Derived.
- Inheritance facilitates – Inheriting existing features, Suppress existing features and Extending existing features.
- Inherit - Do nothing
- Suppress - Hide base class implementation
- Extend - `super.Baseclassmethod()`
- `final` - Prevent inheritance
Exception Handling - I

In this lecture you will understand:

* When things may go wrong
* What can go wrong during execution
* Types of Exceptions
* Ways to tackle Exceptional Condition
* Difference between Checked & Unchecked
* How to tackle Checked Exceptions
Wishful Thinking

Commonly a programmer thinks that everything in his program is ultimately going to work out properly. Some also go to the extent of believing that "I am a good programmer" then what can go wrong with the program? "I have been programming for years hence I am anyway having enough skills for programming. So nothing can go wrong with my program". This is a false confidence. It’s always good to do defensive programming.

When Things May Go Wrong

When we do any program development, things may go wrong at every single stage of the development process. We first type the program and during typing itself lots of mistakes are done. Then we rely on the compiler to check out the grammar of the program. During linking of the libraries with the program, something also may go wrong and the third possibility is when we actually start executing the program, at that stage something would go wrong. During compilation, if something goes wrong then the compiler will assist us. Compiler tells that these may errors exists from these many lines and at the particular places in those lines. Means it can pin point to us that where we had gone wrong. We can rectify those errors at the compilation stage because unless and until the program is compiled correctly linking and execution will not begin. If during linking of the libraries with the program, something goes wrong then it will be reported by the linker and we can take care of this by writing proper imports within the program. If we do proper imports then in that case linking errors can be taken care of and during execution if things go wrong Java Runtime environment will report that something has gone wrong and we got to tackle that situation on the fly dynamically during execution of the program itself. Examples of things which may go wrong during execution are if we might be trying to allocate some memory for an array and the memory requirement of the array is such that, that much amount of memory is not available in that case a out of memory exception would be reported. The Java Runtime environment would report this exception. During execution of some arithmetic statements it might so happen that the denominator has got the value 0. If it has a value 0, the expression cannot get evaluated. In that case some exception condition will occur and which occur at execution time or runtime. We might try to exceed the bounds of the array. Suppose we had decided to allocate space for an array by receiving the number of elements from the keyboard. We receive the value of dimension of the array in variable n and in code we have written statement to allocate space for n integers for the array and after that we may run a loop from 1 to 10 to process different elements present in the array. If n is supplied as 5 and we are trying to process the array 10 times then it is going to report as an error at execution time. The further examples are Stack overflow may occur in the program, arithmetic over flow or under flow, attempt to use an unassigned reference, file may not found and unable to connect to Server. All these error conditions or exceptional conditions occur at execution stage. Hence we need to tackle them at execution time.

What Can Go Wrong During Execution

Suppose while executing a program an exceptional conditions occurs then this exceptional condition we can anticipate this occurrence and then try to recover from that exceptional condition. There might be some conditions which we can really not anticipate in which case recovery would also not be feasible. For example, if we are falling short of memory or if we are trying to open a file for reading which is not on the disk, in that case an exceptional condition can occur. But both of these which can be anticipated and since they can be anticipated recovery from them would be feasible. Those exceptional conditions that occur from which we cannot recover, they may occur due to internal conditions or due to external conditions. For example, if we are trying to pass a null instead of a file name to a function which is attempting to open a file. From this we cannot recover. Hence, we can say that this has occurred due to the internal condition, internal behavior of the program. If we are trying to read the contents of the file present on the disk and as the reading was proceeding may
be the disk for some reason fail. We got some cyclical redundancy check error. Means disk has gone bad then reading would never be feasible. From this exceptional condition a recovery would be impossible. This is because of an exceptional condition that occurred which was beyond our imaginational control. First category of exceptions is known as checked exceptions whereas the second category can be divided into runtime exceptions and error exceptions. The runtime exceptions and error exceptions are together often known as unchecked exceptions. Checked exceptions are those whose occurrence can be anticipated and hence can be checked at the compilation stage whether that have been anticipated or not whereas unchecked exceptions are those which cannot be determined at compilation stage hence those are unchecked exceptions.

**Tackling Exceptional Condition**

Exceptional conditions would occur when some method within some object is going to get executed. Whenever these exceptional conditions occur the Java model of tackling this is the Object Oriented model. Information about the exceptional condition as much as can be extracted that information is packed into an object. Once this information is packed into an object, this object is thrown by the Java Runtime Environment. If we want, we can create such an object, we can also throw it or we can it to the Java Runtime Environment to gather this information into an object and then try to throw that object. When the object is thrown, catch that object and react to that exception. This is the Object Oriented way of dealing with the exceptional conditions. Java provides a class hierarchy to represent these exceptional conditions. At the top of this hierarchy, there is a class Object. From the Object class Throwable class is derived and from the Throwable class two classes are derived; one is Error and another is Exception. From both these classes several more classes are derived. From the Exception class, a special class RuntimeException is derived. So Error, Exception and RuntimeException from these, there are several other classes are derived. Exceptional class’s to take care of the checked conditions whereas the external unchecked exceptional conditions can be tackled using the Error class whereas the internal unchecked exceptional conditions can be tackled using the RuntimeException class.

**Tackling Checked Exceptions**

When a method is called from the client code and that method is getting executed at that time something may go wrong and an exceptional condition may occur. When this exceptional condition occurs, information about that exceptional condition would be packed into an object. When this information is packed into an object, we can either do it ourselves or we can let Java Runtime Environment to do it for us. Once this information is packed into an object then that object is thrown. Once the object is thrown, we have several choices. One is we may just ignore the object that has been thrown. Means, we will do nothing about the object that has been thrown or we can catch the object that has been thrown in the client code. If we do nothing then somebody will have to pitch for us and somebody will have to do something about the object that has been thrown. We have decided to do nothing about it. In such a case, the default exception handler that Java provides that is the one that catch the object and when default exception handler catches the object, it prints out the Stack Trace and then terminates the execution. Stack Trace means the different functions that have been called i.e. from main(), we call fun1(), from fun1() we call fun2(), from fun2() we call fun3() then in fun3() an exceptional condition occurs, an object is created and that is thrown. Since we had done nothing about this hence the default exception handler will catch this object and then it will print out the Stack Trace. This helps us in debugging the program. If we decide to catch the object that has been thrown either by us or by Java Runtime then once the object is caught, we have two choices; one is we can rectify the situation and then continue the execution and another is a graceful exit.

**Case 1: Do Nothing**

Begin with import java.io.*. In class Case1 inside main() we say int i. Then we wish to read the value in the variable i. We read it using a BufferedReader and InputStreamReader. In i the integer that
has been supplied is stored. This appears to be a perfectly all right program. But it is possible that when this program is executed an exception may occur. It will happen when even though we ask the user to enter a number and we supply a string abc. In that case the integer i cannot hold the string abc. So the exceptional condition would occur and since we had done nothing to anticipate this condition hence the Java Runtime Environment will have to obtain the information about the exceptional condition packaging to an object and throw it. Since the object that has been thrown, we have not caught hold off, the default exceptional handler will catch it and then it will report an error saying that the input string that we supplied was abc, this cannot get stored in integer i. So NumberFormatException has occurred. NumberFormatException is a class which is derived from the Exception class. When we see the error message that has been reported by the default exception handler, it indicates that an exception has occurred while inputting a string and it is also reporting the name of the class. The Sample project contain Case1 as the class inside which we have main( ) and in main( ) that error occurred and it is also reporting that the error occurred in line number 24. In main( ), we are saying throws Exception because while carrying out the reading using BufferedReader and InputStreamReader and when we do b.readLine( ) that time the readLine( ) method has declared that some exception may occur and since it has declared that it anticipates some exception to occur while reading a line hence the method which tries to call the readLine( ) method will now be forced to catch that exception or if not catch then at least throw it back to the runtime. So if during reading some exception occurs that error object would be thrown by the Java Runtime Environment or by the readLine( ) function and we had decided to throw it back to the runtime by using throws Exception. Printing out the Stack Trace is a clumsy way of doing it and also a bad practice of doing nothing about the exception that has occurred.

Case 2: Recover & Cont.

We import java.io.*. Then within main( ) of class Case2, we try to receive the integer in variable i. Then we say while ( true ) rather than straight way saying b.readLine( ). Inside the infinite while, we would write a try block which is a keyword and always followed by a pair of braces. Inside the try block, we anticipate something to go wrong. Whatever or wherever we anticipate to things go wrong, those we should put within the try block. So receive the integer, parse that string into the integer, store in i and then break outside the infinite loop. If to the readLine( ) a string like "abc" is passed then it would result into an exceptional condition and when exceptional condition occurs break will not go to work and the exception is thrown. If that exception is thrown then we can either throw it back from main( ) to Runtime such that the default exception handler can tackle it or we can ourselves catch it rather than leaving it for the default exceptional handler. We catch it by writing a catch block. Within the parenthesis of the catch block, we are to collect the reference e and address of the exception object that was thrown is assigned to e and since the exception object that was thrown is NumberFormatException hence we have caught it in NumberFormatException e. After that we can display the message that Incorrect Input. With the infinite while loop, we have ensured that no matter how many times a NumberFormatException occurs, we will still give the user another chance to supply the correct number. When we supply the number then only the break within the try block would get executed. Outside the while loop, we print the number entered. We should try to anticipate what is likely to go wrong in the program and then put those statements within the try block. Immediately after the try block, we have to have the catch block which will catch the exception that may occur. Try catch is a policy which says hope for the best but prepare for the worst.

Case 3: Graceful Exit

Inside main( ) in class Case3, we receive the integer. After receiving the number within the try block, we would parse it and print the number entered. If an exception occurs while parsing then a NumberFormatException object will be thrown. We will catch that object and once we catch that object, we will report that something had gone wrong and since we do not want to recover it, we perform a graceful exit. We display the message that Incorrect Input.
In this lecture you will understand:

* A quick recap...
* How to catch multiple exceptions
* How finally block works
* What is the purpose of the finally block
* How to tackle unchecked exceptions
* Nested try statements
Recap…

- Exceptions are treated as objects.
- Exceptions might be Checked or Unchecked Exceptions.
- When a checked exception occurs then we have three choices - Do nothing, Rectify and continue, Graceful exit.
- We have to enclose code where we anticipate error in try block.
- Catch the thrown exception in catch block.
- catch block must immediately follow the try block.
- When exception is thrown, control goes to catch block.
- After catch block is executed, control goes to the next line that follows after catch block(s) unless there is a return or throw in the catch block.
- On calling a method that has advertised that it will throw an exception, we have to either catch the exception that has been thrown by the method or we have to rethrow that method for somebody else to catch it.

Catching Multiple Exceptions

We begin with import java.io.* and import java.lang.*. In a class MultipleExceptions inside main( ), we have two integers i and j. Then we try to read values of these integers from the keyboard but it is done inside the try block because we do anticipate some error to occur while converting a string to an integer. Likewise receive another integer’s value, parse that and then we print the values of i and j. Then we would also try to print the result of i / j.

Slide 5

Here is a possibility of occurring three exceptions. One is while converting Integer.parseInt( ) and attempting to store the result in i, that time the IllegalFormatException may occur or while converting the string into an integer while assigning it to j that time an exception may occur. Third possibility is when we try to do i / j, that time also an exception may occur because j might turn out to be 0. If denominator is 0 then division by 0 is impossible so an exceptional condition may also occur that time. So while receiving i and j, two exceptions occur and while performing i / j third exception occur. We wish to tackle every one of these possibilities ourselves. So beyond the try block we say catch ( NumberFormatException ne ) and inside it we display a message that Incorrect Input. So if value supplied to i is improper then also the same exception occurs and if while receiving the value of j, an exception occurs then that will be received by the same block. Then we also have catch ( ArithmeticException ae ) this occurs when we do i / j and j is 0. In that case an ArithmeticException object is thrown, collect that object in ae and display message Arithmetic Exc. / by zero. Lastly we say catch ( Exception e ) where we display Unknown Error and the value present in e. So we have three catch blocks for one try block. catch ( Exception e ) block is written because anything other than we are anticipating if that exception occurs in that case it will be caught hold up by the last catch block. If multiple catch blocks are present for one try block that is perfectly all right. At a time only one catch block will go to work. When exception occurs these catch blocks will be matched one after the other. Since they are matched one after the other, the order in which we write them is extremely important i.e. we should write catch blocks from derived to base. Both NumberFormatException and ArithmeticException classes are derived from the Exception class. So if we write catch ( Exception e ) as the 1st catch block then every single exception will be catch by this block. Hence we should always write the most derived exception class’s catch block as the 1st catch block and the most base exception class’s catch block as the last catch block.
finally Block

Inside class Case5 within main( ), we say FileWriter fw = null. FileWriter is a readymade class and we are trying to declare the reference of that class and trying to set its value to null to begin with. Within the try block we say fw = new FileWriter ("a.txt"). FileWriter class allows us to write into the file. We invoke the constructor of the FileWriter class and name of the file "a.txt" is passed to that. File will get created in the current directory. Using the reference fw, we try to write some message to the file i.e. Hello World. Then we say catch ( IOException ie ). While working with the FileWriter class, if something goes wrong in that case an IOException would be thrown; that IOException’s object address is catch in reference ie. We display the message Encountered IO Error. Exception is occurred when we try to write something into the file or exception may occur when we try to create this file. No matter in what case the exception occurs, we want that finally some piece of code must get executed whether file writing was successful or file writing was unsuccessful. That piece of code we write in the finally block. finally is a keyword in Java specially dealing with exceptions.

Slide 7

In the finally block, we should try to close the file. If the file opening itself has failed then question of closing it does not arise. So we should write finally block taking this into consideration. We should check if ( fw != null ) means fw is pointing to FileWriter object i.e. the file was successfully opened. So in that case we try to close the file. But we would try to put this within the try block because something may go wrong within the closing of the file. So closing is done within the try block and then catch ( IOException ex ) and display the message that cannot close the file. No matter whether statements within try block got executed or while executing them an exception occurred and then statements within catch block got executed, irrespective of both these situations code in the finally block is always going to get executed. So we usually write the clean up code in the finally block. Even if a return or break occurs even then the finally block is going to get executed. Once returned, it’s bound to get executed every single time. Only exception to this situation is if we ourselves do System.exit( ) then when exit is done, execution cannot proceed. finally block is written immediately after the catch block or multiple catch blocks. try block must have catch block and/or finally block. finally clause is optional.

Tackling Unchecked Exceptions

Unchecked exceptions are those which are not checked at compilation time. These are never advertised by methods. If a method anticipates that something may go wrong with the execution of code within it then it has a choice of advertising that these are the exceptions that we expect to occur while executing the code. If it advertises those exceptions in that case we have no other choice but to either catch them or throw them back. So every such method which has advertises the exception will have to be either caught or tackled in the client code. As against this, some of the unchecked exceptions are never advertised because they are too many in number. Two types of these exceptions are represented by Error class or its subclasses or Runtime class or its subclasses. Examples of Error class or its subclasses are Internal error or Virtual Machine error. Examples of Runtime class or its subclasses are division by 0 or improper array index. External errors that occur, we do not catch them because we do not know what to do even if we catch them whereas Runtime exceptions are indeed caught. Runtime exceptions represent things within a program which really should not happen but we have caught them such that during testing itself we can fix these errors. While doing this task in C or C++, Assert macro is used to deal with such situations. We should use checked exceptions for those exceptional conditions from which we think we can recover whereas Runtime exceptions we should use for common programming errors.

Nested try Statements
• A try statement can be inside the block of another try statement is acceptable. Means nesting of try statements is perfectly legal.

• Each time control enters a try block, the context of that exception is pushed on the stack.

• If inner try block does not have a catch, then the outer try block's catch handlers are inspected for a match.

• If method is called from a try block and that method in turn has a try block within it, then it is also treated as a nested try block.
User Defined Exceptions

In this lecture you will understand:

* What are user-defined exceptions
* When do we need to create them
* How to create them – Examples
* Exception etiquettes
User-defined Exceptions

Suppose we have a Customer class which has name of the account holder, the account number and the balance against the particular account. It also has two methods within it; one is withdraw( ) and another is getbalance( ). withdraw( ) allows to withdraw the amount from the account whereas getbalance( ) gives how much balance is standing against a particular account. While withdrawing a amount the balance in a particular account should not go below a certain limit. If that balance goes below a certain limit then it would raise an exception. This particular issue of amount or balance against a particular account going below a particular limit is a standard exception cannot handle. So we will have to create our own exception class so we are giving a name myexcep. Within the myexcep class we have a constructor and a message( ) method. myexcep class is derived from a standard Java Exception class. In the Client program, when we try to use the Customer class, we will say that while withdrawing if balance goes below a particular limit; for example, below Rs. 500 in that case we would try to throw an exception. We would try to create an object of the myexcep class and then we would try to throw that particular object. Wherever we call the withdraw( ) method, there we will try to catch the exception in the Client code.

Customer Class

Create a class Customer and within the class we have

```java
private String name;
private int accno;
private int balance;
```

Then we have a constructor of the Customer class which receives a String for name, a for account number and b for balance. The constructor assigns these values into the three private variables of the class. Beyond this we have a withdraw( ) method which receives amount as one of the argument that the user intends to withdraw. This method does not return anything so return type is void. Then we say throws BankException indicates that we propose to throw a BankException object out of this method and we should do so whenever an exceptional condition arises. So within the withdraw( ) method, we must first check whether such an exceptional condition has occurred or not using if ( balance – amt <= 500 ). If it goes below 500 then we throw the exception. While throwing an exception, we have to construct an object of the BankException class. Once this object is constructed, we will try to throw it from this method. While constructing the BankException object, we have to pass the accno and also the balance standing against the account. This exception is thrown only if the exceptional condition arises otherwise the amount get withdrawn using balance -= amt. throw and throws are keywords. throws indicates that the withdraw( ) method would throw an exception if it is necessary to do so otherwise it would not throw any exception at all. Means if an exceptional condition arises then only there is need to throw the exception whereas if such a condition does not arise then carry out the normal processing. throw indicates that a need has arisen to throw an exception so throw it now. So throw is definitely going to throw an exception whereas throws means that a exception might be thrown or it might not be thrown.

Client Class

Within main( ) of the Client class, we create a new Customer object. We provide the name, account number and the balance against that particular account. Then we try to do c.withdraw ( 450 ). When we try to do so, the withdraw( ) method would be called and since the balance that is standing against this account after having withdrawn Rs. 450, it will fall below 500. Since its going to fall below 500, an exception will be thrown. When this exception is thrown, main( ) has to either catch this exception or throw it further. If main( ) decides not to catch the exception then in that case it has to throw the exception because it has been very clearly indicated in the withdraw( )’s method definition that withdraw( ) is going to throw a BankException if needed. So whenever we call a method which is
going to throw an exception either we got to catch that exception on ourselves or we got to throw it further. When we throw an exception, we are throwing a BankException object and the BankException class is derived from the Exception class. In this case we do propose to call that exception rather than throwing it to the Java Runtime. After withdrawing the amount, if the balance is below a particular limit then we got to catch the BankException that is thrown. When the control reaches the catch block, we should print out the message that Transaction failed. We can also call a method inform( ) which is a method of the BankException class and within this method we should try to inform as to the account number for which the transaction failed and the balance standing against that account.

**BankException Class**

BankException class is derived from the Exception class. Inside this class we have two private integers acc and bal. We need these because anytime the BankException object is constructed we pass this account number and balance to it. We also have a inform( ) method inside it. In the inform( ) method, we are going to display the account number as well as the balance where the transaction failure had occurred. From the client when we try to call a method which we anticipate will throw an exception then either we got to catch that exception or throw it further.

**One More Example**

We wish to maintain a Stack but in this stack we wish to store names of the persons as well as integers and floats. Means this is going to be a stack of objects. In this case Vinod and Sanjay are objects but 35 and 3.14 are primitives. So while dealing with integer like 35 rather than treating it like a primitive, we will have to treat it as an object. Storing a primitive value in an object is known as boxing. So we will have to box 35 into an integer object. Likewise we will also have to wrap 3.14 into a float object. Then the boxed objects along with the string objects we can push on the stack. To maintain the stack, we have three private variables; one is an array data[ ] into which we have to actually dump all these different objects and two more private integers capacity and size. capacity indicates how many elements can really push on the stack whereas size indicates how many elements have so far pushed on the stack. Then we have push( ) and pop( ) functions and also the getSize( ) function which will return us information about how many elements are there on the stack right now. As we push and pop the elements, it might happen that the stack becomes full or the stack becomes empty in which case an exception will be thrown. We wish to tackle that exception. Within Client we are going to create Stack object and then we are going to call push( ) and pop( ) several times to push elements to the stack or to pop elements of the stack. In the StackException class, we have a one 0-argument and one 1-argument constructor. We have to derive the StackException class from the Exception class.

**StackTest Class**

We will begin with the client class from where we are going to make use of the Stack class. StackTest is the name of the class. Within this class we have main( ) and inside main( ), we create a new Stack object. We pass a value 3 to the constructor of the Stack class indicating that we are going to construct an object which is having the capacity of 3 elements on the stack. Within the try block, we would try to push elements on the stack. First we push string Vinod then a string Sanjay and then we push an Integer object and to construct that Integer object we say new Integer ( 25 ). Likewise we try to store a Float object using new Float ( 3.14f ). The capacity is 3 whereas we are trying to push 4 elements. So the moment we attempt to push the 4th element on the stack, capacity will be found less than what we push on the stack. In such a case we expect the Stack class to throw an exception. When it throws that exception, we would want to catch them. So in the catch block we would catch the StackException and then whatever is present in x is printed out. Now we would have another try block where we would try to call the getSize( ) function repeatedly. So we say while ( s.getSize( ) > 0
Inside it we call a `pop()` function and whatever value `pop()` returns we will print that value. While popping if an exception occurs then we would catch that also. Whatever is present in `x` is printed out.

### Stack Class

Within the Stack class we have two private integers `capacity` and `size` and we have an array of objects `data`. Into this array of objects we are going to store string objects, integer objects, float objects, etc. In the constructor, we receive the cap and set it to the capacity value. But 1st construct an Object array of that capacity. We set `capacity = cap` whereas `size = 0` indicating that when the Stack object is created, even though the capacity is 3 right now there is nothing on the Stack. So set `size = 0`. In the push operation whichever object is pushed on the Stack that will be received as an argument for the `push()` function. `throws StackException` indicates that if the Stack really becomes full then the `push()` function is going to throw an exception. `throws` indicates the intention of throwing an exception. If `size == capacity` means we cannot push any more elements on the stack in that case we would throw an exception called `StackException`. We are invoking its constructor and passing a string indicating that Stack has become full. Otherwise we should push the element on the stack by using `data[ size++ ] = o`.

#### Slide 10

Now we define the `pop()` function. In the `pop()` function, there is also the possibility of an exception may occur. So, we say throws `StackException`. If the Stack becomes empty and we still try to pop something from the Stack then at that time we have to throw an exception. So if `size <= 0` then we should throw a new `StackException`. This time indicating that the stack is empty. Otherwise we should pop the object by saying `data[ --size ]`. So the return type of the `pop()` function is the Object that we have popped off i.e. `Object`. The `getSize()` function returns the current value of the size such that we can come to know that how many elements are there on the Stack right now.

### StackException Class

The `StackException` class is extended from the `Exception` class. We have two constructors; one is 0-argument constructor which does nothing and a 1-argument constructor which receives a message and calls the base class constructor by using `super( msg )`. `super` is a keyword which calls base class constructor which is nothing but a 1-argument constructor of the `Exception` class.

### Exception Etiquette

- Create a public static variable in class and anytime an exceptional condition arises we try to set different values in this global variable which is a public static variable of some class.
- Force the method to return a boolean value indicating the success or failure of that function. Also return `int` or a string or an object to indicate the success or failure of that function.
- Catch the exception and if we do not catch the exception, an error occurs. We write a catch block and completely ignore the exception which is a bad way.
- Do not use exception handling only for cosmetic purpose.
- Do not catch everything using `Exception`.
- Always try to distinguish between types of exceptions.
- Make it optimally elaborate – not too much, not too little.
Abstract Classes

In this lecture you will understand:

* What is dynamic dispatch mechanism
* What are abstract classes
* How to construct abstract classes and methods
* Example programs to demonstrate abstract Classes
* What can be / cannot be done using abstract classes
Dynamic Dispatch Mechanism

Suppose we have a class Shape inside which we have a Draw( ) function which has a body but we do not written any code in it. Then we create a class Circle which is derived from the Shape class. Inside the Circle class also we have a Draw( ) method which displays the message as Circle. We also create a Rect class which is also derived from the Shape class. Rect class also contains a Draw( ) function which displays the message as Rectangle. Within Myclass inside main( ) we create several Circle objects. For that we declare three references c1, c2 and c3 of the type Circle. Then construct the Circle objects, store their references into c1, c2 and c3 using c1 = new Circle( ) and so on. Then we also create several Rect objects whose references we propose to store in r1, r2 and r3. So we do r1 = new Rect( ) and so on. We create a Shape array s[ ] which is initialized to values like c1, r2, etc. using Shape s[ ] = { c1, r2, ... }. Means all the objects that we have created, their references we would try to store in array s. We would run a loop 6 times and each time through the loop say s[ i ].Draw( ). In the Shape array s, we are not storing the address of the Shape objects. In a shape array of references, we are trying to store addresses of Circle objects and Rect objects. Once this is done, it’s known as upcasting because we are trying to store the address of the derived class object in a reference to the base class. Shape s[ ] indicates that s is an array of Shape references. Means into the Shape array of references we are going to store references of Circle and Rect class. Using the upcasted references we are trying to call the Draw( ) method. Whenever we do so, a suitable Draw( ) method would get called i.e. either Draw( ) of the Circle class or Draw( ) of the Rect class. Which of these two gets called that depends upon which object’s reference is right now present in s[ i ]. Deciding which Draw( ) method would be called is decided at execution stage. Since it is decided at execution stage this mechanism is known as Dynamic dispatch mechanism. We want to make sure that nobody is ever able to call the Draw( ) method of the Shape class. If we want to prevent that we must also say to it that we do not allow a Shape class’s object ever to get created. Means the Draw( ) method of Shape class should never get called. If that is to be prevented then we must see to it that we do not let anybody to create an object of the Shape class. If we want that Shape class’s object should never get created then Java provides a keyword abstract. If we define Shape as an abstract class then nobody is ever be able to create an object of the Shape class. Correspondingly we can define Draw( ) method as an abstract method indicating that we do not ever propose to call this method because its an abstract method. abstract method would not have body. So pair of braces are deleted instead there is only a ;. We expect that the class which extends the abstract Shape class actually provide the implementation of the Draw( ) method. So if we define the method as abstract in the base class, we provide the implementation of that method in the derived class. Since we are providing implementation of the abstract method in the derived class it is expected that we must give same name for the method.

Example I

Suppose we want to represent different printer using classes. Every printer has its own way to print. A Epson printer may carry out printing in some way, a Laser Printer may carry out printing in some other way. The application that we wish to write should be able to print on absolutely any printer no matter which one of them is attached to which machine. Application need to know only the name of the printer to print. To do printing, application would merely call the print( ) method and then the suitable printer to print out the document that the user tries to print out. For that we will have to create a Printer class and we have to declare this Printer class as an abstract class. It is going to have the name of the printer. It can also have a constructor and a print( ) function. We will have to define print( ) as an abstract method. Then from the generic Printer class we derive a more specific Printer class. Since we declare print( ) method as abstract in the base class, its implementation we provide in the LaserPrinter class. We have one more generic class InkjetPrinter which also has a constructor and a print( ) method. So these are the derived classes derived from the abstract base class Printer.
Usage

Suppose the name of client is AbstractDemo. We import java.io.* and inside AbstractDemo class we have main(). Within main() we try to create new LaserPrinter object and we will pass the name of the Laser Printer to the constructor. After that we will try to call the print() method using this reference by using p.print(). We had done upcasting because LaserPrinter is a derived class whereas Printer p is a base class reference. We would create another object of the type InkjetPrinter and address of this object is also stored in p means this is also an upcasted reference. Now we call p.print() and we are going to print different document on this printer and name of document is hello2.doc. On the laser Printer we are printing a pdf file hello1.pdf.

Abstract Class

Name of the abstract class is Printer and within the Printer class we have a protected String name. It is declared as protected such that it becomes available to the derived class and then have a constructor public Printer ( String n ) and store n in name i.e. name = n. We have a abstract method called print() which receives the name of the document. Declaring Printer class as an abstract class, creating an object of Printer class is ruled out. We also defined print() as an abstract method indicating that whosoever deriving from this class, it will have to provide the implementation of this method. So print() method does not have any body. Following are the few tips about the abstract class.

- Object cannot be created from an abstract class.
- An abstract class can contain abstract and non-abstract functions.
- Abstract class may contain instance and static variables.
- These variables may be inherited.
- Abstract methods do not have a body.
- Abstract classes can participate in inheritance.
- If a class contains abstract methods class has to be abstract.

Derived Classes

The LaserPrinter class is derived from Printer class and in constructor we call base class constructor by saying super ( n ). Then we have an implementation of print() function which prints the name of the document that it receives on the screen. We have one more InkjetPrinter class which is also derived from the Printer class. In the constructor of this class also we call the base class constructor by saying super ( n ). Then a print() function which prints the name of the document that it receives.

Example II

While dealing with different image their formats are different. For example, a gif file constructed on a disk is different than a bmp file is constructed. Formats of different files are bound to be different. If file formats are different; when these files are loaded in memory, the images that are in memory they would also be different. We wish to design classes to load the images from the disk into memory and to save those different images back to the disk. It should be possible to extend these designs tomorrow to accommodate newer file formats. In that sense the classes should be extensible. TO do so we create a base class Image which contains name of the file and methods like Create( ), Load( ), Save( ) and setImage( ). From the Image class we derive two new classes GIFImage and JPEGImage. These two classes have a Load( ) function and a constructor. Save( ) function is not present in both these classes because once the image is constructed in memory when its time to save it on the disk, nothing is going to get different from one image to another. Whereas while loading images are
different. So they are implemented differently. Create( ) is a class factory function because its going to create objects.

Usage

In a AbstractImageDemo class within main( ), we would try to create an object by calling the class factory function. So we say Image.Create( "test.jpg" ). Upcasting is done here. Create( ) function is a class factory function which on the basis of extension of the file, we will decide whether it should create a JPEGImage object or a GIFImage object. So the extension passed is important. If its going to create a JPEGImage object, its reference is returned which is collected in i. Then we say i.Load( ). Based on which objects reference is present in i, that particular class’s Load( ) function would now be called. Same is done for the gif image. It will create the GIFImage object, return its reference, collect that in i. The class factory Create( ) function will have to look at the extension that we passed to it and the basis of that create the suitable object. So when we do i.Load( ) 2nd time, GIFImage class’s Load( ) function would get called. This is a dynamic dispatch because based on which object’s reference is in i, appropriate Load( ) function would be called. Which Load( ) function would be called that decision is taken dynamically during execution. So, known as a dynamic dispatch mechanism.

Abstract Class

In the Image class we have a protected String fileName such that its available to the derived class. Its also have a setImage( ) function which receives a string and stores it in fileName i.e. this.fileName = f. Then we have a class factory function Create( ) which will first check up what is the extension of the filename that it has received in String f. Before checking the extension we convert the string into lower case by using toLowerCase( ) function of the String class. It returns the reference of the lower case string. We use that reference to call method endsWith( ) of the string class. To the endsWith( ) function we pass .jpg. This method returns true / false depending on whether the lower case string that we are passing to it does it ends with .jpg or not. If it ends with .jpg then it will return true, otherwise it will return false. If it returns true then we return a new JPEGImage object and pass string f to the constructor of the JPEGImage object. Otherwise we check that the file name passed to the Create( ) function is .gif. If it is .gif then we return a new GIFImage object and pass string f to the constructor of the GIFImage object. If it’s anything other than jpg or gif then return null. So return type of Create( ) is Image.

Slide 11

Whenever its time to save a image the address of that image is passed into some char[ ] buf. It has to create a new file. For that it has to construct an object of the FileWriter class. Name of the file is available to this class which we will use to create a FileWriter object. Using the FileWriter object, whatever is present in buf is written into the file. Finally we close the file. Then we have a public abstract void Load( ) function which is a abstract method which derive class will implement.

Derived Classes

The JPEGImage class is extended from the Image class which has constructor which receives name of the file and passes it on the base class’s setImage( ) function. Then there would be a Load( ) function which is the implementation of the abstract Load( ) function in the base class. Inside this we print out the message here. Similarly the GIFImage class is extended from the Image class which also has the constructor. The constructor receives the file name and passes it to the base class’s setImage( ) function. Then the Load( ) function implementation is done which prints out the message.
In this lecture you will understand:

* What are interfaces
* How are they different than abstract classes
* In which different situations can we use interfaces
* Sample programs to demonstrate the situations
What Are Interfaces

Interfaces is a collection of declarations of different functions or methods. The syntax for writing an interface is similar to the syntax for writing a class except that instead of class keyword, we have to use interface keyword. The definitions of the functions will be done in those classes which have decided to implement that interface. For example, we declare an interface Mouse within which we have two methods declarations; one is lbtndn( ) and another is rbtndn( ). Each one of these methods receives a Point p where Point p represents a structure containing x and y co-ordinates. If we want to implement this interface in a class then we say class GeniusM implements Mouse and then we will provide the implementations of lbtndn( ) and rbtndn( ) methods. In both the methods we have printed out some message like Left Button and Right Button. The definition of functions is always be done in the class that implements the interface whereas interface will merely contain the function declaration. An interface just declares what a class must do. implements is a keyword. Multiple classes can also implement the Mouse interface. Different classes may implement the same interface in a different manner. A class can implement any number of interfaces. For example, if we have two interfaces i1 and i2 and class C implements that then we say class C implements i1, i2.

Interfaces V/s Abstract Classes

- Abstract class functions may have a body.
- Interface functions never have a body.
- Abstract class may contain instance variables.
  - These variables may be inherited by those classes which are derived from abstract class.
- Interface may contain variables.
  - These variables can be accessed but these variables almost act like constants.
  - These variables are by default static and final.

Interface Scenarios

There are three scenarios where interface can be used. The first way permits a focused view of a large implementation. For example, we have a implementation in a class which contains functions like Encrypt( ), Decrypt( ), Compress( ), Decompress( ), Authorize( ) and Authenticate( ). These are different functionalities but all of them stand implemented in a class. Each one of these methods we can declare in three different interfaces and then using the interface reference of one particular type we would be then allowed to call only functions that are declared within that particular interface. Means if we have an IEncrypt interface then using the IEncrypt interface reference we can call the Encrypt( ) and Decrypt( ) methods. Means the interface is giving the focused view, a small window. Through that window alone we can look at the class. Likewise there can be ICompress interface which allows to call only Compress( ) and Decompress( ) methods which gives another focused view of the same large implementation. Second possibility where we may decide to make use of interfaces is when we wish to provide different implementation of exactly same tasks. Suppose we have a class Array which has the ability to count how many elements are in the array and it also has the ability to insert new elements in between existing elements or add a new element at the end or remove some of the existing elements from the array. To carry out these operations there are methods like Count( ), Insert( ), Add( ) and Remove( ). In place of an Array we may have a Linked List which may decide to implement exactly the same methods like Count( ), Insert( ), Add( ) and Remove( ). These four methods we can declare in an interface and let the Array class implement that interface or the Linked List class implement that interface. Advantage of this is, once we know that the Array class implements a particular interface then we know that the methods we need to call to do counting,
addition, insertion or removing is Count( ), Insert( ), Add( ) and Remove( ). Once we get experienced with the usage of the Array class if we switch over to the Linked List class and same interface also stands implemented in the Linked List class then our ability to use Linked List get enhanced. Second possibility where we may decide to make use of interfaces is when we wish to inherit desired qualities from unrelated sources. For example, as shown in the slide, we want the looks of John Abraham but want the characters of Dr. Kalam. Character of Dr. Kalam could be the patriotism that he has displayed from time to time. Since these are unrelated people, we cannot think of inheriting one from the other. So in such case we can think of using interfaces. Some can say that they want only the characters of Dr. Kalam. So we can have an ICharacter interface and implement it in two different classes.

I. Different Implementation

We have a package diffimpl, import java.io.*. In class DiffImpl inside main( ) we say IListMethods i. Then we say i = new MyArray( ) where MyArray is a class which implements IListMethods interface. Then we say i.Add ( 1 ) ; i.Remove ( 1 ) ; i.Count( ). i is an interface reference whereas new MyArray( ) would create an MyArray object and return the reference of MyArray object. Similarly we can say i = new MyLL( ) and then we call i.Add ( 1 ) ; i.Remove ( 1 ) ; i.Count( ). MyLL would also implement the IListMethods interface.

Interface

We have interface IListMethods which contains public int Count( ) ; public void Add ( Object o ) ; and public void Remove ( Object o ). So when we pass 1 while adding that 1 get boxed into an Object.

Array Implementation

We declare a class MyArray which implements the IListMethods interface. Within this class we have to provide three methods Count( ), Add( ) and Remove( ). Within all the three methods, we are printing out some message and not doing any operation.

Linked List Implementation

We also declare the class MyLL which implements the interface IListMethods. This indicates that one interface can be implemented by multiple classes. We provide implementation of three methods Count( ), Add( ) and Remove( ) within this class. Inside these methods we are printing different set of messages. IListMethods are fully implemented in MyArray and MyLL classes because we declare three methods in IListMethods interface and we provide implementation of all three methods in both the classes. If we define only two methods within any of the classes then it would become partial implementation. In that case we would not be able to create an object of those classes.

II. Focused View

We have a class InterfaceDemo inside which within main( ) we create a new FocussedView object and its reference is collected in FocussedView o. Then we have an interface ie to which o is assigned i.e. IEncrypt ie = o and using which we call the Encrypt( ) and Decrypt( ) methods. Likewise we have ICompress ic which is another interface which has been implemented in the FocussedView class. Using ic we call Compress( ) and Decompress( ) methods. Then we have an IAuth interface reference ia to which we assign the same object’s address i.e. IAuth ia = o and then we call Login( ) and Logout( ) methods of the ia interface. If we do ia.Compress( ) then we get a compilation error because using ia we can call only those interface methods whose declarations have been done within the interface IAuth. Similarly using ia, we would not be able to call methods like Encrypt( ) or Decrypt( ).
Interfaces

We declare interface IEncrypt which contains public void Encrypt( ) and public void Decrypt( ). Likewise ICompress interface has methods declarations like public void Compress( ) and public void Decompress( ). Lastly the interface IAuth has methods declarations like public void Login( ) and public void Logout( ). All three interfaces are implemented in the FocussedView class.

Implementation

A class FocussedView implements IEncrypt, ICompress, IAuth. Within this class, we have implementation of the methods which we are calling i.e. Encrypt( ), Decrypt( ), Compress( ), Decompress( ), Login( ) and Logout( ). We are printing out message inside each one of the method such that when we call them from the output we can come to know which method is called. This indicates that within one class multiple interfaces can be implemented.

III. Qualities

Suppose we have a class Qualities inside which we have main( ) and within main( ) we say Model m which is a reference of the Model class. Then we say Actor myself and myself = new Actor( ). Means we create a Actor object and store its reference in myself. Then we say m = myself ; m.style( ). Then we say ICharacter ch which is a interface reference to that also we assign the address of the same Actor object i.e. ch = myself. Then we call a method patriotism( ) using it i.e. ch.patriotism( ).

Interface & Model Class

We have a ICharacter interface which has a method declaration called public void patriotism( ). Then we have a class Model which has a protected String hairstyle and a method public void style( ) inside which we print out a message.

Actor Class

We have a Actor class which is derived from the Model class and within the Actor class we implement the ICharacter interface. Inside Actor class, we provided methods doActing( ) and style( ) which displays the message. Within the style( ) method we call the super.style( ) indicates that we are proposing to call the style( ) method of Model class also and then we are printing the message. Then we had provided the implementation of the method patriotism( ) which comes from the ICharacter interface and displays a message. Means Model is the base class from which we had extended the Actor class and within the Actor class we had implemented the ICharacter interface.
Interfaces - II

In this lecture you will understand:

* Interface recap...
* Bow and Arrow - Class Design
* Bow and Arrow - Program
Recap…

- Interfaces are not derived from Object class.
- Interfaces are not derived from any base interface.
- Interfaces contain only method declarations.
- Interfaces may contain static and final data; it will not contain any instance variables.
- Objects cannot be created from interfaces.
- A class can implement multiple interfaces.
- Interfaces can be inherited using the same keyword extends.
- Multiple interface inheritance is allowed.
- Class cannot implement an interface partially.
- Interface can be used in three situations:
  - Focused view
  - Different implementation of same task
  - Alternative for multiple inheritance

Bow & Arrow

In this game we had a set of balloons which used to travel from bottom of the screen towards the top of the screen and they used to move at top at a fixed time interval. Then we have a guy called Robin who holds a bow and an arrow and the moment we hit the Enter key, from the bow an arrow used to get shot out and aim of Robin is to shoot down as many balloons as he can. So as the balloons move up, we cannot keep Robin as fixed. Robin has to reach at the height of the balloon and then shoot out the arrow. So Robin also can be moved up and down. We have to maintain different positions of Robin and different positions of different balloons. We can move Robin up and down whereas balloons can move only in the upward direction. In the actual game we have 10 different balloons. There are elementary and advanced levels wherein in the elementary level, all the balloons of 10 different colors is used to start at the bottom of the screen and used to together move at the top. In advanced level of the game, we have to start 10 different balloons at 10 different levels on the screen and then all of them are used to move together.

Classes

We can think of having a Shooter class, a Balloon class and a Timer class. Within the Shooter class we have methods like setstep( ) and move( ). The method setstep( ) is used to set up the step value by which we should move the shooter up and down. The move( ) method will actually move the shooter either up or down based on what step value we have chosen. Likewise, there can be a setstep( ) and a move( ) method within the Balloon class. Then we have a BowAndArrow class wherein within main( ), we can set the step for the Shooter as well as for the Balloon by calling the setstep( ) and the move( ) method using the Shooter and the Balloon object. Then we actually carry out the movement of Shooter and the Balloon by calling the move( ) method. So within main( ), we try to make use of the Shooter, Balloon and the Timer class.

Possible Solution

A possible solution that we can make is a Person class, Timer class and a Target class. Because in place of balloon there might be any other object and in place of one shooter there can be multiple
shooters. Then we create a Shooter class which we derive from the Person as well as the Timer class and create a class called Balloon which we derive from the Timer and the Target class. Then we can create a new Shooter object by saying Shooter s1 = new Shooter(). Likewise we can say Balloon b1 = new Balloon() and Balloon b2 = new Balloon(). So we create one Shooter object and two Balloon objects. Then we have an array of Timer references where we store s1, b1 and b2 using Timer t[ ] = { s1, b1, b2 }. Then we say Random r = new Random(). In a loop we say r.nextInt ( 5 ). Using nextInt(), we can get the next random number. Whatever value is returned by nextInt ( 5 ) is used as a step value and that value is set using t[ i ].setstep( ). Means set the step value for all the entities s1, b1 and b2. Once the step value is set up then we can carry out the movement by calling the move( ) method using t. But in this a problem will come when we try to derive Shooter class from Person and Timer and try to derive Balloon from Timer and Target because we can have multiple inheritance for interfaces but we cannot have multiple inheritance for classes. We are also trying to force an unnatural relationship between Timer and Shooter. The properties a Shooter inherit from a Timer are unrelated.

**Correct Solution**

The correct solution for this problem is we can have a ITimer as an interface instead of a class. Person and Target are indeed be classes. Then we can derive a Shooter from a Person class and Balloon from the Target class. The ITimer interface is implemented by the Shooter as well as the Balloon class. This is the correct way of designing this solution.

**Timer Interface**

In the ITimer interface we have the declarations of methods like setstep( ) and move( ).

**Shooter Class**

The Shooter class is extended from the Person class and implements ITimer interface. So Shooter is a derived class whereas Person is the base class. Within the Shooter class we have a private int step. Then we have the implementations of setstep( ) and move( ) methods. In setstep( ) method, we are going to store whatever is received in s to step variable i.e. step = s. In the move( ) method, we print the step value and then increment it such that every time move is done, the step value is incremented.

**Balloon Class**

Balloon class is extended from Target class and also implement ITimer interface within Balloon class. It also contains the variable private int step. This class also provides the implementation of setstep( ) and move( ) methods. In setstep( ) method, we are going to store whatever is received in s to step variable i.e. step = s. In the move( ) method, we print the step value and then increment it such that every time move is done, the step value is incremented.

**BowAndArrow Class**

BowAndArrow class makes use of these different classes. Within main( ) we create a Shooter object, two Balloon objects using Shooter s1 = new Shooter(); Balloon b1 = new Balloon(); Balloon b2 = new Balloon(). In the ITimer array of references we can store s1, b1 and b2 using ITimer t[ ] = { s1, b1, b2 }. Using Random class we will try to generate random numbers. Each time we will call a setstep( ) method and pass new random number to it such that step value is generated randomly. Then through another loop, we can keep calling the move( ) method.
Packages - I

In this lecture you will understand:

* Anatomy of a NetBeans Project
* What are packages
* What do we gain from using packages
* How to create packages and use them
Anatomy Of A NetBeans Project

We are creating projects and giving names to them. Suppose we create a project whose name is J43 and for that project name a folder gets created in a JavaPrograms directory. This JavaPrograms directory is present in a different subdirectory chain if it’s Windows Vista and in a different subdirectory chain if it’s Windows XP that we are working on. As shown in the slide, the 1st path is for Windows Vista whereas the 2nd path is for Windows XP. The Documents and Settings folder that we have in Windows XP is no longer there in Windows Vista instead there is a folder known as Users. Inside Users there is a subfolder Kanetkar inside which there is a directory JavaPrograms where we decided to organize all the NetBeans project. J43 is a project name and this folder is created in the JavaPrograms directory. C: \Users\Kanetkar\JavaPrograms is the path for Windows Vista and C: \Documents and Settings\Kanetkar\JavaPrograms is the path for Windows XP. If we move to the J43 folder in Windows Explorer, we would see lots of other folders in this J43 folder. For example, we have a build folder, src folder, dist folder, test folder, nbproject folder and we have two files a manifest.mf file and a build file which is an xml file.

Project Contents

The program that we type, they always have extensions of .java. So, all the .java files are stored in the src folder of the project directory. When we compile the program, that time all the .java files get converted into .class files. All the .class files get organized in the build subdirectory. Then we have a dist directory which contains a jar folder which has .jar files. In addition to that in the dist folder, there would be also any javadoc files that would decide to create as documentation for the program. So jar files and the javadoc files are present in the dist folder. Using these jar files we can compile the program on any other machine as well. This is a distribution mechanism. If we create libraries, we can also distribute those libraries in the form of jar files. Jar stands for Java Archive. In the nbproject folder we have a ant Script and name of the script file is build-impl.xml means it’s a xml file. ant stands for a neat tool. ant script is replacement for makefiles. Instead of compiling the program and instead of guiding that compilation through makefiles, that guidance is now done by the ant script. ant came into existence when Tomcat server was created. Then we have a build xml file used for customizing the standard ant script. If we want to make any changes in the ant script those we have to register in build.xml file. In test folder we can keep all the test classes. We have manifest.mf file which contains metadata about the project which includes version information. Versioning support is achieved by using manifest.mf.

NetBeans Views

Whenever we create a project, we can easily view the different classes present in the project as well as different files present in the project. NetBeans afford two different views for the same project; one is what different classes does it have and second is what different files are present in that project. Within the IDE of NetBeans at the extreme left of the Window, we see Files and the Projects options. The moment we click on that we will see the Project View and when we click on Files, we will see the Files View which is indicated as shown in the slide using arrows. In addition to the current project, the different projects that we have created so far are also shown. For example, in a Project View pane we can locate JavaApplication1, JavaApplication2, etc. All are the names of the projects that were created on the machine. We can also locate that there is a JavaApplication7. If we click on JavaApplication7, we would be able to see different source packages. When we click on the branch of Source Packages, we can see JavaApplication7 inside which there is a Main.java. Within Main.java, there are different Fields, Constructors, Methods, etc. In addition to Main.java, there is one more file called Sample.java and in Sample.java also different Fields, Constructors, Methods, etc. are present. If we click on the + sign of Fields, Constructors, those sub-branches will also get expanded. There are Test Packages, Test Libraries and a Libraries folders present. If we click on the Files view, we would be able to look at the project from the point of view of what different files have
been created in the project. For example, in JavaApplication7 we see the build folder, dist folder, nbproject folder and a src folder. If we locate the src folder, inside that there is a javaapplication7 branch inside which we can see Main.java and Sample.java. The other different folders and branches are also shown in the slide. As shown in the slide for Main.java, Main.class gets created in the build directory and Sample.class gets created for Sample.java.

What Are Packages

Classes are container for different variables and functions. Variables are also known as fields and functions are also known as methods. In addition to this, we also have different interfaces. For example, there might be an interface I which contains declarations of three functions fun5( ), fun6( ) and fun7( ). In addition to that there might be an enumeration or an annotation. Enumeration is a special type of class whereas annotation is a special type of interface. Since enumeration is a specific type of class and annotation is a specific type of interface, in general we can say that the Java type system contains classes and interfaces. All the classes, interfaces, enumerations and annotations are put together into a container known as Package. So package contains one or more types where type would a class, an interface, an enumeration or an annotation. All of these are together present inside a package. Package acts as a container for related types. Means the classes, interfaces, enumerations and annotations are put together in a package all of them must be have some or the other relations. Package also provides access protection. The way private, protected and public are the access specifiers for the class, similarly whatever is present within a package what is accessible and what is not is controlled within a package. Package also makes it easy for other programmers to locate the different types and then use them appropriately. Packages also help in avoiding naming conflicts.

What Do We Gain

Suppose we have a package Graphics having four different classes Rectangle, Circle, Spline which draws Bezier splines on the screen and Freehand which draws freehand drawing within a window on the screen. By organizing all these classes within a Graphics package, we get a intuitive indication that for everyone these types are related. This indication everybody gets because all of them are supposed to be related types that goes into package Graphics. It also gives an indication as to where to find graphics-related functions. Package always creates a new namespace for itself. So any name conflicts that occur would get avoided. Types in the package have unrestricted access to one another. There is an unrestricted access to types outside the package.

Package Declaration

Suppose we have a package simpleinterest within which we have a public class SimpleInterest inside which there is a public method called calculateSi( ). We calculate simple interest on the basis of whatever values of principle, number of years and rate of interest we receives in the calculateSi( ) function. We return the value of si from calculateSi( ), package is a keyword. Package declaration is the very first line in the file or a program. By saying package simpleinterest, we are saying that every single type that we are declared within this file belongs to the simpleinterest package. We are declared only one type and it is a class SimpleInterest. But below the SimpleInterest class, if we define few more classes, interfaces, enumerations or annotations all these different types that are present in the current file, all of them considered to be belonging to the simpleinterest package. There is only one package statement per file. To access a particular type from outside the package, it has to be declared as public. So we have declared class SimpleInterest as public. Unless it is public, it is not accessible from outside the package. Name of the .java file and name of the public type within this file has to be same. Within one file there has to be only one single public type.

Package Usage

Go to the File menu of NetBeans and select New File.
Slide 10
After that we will get a dialog as shown in the slide. Within this dialog from categories select Java Classes and from File Types select Java Class and click on Next. We are trying to add a new class to the existing project and name of that project may be J43.

Slide 11
After clicking on Next we get a dialog as shown in the slide. We have to give the name of the class and name of the package within it. So we are creating a Sample class which goes into a sample package. Finally click on Finish.

Result
Now in the J43 folder inside which there is a src folder inside which there is already a folder called j43. Inside src along with j43 one more folder would now get created whose name we have given as sample. Within this sample folder, there is going to be Sample.java. Since Sample is the name of the public type, name of the file is Sample.java. While creating the Sample class we had given the name of the package as sample so at the beginning there is a statement package sample ; is added by the wizard. We wish to make use of this Sample class within Client.java file which is created within the J43\src folder. In this file, we have a public class Client and within that we create a new Sample object and then we try to call a show( ) function which is present inside the Sample class. To be able to use the Sample class, we have to say import sample.Sample. Means inside the sample namespace there is a type called Sample which we are trying to import. Unless and until we add the import statement, we would not be able to make use of Sample class by saying Sample a = new Sample( ). If we do not do import then we will have to make use of a fully qualified name for that Sample class. Means we have to indicate clearly that this Sample class belongs to which particular package. So we have to say sample.Sample a = new sample.Sample( ).

Package FAQs
• What happen if we do not mention the package name?
  o In that case all the types in the file are belong to default package.
• Should package name always be in small case?
  o That is a good idea and is always be used by NetBeans and it avoids name conflict with class / interface names.
• How do I ensure uniqueness?
  o Always use reversed Internet domain names for creating packages. For example, package name is com.ksetindia.simpleinterest or com.ksetindia.quest.
• What happen if there is a hyphen in domain name?
  o In such as case replace it with underscore like com.google-ads \rightarrow com.google_ads.
• What happen if there is a name conflict within the company?
  o We evolve a convention within the company to resolve it.
• All packages in the Java language itself begin with java. or javax.
In this lecture you will understand:

* What are split packages
* How to access types in multiple packages
* How to create and use nested packages
* Access control mechanisms
### Split Package

Suppose we have a public class `Sample1` inside which we have a `show()` function which displays a message `Hello`. This `Sample1` class belongs to a package called `sample`. Then we would have one more public class `Sample2` which has a public function `display()` and prints the message `Hi`. We put this `Sample2` class as well within the same package sample. We would want to use both `Sample1` and `Sample2` classes which belongs to same package within `main()` which is present in a different package. To create these packages, go to the File menu | New File then give name of the class as `Sample1` and package as `sample`. Repeat same for class `Sample2` and after typing `show()` function in `Sample1` and `display()` function in `Sample2` save both the files. First file would get created with the name `Sample1.java` and suppose it would get created in J44 folder which is the name of the folder for this program. Inside which there would be a sample folder and inside that `Sample1.java` would be created. Second class is present in `Sample2.java`. But `Sample1.java` and `Sample2.java` have been stored in the same folder sample because they both belong to the same package. Name of the package is always same as the name of the directory. Name of the file is same as the name of the public type within it. When we compile the project the .class files i.e. `Sample1.class` and `Sample2.class` would get created in J44\build\classes\sample folder.

### Using Split Package

`main()` belongs to a class `Client` and it belongs to a package `j44`. At the beginning we say import `sample.Sample1` and import `sample.Sample2`. Now within `main()`, we can easily create a `Sample1` object by saying `Sample1 s1 = new Sample1()` and call the `show()` method inside the `Sample1` class using `s1.show()`. Likewise create the `Sample2` object using `Sample2 s2 = new Sample2()` and using reference `s2` call the `display()` method i.e. `s2.display()`. A package can be split over multiple files. Each file can contain only one public type. Hence, `Sample1.java` had only one public class and `Sample2.java` also had only one public class. Instead of saying import `sample`. `Sample1`, we can also say import `sample.*`. `*` means import all the public types present in the sample package.

### Multiple Packages

As shown in the slide, both the packages contain the same `Sample` class. So, we have a public class `Sample` which belongs to a package `sample1` and contains method `show()` which prints the message as `Hello`. There is another class `Sample` which belongs to the package `sample2` and has method `display()` which displays the message as `Hi`. We can create this by using File menu | New File and give name of the class as `Sample` and package name as `sample1`. Repeat same for class `Sample` but different package name i.e. `sample2`. Type the functions in the file and save it. When we save this, it will get stored in a `sample1` and `sample2` directory and inside that there is a file called `Sample.java`. When we compile this program `Sample.class` files would get created in the two different folders `J45\build\classes\sample1` and `J45\build\classes\sample2`.

### Using Multiple Packages

The client program has a package called `j45`. Inside this we have a class `Client` wherein we `main()` and inside `main()` we try to create a new object by saying `sample1.Sample s1 = new sample1.Sample()`. Means we create an object of the type `Sample` which belongs to package `sample1` and store the reference in `s1`. At the beginning we say import `sample1.Sample`. Then we say `s1.show()`. Likewise we create another object of the type `Sample` but this time `Sample` class belongs to the `sample2` package. At the beginning we say import `sample2.Sample`. We say `sample2.Sample s2 = new sample2.Sample()` and then call method `display()` using `s2.display()`. `sample2.Sample` is a fully qualified name and in such a case we do not need import statements. We cannot drop fully qualified name because the name of public type in both the packages is same. So to identify them we have to precede them with the package name. In place of import `sample1.Sample`, we could also say import
sample1.*. We can completely eliminate the import statements within this program because we are using fully qualified names in main( ). So wherever we use fully qualified names, the usage of import is unnecessary. But it is better to import at the beginning and then use the type name in the program instead of using long fully qualified names.

**Nested Packages**

Suppose we have a public class Sample having a show( ) function within it which displays Hello. The Sample class is present within the sample package. Then we have a public class Trial within which we have display( ) method which displays message Hi. This Trial class belongs to the package sample.trial. sample.trial is a nested package. When we create this using File | New File and then give the name of the class, name of the package, the folders get created. In the slide it is clearly shown that in the J46 project there is a sample folder inside that there is a trial folder. Since we said sample.trial, a trial folder got created in the sample folder. Sample.java is present in the sample folder whereas Trial.java is present inside the trial folder.

**Using Nested Packages**

In the figure it is clearly shown that Sample.java is present in the sample folder whereas Trial.java is in the trial folder which is present inside the sample folder. We create a class Client inside which main( ) is present. Then we create a new Sample object by using Sample s = new Sample( ) and for usage of that class we say import sample.Sample. Sample. Using the references we can call the show( ) method inside the Sample class i.e. s.show( ). Similarly we can say Trial e = new Trial( ) and for usage of Trial we say import sample.trial.Trial. Trial. Then using e we can call the display( ) function i.e. e.display( ). We cannot say import sample.* because if we do so, only the Sample class will get imported; Trial would not get imported.

**import FAQs**

- Which packages are imported by default in any .java file?
  - The current package i.e. if our package is j43 then it would get automatically imported, the default package and java.lang.
- Which packages would get imported by import graphics.A* ;?
  - None. Compilation error.
- Are the following set of statements same?
  - import java.awt.shape.* ;
    - import java.awt.color.* ;
  - import java.awt.*
  - If we combine them into one import then that would not work.
- What do the following import statements mean?
  - import example.ex1.* ;
  - import example.ex2.ex3.* ;
  - 1st means import all the public types that are present in the ex1 package whereas 2nd would mean import all the public types that are present in ex3 package; ex3 is present in ex2 and ex2 is present in example.
- If a class is present in two different packages then will the code compile?
  - Yes. Till we do not start using it.

**Same Package’s Class**
Suppose we have a package called p1 and inside it we have a class Myclass. We also have a package p1 which contains public class First. We have four fields within Myclass i.e.

```java
private int pri = 10;
public int pub = 20;
protected int pro = 30;
int no = 40;
```

Then we have a 0-argument Myclass constructor. We use Myclass within main( ) by creating an object using Myclass m = new Myclass( ) and then we try to access all the fields of Myclass and try to print out those fields i.e. m.pri, m.pub, m.pro and m.no. no has a default access specifier. In this m.pri is going to report an error whereas m.pub, m.pro and m.no work properly. Protected member is accessible to main( ) from outside the class so long as both the classes belong to the same package. For same package's subclass the behavior is same.

**Different Package’s Class**

Suppose we create a package p1 and it has public class Myclass and it contains same four fields within it and a 0-argument constructor. Now we have another class Second which is present in package p2. Then we import p1.Myclass to make usage of Myclass in class Second. Inside main( ) we create an object of Myclass using Myclass m = new Myclass( ) and try to print the values of different variables in Myclass i.e. m.pri, m.pub, m.pro and m.no. When we execute this program, only m.pub will work or accessed within main( ), rest of the fields report an error because we are trying to access them from outside the package p1. Only public members are accessible when we try to access fields from outside the package.

**Different Package’s SubClass**

If we have a package p1 having a class Myclass and contains same four fields and a 0-argument constructor. We have another package p2 which contains class Second which extends Myclass. Inside Second class we have main( ) and we create a Second object by saying Second m = new Second( ) and then try to access and print the values of four fields i.e. m.pri, m.pub, m.pro and m.no. When we do so, only m.pub and m.pro are accessible to main( ) whereas m.pri and m.no are inaccessible. Means a subclass which has been extended from Myclass can have an access to protected and public members of the base class even though Myclass is in different package whereas private and default would remain inaccessible.

**Summary**

Within the same class anybody is accessible to anybody else. Within another class that belongs to the same package default, protected and public are accessible whereas private is inaccessible. If we create a subclass and it belongs to the same package as base class then private is not accessible whereas default, protected and public are accessible in the derived class. If we have a class which belongs to another package as compared to the package in which we are trying to use it then private, default and protected are inaccessible whereas only public is accessible. Means from a different package when we try to access a class member which has not been derived from our class or we have not derived from that class then only public members would be available to our class. If we have a subclass which is present in a different package then the protected and public are accessible whereas private and default would be inaccessible.
Effective IO - I

In this lecture you will understand:

* Expectations from an IO system
* How does Java meet these expectations
* What is stream-based IO model
* Different types of streams
* Program to illustrate the difference
Expectations From An IO System

One of the foremost expectation that we should have is we should be able to communicate with different sources and destinations. Source is a place from where we try to read the data and destination is the place to where we try to transfer the data or write the data. We should expect that there must be a provision in the IO system which allows to read from a source file or write to a destination file. Similarly there must be a facility to read from the keyboard or write to the console. We must also have the ability to communicate with different ports. We should also be able to write data to a network or read data from a network. Then we should also be able to communicate with the devices like Printer, plotter, etc. Another expectation that we could have is that we should be able to read or write multiple types of entities. For example, we might want to read a file on a byte by byte basis or we may want to read it on a character by character basis. Characters are different than Byte because characters are represented using Unicode and are of 2-bytes. Likewise, we should also be able to deal with numbers. We must be able to read integers, floats, doubles, long ints, etc. We should also have the ability to write records into a file and read record from a file. Record is a collection of strings, numbers, etc. We must have the ability to write the state of the entire object on the disk and then read back that state and reconstruct the object as and when we want. The process of writing the state of an object to the disk is known as serialization whereas the process of reading the data back and reconstructing the object once again in memory is known as deserialization.

Java Solution

- These expectations can be met if we perform all input / output using Streams.
- Stream is a sequence of bytes that travel from the source to destination over a communication path. For example, source might be network, destination might be a file on the file system. We may want to read bytes over the network and then write them into a file. Communication path can also vary from one operation to another. As shown in the figure, using a input stream a Java program is trying to read certain binary data from the source or alternatively a Java program using a output stream may want to write something to the destination.
- A program can read from a stream or write to a stream.
- Streams hide the details of communication from the user.
- Streams are linked to physical devices by Java IO system.
- Streams are implemented using classes in java.io package.

The methods that are used for carrying out interaction are same; the implementation however changes based on with which device we are trying to interact.
Byte & Character Streams

- Two types of streams are provided by Java – Byte and Character.
- Byte streams perform input /output one byte at a time.
- Character streams perform input /output one character (2 bytes) at a time.
- Byte streams are used for input /output of binary data.
- Character stream are used for input /output of text data.
- java.io package has many classes to deal with both these types of streams.
- File utility classes are also present in the java.io package.

Byte Streams ( Binary IO )

The different classes that are present in the Byte Streams are derived from the base class i.e. Object class. Every class in Java is directly or indirectly derived from Object class. There are two classes InputStream and OutputStream both of which are derived from the Object class. Both of these classes are abstract classes. From InputStream and OutputStream, there are varieties of classes derived. For example, there is a FileInputStream, FilterOutputStream, etc. The FilterInputStream and FilterOutputStream classes read or write the bytes from the underlying stream. Means within the FilterInputStream or FilterOutputStream object we may try to create using container relationship some InputStream or OutputStream object and using these InputStream and OutputStream we would try to read or write the data. It filters data for contained stream. Few more classes are derived from FilterInputStream and FilterOutputStream. One is BufferedInputStream and DataInputStream. Likewise we have BufferedOutputStream, DataOutputStream and PrintStream which are derived from FilterOutputStream. FileInputStream and FileOutputStream classes perform read / write streams of bytes from file. BufferedInputStream class provides buffering ability. DataInputStream class provides ability to read Java primitives data types.

Character Streams ( Text IO )

Character Stream is basically used for text input / output. Here also we have Object as the base class from which two abstract classes are derived; Reader and Writer. From Reader and Writer classes we further have BufferedReader, InputStreamReader. Similarly, we have BufferedWriter, OutputStreamWriter and PrintWriter classes. From InputStreamReader we again have derived a class called FileReader and from OutputStreamWriter we have the FileWriter class. InputStreamReader and OutputStreamWriter classes allow reading / writing characters from / to a stream. FileReader and FileWriter classes allow to read a character from a file or write a character to a file. PrintWriter class permits to do formatted writing in text representation.

Difference - Byte & Char Streams

In a program, import java.io.*. In a class BasicIO class within main( ) we declare an integer i.e. int i = 123456. We want to write this integer value to a file. We intend to write this value in three different ways. One way is RawWrite( ) i.e. the binary write. If we are doing binary write then we would be using the Byte Stream whereas if we use the CharWrite( ) or a UnicodeWrite( ) functions then we would be using the Character Stream. When we call these functions, they would report the sizes of the files to which we write the same integer. Using RawWrite( ), CharWrite( ) and UnicodeWrite( ), we will find that the file sizes are different. Using RawWrite( ), the file into which the integers get written, the size of that file is only 4-bytes. When we use CharWrite( ), the file into which the integers get written, the size of that file is 6-bytes and when we use UnicodeWrite( ), the file into which the integers get written, the size of that file is 14-bytes. When we try use a RawWrite( ) function, integer in memory is 4-bytes. So, the same 4-bytes are written to the disk
because when we use RawWrite( ), we propose to use a Byte Stream to write output to a file. As against this when we use CharWrite( ) and UnicodeWrite( ), we propose to make use of Character Stream to do so. In i, there are 6 characters here. If we use CharWrite( ) then these will be written as ASCII characters. One ASCII character is 1-byte so for 6 characters become 6-bytes. So file size is 6-bytes. Whereas if we write characters as Unicode characters, 1 character is 2-bytes, 6 characters means 12-bytes plus the encoding format used is also written which is 2-byte information. So file size is turned out to be 14-bytes. RawWrite( ) function receives an integer and while carrying out operation it might throw an exception. So throws IOException is used. First DataOutputStream object is created and within this object a FileOutputStream object is stored. Name of the file that we intend to open to write integer is passed to the constructor of a FileOutputStream object. When we say ds.writeInt( i ), we are using a DataOutputStream reference, calling the DataOutputStream method writeInt( ) which internally has access to the FileOutputStream reference using which it will actually write the integers into the file. So, whenever we use this containership relationship the object that is contained within the DataOutputStream object that underlying object is actually used to carry out the writing. Finally we close the file using ds.close( ). Then we would try create a File object and then call a method length( ) which tells the size of the file. When we print it, it will print 4.

**Slide 10**

In the CharWrite( ) function, we first create a File object and then pass that File object to the FileWriter object. Once again we are using containership relationship. That File object is used to write the integer character by character to the file Char.txt. throws IOException is necessary because when we start using these classes, the methods in these classes have decided to throw an exception which we can throw it on ourselves. i is the integer and we are casting it to the Integer class known as boxing and then using toString( ) method we are converting it to a String. Then close the file and try to get its length and it is 6-bytes this time. The UnicodeWrite( ) function also throws an IOException if it occurs. Within this function, we will create an OutputStreamWriter object and submit a FileOutputStream object to that. To the constructor of OutputStreamWriter, we are not only passing the FileOutputStream object but also the encoding scheme that we propose to make use of i.e. UTF-16 means each character is 16-bits big. Then we say ow.write( ). Once again we convert i, into Integer object using boxing operation and then call the toString( ) method to convert it to a String. Then close the file and print the length of the file. This time the length turned out to be 14-bytes. 6 characters means 12-bytes plus the encoding format used is also written which is 2-byte information, so forms 14-bytes.
In this lecture you will understand:

* Meaning of common IO code snippets
* Working of Stream IO
* Reading Strings
* Record IO
What Were We Using…

We say `System.out.println()` is used to display the message or output on the console. Similarly `System.in` is used to read a character and store it in some integer `ch`. When we say `import java.lang.*` which is a package and `java.io.*` is also a package and by doing * we are trying to import all the public types. `java.lang.*` package is automatically get imported whereas we have to import `java.io.*` package. `System` is a class and out that we are using is a reference to a `PrintStream` object. Likewise when we do `System.in`, in is also a reference to an `InputStream` object. We are straight way able to access them by using the `System` class because both of them are public static data members of the `System` class. The syntax itself represents that in and out are static data members. The `PrintStream` object reference is derived from a class `OutputStream`. `println()` and `print()` are the member functions of the `PrintStream` class. `PrintStream` class allows to write something to the `OutputStream` i.e. the console. `System.in` allows to read from the keyboard i.e. `InputStream`.

What Happens When…

We create an object of `InputStreamReader` and pass `System.in` to it. We have to follow this procedure whenever we want to receive integer, float, double, etc. from the keyboard. Then create a `BufferedReader` object and while creating the `BufferedReader` object, the reference of `InputStreamReader` object `ir` is passed to its constructor. Whatever object is created of the `BufferedReader` type, its reference is collected in `b`. Using `b`, we say store in the `String s` whatever is read using the `readLine()` method where `readLine()` is a `BufferedReader` class’s method. `BufferedReader` object within it contained an `InputStreamReader` object and within that `InputStreamReader` object there was a reference to `System.in`. The job of `System.in` is to read the characters whereas `InputStreamReader` would try to converts the bytes that are read into characters. It also used to carriage return and line feed. Carriage return and line feeds are the characters which get generated the moment when we hit the Enter key. So using `readLine()` if we supply integer like 123, after that when we hit the Enter key, the carriage return and line feed characters get generated. Carriage returns value is 13 whereas line feed’s ASCII value is 10. So that carriage return and line feed characters had to always follow the line that we had read. Whatever characters are filtered out, they used to gets stored in a Buffer using the `BufferedReader` object.

Slide 5

Suppose we have a class `Charread` which allows to read some characters. Inside the `Charread` class we have `main()` and within `main()` we create a new `InputStreamReader` object and pass `System.in` to it. When we do so, the constructor of the `InputStreamReader` class would get called. Within the `InputStreamReader` constructor, the `System.in` that we used to pass is collected in `InputStream` reference `p`. There is already a field in `InputStream` class called `q` of the type `InputStream`. Whatever we received in `p`, is setup in `q` i.e. `q = p` in the constructor of the `InputStreamReader`. After that control goes back to `main()` where we used to create `BufferedReader` object. When we say new `BufferedReader()`, the constructor of `BufferedReader` class would get called and to that we pass the `InputStreamReader` reference. In the `BufferedReader` class’s constructor, the `InputStreamReader` reference get collected in `InputStreamReader` i and it is to get stored in the private variable of `BufferedReader` class i.e. `j = i` is done in the constructor of the `BufferedReader` class. Once the `BufferedReader` object is constructed, we do `b.readLine()` which reads the line from the `Buffer`. Inside the `readLine()`, function `fun1()` is called using the reference `j` which is a `InputStreamReader` reference. So, `fun1()` must be some method in the `InputStreamReader` class. Inside `fun1()`, using the `InputStream` reference `q`, the `readLine()` function is called and `String str` is passed to it. `fun1()` method of `InputStreamReader` reads the byte from the keyboard, if it is a carriage return and line feed then eliminate that otherwise convert it to a character and return the resultant string. Return of `fun1()` takes the control to the `readLine()` of the `BufferedReader` class. `BufferedReader` class puts the characters into buffer and then returns them. Once the characters are returned we go back to `main()`
and it is collected in String s. So, three classes are involved in carrying out the reading; InputStream class, InputStreamReader class and the BufferedReader class.

Reading Strings From File

Create a class ReadTest which contains main( ) and within main( ) we create new File object. new File( ) will create a file object. We are proposing to read strings line by line from a file called ReadTest.java i.e. the source code itself. We are using / instead of \ because if we use \ then we have to put two \ i.e. we have to say src\readtest\ReadTest.java. Using the File object’s reference we call two functions; one is exists( ) and another is canRead( ). exists( ) will return a boolean value indicating that ReadTest.java exists or not. In this case f.exists( ) return true and then we try to find out where we have permissions to read this file or not using canRead( ) function. If both the conditions are true then we say BufferedReader br = null. We make use of try block but the BufferedReader br is never put in the try block. Within the try block, we create a new BufferedReader object and store reference to the new FileReader object within it. To the FileReader object, we are passing the File reference. Define string into which we read from the file and store each line that we read. To carry out reading we use readLine( ) function of the BufferedReader class to read the file line by line. If it returns null means we have reached the end of file whereas if it returns which is non-null then whatever line has been read, we should print it on the screen by System.out.println ( line ). When we say new FileReader( ) at that time there is a possibility of a FileNotFoundException might get thrown and when we do br.readLine( ) at that time also there is possibility of IOException might get thrown. These exceptions we can either catch or throw.

Slide 7

Closing of the file should happen irrespective of whether we carry out reading or do not carry out the reading. So actual closing will happen in the finally clause. Within the catch block we should have the code only to catch the exception. finally block is always gets executed no matter whether try going to work or catch going to work. So, all cleanup and closures must be done in the finally block. In the catch block we catch FileNotFoundException inside which we just print out the name of the file which cannot be opened for reading. Then the finally block is written if ( br != null ) then only we attempt to close it. If br is already null then no need to close it. So we had initiated br outside the try block.

Record IO

We begin with import java.io.* and within an Empinfo class we have main( ). Inside main( ), we say FileOutputStream fos. Create a FileOutputStream object and to that pass the name of the file i.e. emp.dat. Then a BufferedOutputStream reference bos will be declared. While creating BufferedOutputStream object, the FileOutputStream reference is passed to it. Then we create a DataOutputStream reference dos which actually allows to write the primitives into the file. While creating the DataOutputStream object, we must pass the BufferedOutputStream reference to it. Using this we are able to perform a formatted buffered output to a file emp.dat. FileOutputStream actually writes to a file. BufferedOutputStream buffers the data which is being written to the file. DataOutputStream allows to write standard primitives like integer, float, etc. Then we want to perform reading and hence for reading we create an InputStreamReader object isr1 and pass System.in to it and then wrap the InputStreamReader object into a BufferedReader object. This is for doing the console input.

Slide 9

We declare a String choice and have initial value "y" to begin with so that we can go on continue to run this loop so long as the value of choice remains to be "y". equals( ) is a method of String class. If choice contains "y" then we will enter inside the loop and then ask the user to supply the employee
ID, read this by saying `br1.readLine()` and collect the string that is read into a String variable `temp` and then convert `temp` into an integer by using `Integer.parseInt(temp)` and we write that using a `DataOutputStream` reference by using `dos.writeInt()`. Likewise we can receive the salary using `br1.readLine()` and convert the double which in the form of a string into double by using `Double.parseDouble()` and write by using `dos.writeDouble()`. Similarly receive the name of the employee and the name which is string can be written to a file using `dos.writeBytes()`. At the end we ask the user that does he/she want to enter the data for one more employee. Based on what he supplies, the while loop would continue or stop. Read "y" or "n" into choice using `br1.readLine()`. When choice contains "n", control goes outside the while loop and as soon as control goes outside the while loop, we say `dos.close()` which will close the file.

Slide 10

To verify what we had written to the file has indeed gone to the file or not, we can start reading the file contents and try to display them on the screen. To carry out reading, we will first construct the `FileInputStream` object and pass the same file name i.e. `emp.dat` to it. Then create a `BufferedInputStream` object `bis` and pass the reference of the `FileInputStream` object to it. Create a `DataInputStream` object and pass `BufferedInputStream` reference to it. This construction is done for carrying out formatted buffered input. Likewise we have an `InputStreamReader` object `isr2` and pass the reference of `DataInputStream` to it. Then wrap it around a `BufferedReader` object. This construction is used for reading strings.

Slide 11

We declare a `String` `s` and int `i = 1`. We print the message on the screen that the employee information is now going to print. In an infinite while loop implemented using while (true), we start reading the records. Using `DataInputStream`, read the integer by saying `dis.readInt()` and read double by using `dis.readDouble()` i.e. `Id` and the salary are read. Then we say `dis.mark(0)` means wherever we are within the file that position is now marked. Then we say `br2.readLine()` using which we can read name. Whatever name is read is returned in `s` and printed on the screen. Then we say `dis.reset()` means we want to go back to the position which we had marked earlier and from there we want to skip bytes using `dis.skipBytes()` and the bytes that are skipped are length of name + 1 because we want to go beyond that name. Then we skip `br2` by 12 because double was 8-bytes and integer was 4-bytes i.e. 8 + 4 = 12-bytes. Finally increment `i` to go to the next record by using `i++`. 

Effective IO - III

In this lecture you will understand:

* How to recursively list files
* How to access FileSystem attributes
* How to create our own filters
Listing Files

The output of the program that we are going to write is as shown in the slide. It begins with build/ where / indicates that build is a folder. Inside the build folder, there is a classes/ folder which is slightly indented to the right. Inside classes folder we have fileio as a folder which is further indented to the right. We have files inside the fileio folder, one of the file is BasicIO.class, DirRead.class, FileIOTest.class, ReadTest.class, SysPropTest.class and WriterTest.class. Besides each file in the fileio folder, sizes of the files are listed. Then we go to left and we have dist/ folder inside which we have dependencies.txt file of 1KiloByte. We again move to left and we have dist/ folder inside which FileIO.jar and README.TXT files are present. Means we want proper indentation in the output. We have package called dirread. Then we have import java.io.* and we have public class DirRead which contains main( ). We have static final long OneKb = 1000L ; static final long OneMeg = 1000000L ; and static final long OneGig = 1000000000L ; variables declared outside the main( ) for calculating sizes of different files. Inside main( ), we call a function listFiles( ) which is a recursive function and list out folders and the files along with their sizes. Two parameters are passed to the listFiles( ) method; one is an object of a File class, 1st the File object is constructed and . means the current directory. So, from the current directory it will start to list all the folders and the files. Second parameter passed to the method is the indentation we want. To begin with indentation is 0, so passing an empty string as the 2nd parameter to the listFiles( ) method.

Slide 4

This function is defined as public static void because we are calling it from main(). Two parameters are passed to it; one is File object’s reference and indentation is collected in the form of string. Then we say System.out.println( indent + f.getName( ) + "/" ). Here f.getName( ) will give the name of the current directory and put / along with it. Then we declare an array of File references and within try block we say list = f.listFiles( ) means retrieve all the files present in the current directory. This is a recursive call. In the catch block we catch the SecurityException. When this exception is thrown, we return from that call. In for loop we say for ( File ch : list ) means for each entry in the list, we should check whether that entry is a directory or not. If it is not a directory then we should print the name of the file. Before printing the name, to the current indentation level one more space is added. Besides the name of the file, we want to print the size of the file and that size is placed within [ ]. To retrieve the size, we called the method getSize( ). This will list out files. But if we find the directory then we need to go inside that directory.

Slide 5

Again the same for loop is run. Inside the for loop, check whether the entry is a directory. If it is a directory then we need to go inside that directory and increment the indentation level. So, add one more space to the current level and pass it to the listFiles( ) method. With this as the directory, it will now recursively go inside that particular directory and listing out that files which are present in that particular folder. In this way files within folder, sub-folder will get listed out. So listFiles( ) forms a recursive function.

Slide 6

The getSize( ) function is going to return the string. szStr is initialized to an empty string and retrieve the length of the file using f.length( ) whose reference has been passed to us in getSize( ). length( ) is a function of the File class. Then we check that if ( size < OneKb ) and if it is then convert the number into a string and for that we use a wrapper class Long. So Long.toString ( size ) convert a size into a suitable string. Once the string is created, we return from the getSize( ) method. Then we find if ( size > OneGig ) and if it is then we say szStr = szStr + Long.toString ( size / OneGig ) + "," and the size is reduced by using size = size % OneGig which represents the bytes cannot be
represented by Gigabytes. These balance bytes are compared with OneMeg using if (size > OneMeg) and if it is true, the number of MegaBytes is calculated using szStr = szStr + Long.toString(size / OneMeg) + "," and the remaining bytes are found using size = size % OneMeg. Then balanced bytes are checked with OneKb using if (size > OneKb) and if it is true, then we say szStr = szStr + Long.toString(size / OneKb) + "K" and if it is not more than OneKb, then return the szStr. In this way we can list the sizes of the files.

**File Class usage**

As shown in the slide, we have a package fileiotest then we import java.io.* as well as java.security.* and write the FileIOTest class. Inside the class, we have main(). Within main(), we assign to a string fName a string FileIOTest.txt. Then create a new File object and pass string fName to it. A new File object’s reference is available in f. Within the try block, we say f.createNewFile(). Means when we pass the name of the file to the constructor, that time it does not create a new file. We have to call the method of File class, to actually create a file which is an instance method. So it will create a file FileIOTest.txt. Then we will call another instance method setReadOnly(). When we call this method, the FileIOTest.txt will become a read-only file. So, we cannot write to such a file and if we attempt to do so by saying FileWriter fw = new FileWriter(f) will immediately throw an exception because the file is read-only. Then we close the file using fw.close().

**Slide 8**

The FileNotFoundException will be thrown the moment we attempt to create the FileWriter object. Now we make it writable by saying f.setWritable(true, true). So in catch block, we make a file as read-write file which was initially a read-only file. Now we set FileWriter fw = null. Within the try block, we try to create a fw = new FileWriter(f). Now the exception is not thrown because we made a file as a writable file. To demonstrate that the file is writable we write a string Hello World to it using write() method of the FileWriter class. On the screen we display the message that we wrote contents to the file FileIOTest.txt. To find the absolute path of the file, we call a method f.getAbsolutePath(). If we want to create directories then File class provide this facility. So we create a reference d and say d = new File("d1"). Using the File object, we call a mkdir( ) method which is another instance method of the File class and it will make a new directory within the FileSystem and its name is d1. There is another method called mkdirs( ) means we can recursively build the entire path of directories. Within the current directory d2 directory is created, inside that d3 directory is created and within it directory d4 is created using mkdirs( ) method. Then after try block, we can either write the catch block if we expect some exception or in the finally clause we can do fw.close().

**Filter Streams**

If we have a file which contains abcde, fgh, ijk and we want that they should get converted into alphabets like ABCDE, FGH, IJK the moment they start getting read from the file. To achieve this in between we put a upper-case filter. We have BufferedReader object, inside which we have a UppercaseReader object and inside which we have a FileReader object. FileReader object will actually read the characters. We will pass them onto the UppercaseReader which is a filter. This filter will transform the character into upper-case and feed it to the BufferedReader. BufferedReader will buffer those characters and then we can extract the characters from the buffer. UppercaseReader class is derived from a FilterReader class which in turn is derived from the Reader class. Reader and FilterReader classes are present in java.io package. Reader and FilterReader are abstract classes. So in the UppercaseReader class, we have to implement all the methods in the Reader as well as FilterReader classes.

**Uppercase Filter**
We have a `UppercaseReader` package and we imported `java.io.*`. Then we have a class `UppercaseReader` which extends the `FilterReader` class. Within `main( )`, we expect, two exceptions to be thrown; so we have given `FileNotFoundException` and `IOException`. So we can throw multiple exceptions using the throws clause. Then we create a new `File` object using `File f = new File( "src/fileio/UppercaseReader.java" )`. Means we want to convert the current file into uppercase. After that we try to find out whether such a file is there or not. It is done using a File method `exists( )`. So if the file exists then if it is a file and not a directory and we can read from that file. If all these three conditions are satisfied then only we would start actually reading the file. Before we start reading, we will first create a new `FileReader` object and pass the reference of the file i.e. `f` to it. Whatever is read, passed to the `UppercaseReader` and then whatever is read will have to be buffer so pass it to the `BufferedReader` object. `BufferedReader` acts as a wrapper around the `UppercaseReader`. Once the `BufferedReader` object is created, we can start the reading process. We want to read line by line, so create a string line and within the while loop, we call `br.readLine( )`. As we keep reading line by line, some time we come across a null. When we reach null means we reached end of the file. So at that time we will come outside the while loop. Whichever line we read, we print it on the screen using `System.out.println( )`.

Helper Functions

The `UppercaseReader` constructor receives a Reader reference. `in` represents the underlying character input stream. Within the constructor we pass it to the base class constructor i.e. `FilterReader`. So `FilterReader` will store the underlying character reference in one of its private or protected variable. We have `transform( )` function. Whatever integer it receives will transform it and will return it back. This is a private function. Within `transform( )`, we will check whether `ch != -1`. `ch = -1` when we read the end of file character. End of file character is commonly represented using -1. If it is not -1 and if it is in lower-case we use function `Character.isLowerCase( ch )` then we return a upper-case character using `Character.toUpperCase( ch )`. If any one of the condition fails i.e. end of file, special symbol like +, -, etc., digit or a already capital letter then we would return the character. Then we have a `read( )` function which reads one character from the input stream using `int ch = in.read( )`. `in' is a protected 'in' which is inherited from the `FilterReader` class.

Helper Functions

The `read( )` method can read an array means reading of multiple characters into the `cbuf[ ]` array. It reads characters into the portion of the array, portion which is indicated by `off` and `len`. So start from is the `off` and number of characters is `len`. When we do `br.readLine( )` then `readLine( )` method is going to call one of these `read( )` methods. Inside `read( )` method, we say `in.read( )` where 'in' is protected 'in' came from base class and we pass `cbuf`, `off` and `len` to it. `read( )` method returns the number of characters it is able to read successfully. Then we have to run a loop from `off` to `off + rc` in a step of 1. Each time we run the loop, we will convert the character using the `transform( )` function i.e. `cbuf[ i ] = ( char ) transform ( cbuf[ i ] )`. We have to cast into a char because `transform( )` returns an int. Finally return `rc` means how many bytes the `read( )` function is able to read successfully.
Filter And Other Streams

In this lecture you will understand:

* Input and Output Filters
* How to create output filters
* What are predefined streams
* Streams overview
* Tokenizer program
Encrypt / Decrypt Filter

Suppose we open a file which contains certain characters and the file is a text file. This text file we want to convert in a form which nobody can understand means we want to encrypt the file. For that we wish design a filter and we want to make use of the same filter to decrypt the file to get back the original file. So through the same filter we want to carry out encryption as well as decryption process. The common filter we are using, it makes use of substitution cipher. We will do coding and encoding using the substitution cipher. It is known as substitution cipher because for each character that we read from the source file, we will convert it into some other character before writing it to the target file. For example, as shown in the slide, if the 1st character read from the file is 'a' then we may convert it into a character 'f' before writing it to the target file. Likewise, if we read 'b', we would convert it into 'h', etc. To achieve this we have an interface ITransform which has a transform( ) function which receives an int and returns another int. From the ITransform interface we have an Encrypt and a Decrypt class which will implement this interface within them.

Encryption

We have a package fileio and import java.io.*. Then we will first declare the interface ITransform which has a transform( ) function within it. The transform( ) function receive an int, transform it and return an int. The class which implements this interface, it is the responsibility of that class to provide the definition of that function. First we write the Encrypt class which implements the ITransform interface. Encrypt class provides the definition of transform( ) function. We create a constant string using static final i.e. static final String str = "xyfaghbimourvnqsdewtkjzl". Every character is present in the string and no single character appears twice in the string. Inside the transform( ) function we will check whether the character received is a lower-case character or not using if ( Character.isLowerCase ( ch ) ) and if it is a lower-case character then we will use a String function charAt( ). We are passing ch - 'a' to charAt( ) and finally we will return the character. And if the condition is not satisfied then we will return the character as it is. Means if in a file a digit, capital letter or special symbol is present then we do not intend to transform it. Only lower-case character is encrypted. charAt( ) function will always return the character at a given position within the string. So when we pass ch - 'a' to charAt( ), we expect the character present at that position. Suppose ch contains a character 'b' and when we do ch - 'a' then it becomes 'b' - 'a' i.e. 1. So we want the character present at 1st position within the string. We get the character 'y' at that position. Means we read 'b' and we have transformed it into 'y'.

Decryption

In the Decrypt class we implement the ITransform interface. So we provide the transform( ) function within this. Same static final String str is created and then we write the transform function. Within the transform( ) function, character that is received is checked whether it is a lower-case or not. If it is a lower-case character then we use the indexOf( ) function of the String class. When we say indexOf ( ch ) then we get the position at which ch is present. Whatever is the position obtained, we add 'a' to it, the resultant character is stored in ch and finally that character ch is returned. If any other character is received then that character is returned as it is without applying any other transformation on it. The string must be exactly same as one which we used in the Encrypt class. Suppose ch contains a value 'y' then indexOf ( 'y' ) will be returned as 1. 1 + 'a' gives 'b'.

TransformWriter Class

We create a XformWriter class and extend it from the FilterWriter class. We have main( ) inside XformWriter class and inside main( ) we call functions like doEncDec( ). A source file and a target file is passed to it. Source file is XformWriter.java and target file is enc.txt. The 3rd parameter passed is true indicating that XformWriter.java should be converted into enc.txt and apply encryption
transform while doing so whereas through the same function when we do the decryption, then enc.txt acts as a source file which is the encrypted file and its decrypted file would be dec.txt and false indicates that do decryption rather than encryption. So doEncDec( ) function do encryption or decryption based on true or false. true indicates the encryption and false indicates the decryption. XformWriter.java is the current program itself, is converted into enc.txt which is the encrypted file and if we want to get back the original file from the encrypted file then we have to perform decryption.

Driver Fun.

We define a private variable ITransform trans = null within the XformWriter class. The doEncDec( ) function receives a source string, a target string and a boolean flag variable. Inside the doEncDec( ) function, we create a XformWriter object. This object’s address is stored in the out reference. While creating an XformWriter object, the 1st parameter passed is an address of the PrintWriter object and target file name is passed to the PrintWriter object and target file name is passed to the PrintWriter object. Based on flag, we will either pass Encrypt object to it or Decrypt object to it. We define the XformWriter constructor. Whatever we are passing to it, we are collecting it in reference out and either Encrypt or Decrypt whichever object we are passing to it, we are collecting it in ITransform. Within the constructor, we do super ( out ) i.e. FilterWriter’s constructor is get called. Variable trans is set to the value t means we have either encryption object’s address or decryption object’s address within the trans reference. Now within the doEncDec( ) function, we create a new File object and source file is passed to the constructor of the new File object. BufferedReader object is then created and a FileReader object is wrapped inside it. Value of File reference is passed to the constructor of the FileReader. Once the out object and the BufferedReader objects are constructed, we can start reading using the BufferedReader object. Define a String variable sep and we want to set up the value returned by the function getProperty( ) into it. The getProperty( ) function contains a string "line.separator". The getProperty( ) function retrieves the value of the separator that is used on the operating system on which we are trying to run the program. We can start reading by using br.readLine( ). We continue within the while loop, so long as the readLine( ) function returns a non-null value. We must write the line each time we read it. So we say out.write ( line + sep ). Finally we close the output stream by using out.close( ). When we do out.write( ), one of the functions within the XformWriter class would get called.

write( )s

We have three versions of write( ). One write( ) method receives an int. Then it would call the write( ) function and pass trans.transform( c ) to it. trans contains either the encryption object’s reference or the decryption object’s reference. Based on that the suitable transform( ) function would get called. Then we have another version of the write( ) function. The write( ) function receives the char[ ] reference, off and len. off indicates from where to start writing and len indicates how many bytes to write. We run the loop from off to off + len. Each time through this loop, we call the transform( ) function, pass to it the ith character. Whatever it returns, we cast it into char and store it in ch[ i ]. Then we call base class’s write( ) function and pass ch, off and len to it. The third version of write( ) receives String str, off and len. This function first converts the string into a char[ ] using a function of the String class toCharArray( ). Then call the 2nd version of write( ) function defined and pass ch, off and len to it. Which write( ) method is called by out.write( ) method after reading the line, we have no control over it. Whichever write( ) method is called, we have to provide the appropriate transformation. When we execute the program, the XformWriter.java would get converted into enc.txt and enc.txt would get converted into dec.txt. The XformWriter.java and dec.txt would turn out to be exactly same byte for byte.

The Predefined streams

- java.lang package defines a class called System.
It contains three predefined public static variables - in, out, err where err stands for error.

They are accessible from any part of the program.

System.out refers to standard output stream (console).

System.in refers to standard input stream (keyboard).

System.err refers to standard error stream (console).

These streams are already open & ready to supply/accept input/output data.

Streams Overview

Five different groups of classes are defined for streams in java.io package.

- For building different types of byte & character streams.
- For filtering and piping.
- For reading/writing primitive values and strings.
- For Interacting with files.
- For object serialization.

Reader provides a character stream analogous to byte streams InputStream provide.

Writer provides a character stream analogous to byte streams OutputStream provide.

Methods of Reader are similar to methods of InputStream.

Methods of Writer are similar to methods of OutputStream.

InputStreamReader & OutputStreamWriter are converter classes.

They translate between character and byte streams.

They use either a specified character set encoding or the default encoding for the local system.

Specials Streams

Streams that have input/output pairs and byte stream & character stream variants those are

- Filter streams
- Buffered streams

In-memory streams are

- ByteArray streams
- CharArray streams
- String streams

I/O Streams that do not have an O/I counterpart means there is input/output by output/input counterpart is not present are

- The Print streams
- LineNumberReader
- SequenceInputStream which provides concatenation of all the underlying input streams.
Streams that are useful for building parsers like

- Pushback streams allow to unread the character that we are already read from the stream. This is useful for breaking input into tokens.
- Tokenizer streams allow to parse different things read from the streams into tokens. This stream implemented in a class which allows tokens to be read one token at a time.

**Simple Tokenizer**

Suppose we want to write the tokenizer which reads from the file Tokens.txt and it contains information as shown in the slide. If we tokenize this then it can get converted into a form as shown in the slide. We had separated each word present in Tokens.txt and while printing we are ensuring that all the words belonging to 1st sentence prepended with line number i.e. 1 and all the words in the 2nd line are prepended with the number 2.

**Program**

We have a package mytokenizer and import java.io.*. Inside class MyTokenizer we have main( ). Inside main( ), we create new FileReader object and pass name of the file to it i.e. tokens.txt. Then we wrap the FileReader object that we constructed into the BufferedReader object. Then BufferedReader object’s reference we should pass to the StreamTokenizer object. StreamTokenizer class allows to identify identifiers, numbers, quoted strings and comments. Then we call ordinaryChar( ) function of the StreamTokenizer class and pass a character ‘.’ to it. It indicates that we want ‘.’ to be treated as just any other character. When we say st.wordChars ( '"', '"' ) means we treat ‘ as part of word. We can give range to wordChars( ). ’ within two ‘ is never acceptable. So we use \ as an escape sequence. Now we start reading the token from the input stream i.e. from tokens.txt. For that we call nextToken( ) method of the StreamTokenizer class. We go on calling nextToken( ) so long as we do not reach end of file. Each time through the call we check what is the token type. So st.ttype gives the token type that e have read from the stream.

**Slide 14**

Within switch we have case StreamTokenizer.TT_WORD. TT_WORD is a static member of the StreamTokenizer class. Inside this case we print the value of that token using st.sval and also the line number in which that token is present using st.lineno. Line number is put within a pair of parenthesis. Similarly we can check for the case StreamTokenizer.TT_NUMBER and within this case also we print the line number and value of the token using st.nval. If the token is neither of the two then the control would land into the default case and there also line number along with the character is printed using ( char ) st.ttype. Once the control goes outside the while loop, we should close the FileReader stream.
Serialization

In this lecture you will understand:

* What is Serialization & Deserialization?
* Where is it useful?
* How to Serialize selectively?
* Problems in selective Serialization
* How to overcome the selective Serialization problems?
* Program to illustrate Serialization of a collection
Serialization

Serialization means storing objects persistently. Disk is a persistent storage medium because whatever we store on the disk today is available to us to the next day. Reconstructing the objects back is a deserialization process. We need to store the objects in such a form that restoration of objects back into memory should be feasible. When we try to reconstruct the object i.e. during deserialization, the constructor of the class does not get called. Serialization and deserialization is used in inter-process communication. When we want to store objects itself into a database then serialization is used. When we visit a site like Yahoo.com, it will personalize the site according to our preferences. The terminology associated with preferences is profiles. We may not want to serialize an entire object. Objects may not be as simple as the object that we have been creating so far. One object may contain reference to another object within it. So if 1st object is serialized then another object must be serialized. So in this way there might be chain of objects. In that case serializer has to keep track of what has to serialize and what has not serialize. Suppose we want to serialize an object which contains four fields; time at which the object has been created, id, name and the price. These four fields we want to serialize in a file called Cart.dat. During deserialization we should be able to reconstruct all of these back.

Program

The package name is serialization1 and we import java.io.* and java.util.*. The name of the class is Serialization1 which contains main( ). Inside class first we create a constant fname i.e. static final String fname = "cart.dat". Within main( ), we do several things related with serialization and deserialization. While carrying these out three types of exceptions may occur i.e. IOException, ClassNotFoundException and InterruptedException. Within main( ), we have two references p1 and p2 to a class Product. Then we create a product by saying p1 = new Product ( 1, "Quest", 500.0f ). This object we had created want to serialize by using the saveObject( ) method. Then we invoke a function sleep( ) which is a Thread class method. This sleep( ) method postponed the execution of next instruction by 2000 milliseconds i.e. 2 seconds i.e. do nothing for the next 2 seconds. Then we restore the object by calling a method restoreObject( ). This method is responsible for carrying out deserialization. Once we reconstruct that object, its reference is collected in p2. Then we print both the objects. We have to override a method toString( ) within the Product class then System.out.println( ) would be able to print the objects. Output of the program is shown in the slide. Last three elements in the output are same i.e. 1, Quest, 500.0. Means 1, Quest, 500.0 is present in the object p1. It got stored on the disk in a file cart.dat. This is done using saveObject( ). restoreObject( ) restore it back. So in p2 also we have 1, Quest, 500.0. Time also seems to have been restored. Time for 1st and 2nd objects is exactly same. But this is a problem. Because the 2nd object did not get created this time when the 1st object is created. Ideally we should get 14:25:26 for 2nd object because we got sleep of 2 seconds for it. Hence this time for 2nd object is wrong.

Save and Restore

Saving and restoring is done using saveObject( ) and restoreObject( ) methods. To saveObject( ) we passed Product reference i.e. p1. So we collect it in Product p. Then we create a reference of ObjectOutputStream. This allows us to write an object to a stream. A FileOutputStream reference is passed to the new ObjectOutputStream. The name of the file i.e. cart.dat is passed to FileOutputStream reference. The entire object represented by reference p is serialized using out.writeObject( ). Then we close the file using out.close( ). The restoreObject( ) method should be able to read from the file. If it should be able to read then it should be able to reconstruct the object. For that it will make use of ObjectInputStream reference. We 1st create ObjectInputStream object, inside it we wrap a FileInputStream object. Then we say in.readObject( ). Whatever is returned by readObject( ) is casted into a Product and collect in p. Then close the input file cart.dat by using in.close( ) and return the product.
Product Class

Any class whose object we want to get serialized should implement the Serializable interface. This interface does not contain any method. Such interfaces are known as marker interfaces. Inside Product class, we have four private variables i.e. id, Name, price and creationTime. In the 0-argument constructor of the Product class, we create a Date and store it in a creationTime and print a message. In a 3-argument constructor of the Product class which receives id number, name of the product and price, store it in three variables. In the creationTime we store the current date. Then print a message that 3-arg Ctor. Then the toString( ) method is called by the System.out.println( ) whenever we want to print out the object. Inside this method, we will concatenate all the fields together and put them in a pair of [ ]s and return the resultant string.

Transient Data

The dates we got and the object creation time for the 1st and the 2nd object turned out to be same. But it should not have been same. If we want to avoid that then we should not serialize the creationTime data. For that we should declare the creationTime data as a transient data as transient private Date creationTime. This declaration indicates that the date would not get serialized. Then rest of the methods i.e. 0-argument constructor, 3-argument constructor and a toString( ) method remain same. When we declare creationTime as transient, when serialization happens, data gets serialized but during serialization only the id, name and price would get serialized. Date would not get serialized at all. For the 1st product p1, date would get reported as correctly because p1 has indeed been assigned the date of creation. But when we restore the object back from the cart.dat file since during serialization, we serialize only id, name and price; only that part of the object get reconstructed properly. But creationTime is reported as null. So declaring creationTime as transient does not solve the problem.

Solution

We want that due to Thread.sleep( ), the 2nd object’s creation should be delayed. So in the output, we want seconds to be differing between two objects by 2 seconds. So the output we expect is 14:34:28 which is a time of creation for the 1st object and 14:34:30 is the correct time of creation of 2nd object. For that in the Product class which implements Serializable interface, the data will remain as it is. creationTime is declared as transient. 0-argument, 3-argument constructors remain same and a toString( ) method also remains same. In addition to these methods, we should also provide a private readObject( ) method. Within this method, we should receive ObjectInputStream in. Because this is called when we want to reconstruct or deserialize the object. So ObjectInputStream is used here. Within readObject( ) method, 1st we do the in.defaultReadObject( ) such that the id, name and the price should get properly deserialized. To set the time correctly, we say creationTime = new Date( ). When we execute this program then through the output we come to know that the object creation time would be correctly reported through the difference of 2 seconds within them.

Product List Serialization

We can serialize and deserialize an entire collection of products. This time package name is serialization2 and we import java.io.* and java.util.*. Inside class Serialization2 we have a static string, static final String fname = "cart.dat". In main( ) we have throws clause which throws IOException, ClassNotFoundException and InterruptedException. Within main( ), we create a new ProductList object pl1. Then we call saveObjectList ( pl1 ). This indicates that entire product list being serialized. Then sleep for next 2 seconds and then restore this entire ProductList back by using restoreObjectList( ). Then we print out the entire list using System.out.println( ). Within ProductList class we have to provide the toString( ) method. Suppose we have two products in the ProductList. When we execute this, they have been serialized. Their date, id, name and price have been stored. The output is shown in the slide.
Save & Restore

Within saveObjectList( ), we define ObjectOutputStream reference out. In ObjectOutputStream object, we will wrap the FileOutputStream object. Then we do out.writeObject ( p ) and out.close( ). writeObject( ) will properly walk through the list and serialize the object or all the objects in that list. In restoreObjectList( ), we read each object and cast it using ProductList cast operator. Close the stream and return p.

Product Class

Within the Product class, we have three private variables and a transient private Date creationTime. In the 0-argument constructor, we set creationTime = new Date( ). In the 3-argument constructor, id, name, price and date are set. In the toString( ) method, we return the constructed string using creationTime, id, name and price.

Product Class Contd.

In the readObject( ) method, we call the defaultReadObject( ) method to do the reconstruction of that part of object which was declared as non-transient. Whereas for the transient data i.e. creationTime we will set up our new Date. Since this is called after 2 seconds, the creation time will be set 2 seconds later than the creation time that was done for the 1\textsuperscript{st} and 2\textsuperscript{nd} product in the product list.

ProductList Class

The ProductList class also implements Serializable interface. Within this class, we have two products in the Product[ ] array. In the constructor, we run through the loop. plist.length gives the number of elements present in that array. Each time through the loop, we create a new Product ( i, "Quest" + i, 550.0f ). Id is also keep changing. In the toString( ) method, we create a new StringBuilder object, prepend that with two [[]. Then we walk through this array using the for loop. Each time through the loop, we do sb.append ( plist[ i ] ). At the end append two ]]. Whatever is the resultant StringBuilder object, convert it into string and finally return that string. String is immutable whereas StringBuilder creates a mutable string.
Collection Classes

In this lecture you will understand:

* Why do we need Collections
* Cornerstones of Collections
* Advantages of using Collections
* Types of Collections
* Using the Vector collection
Store n Integers

Suppose we want to store 'n' integers in memory. For that we have several choices. We can create 'n' different variables, each variable holding one value. We can create an array of 'n' items and then work on one variable rather than 'n' variables. We can also store the 'n' different integers into a linked list. Linked list is a data structure. If we wish to store 55, 63, 28, 45 and 60 in memory then rather than storing them in adjacent locations in memory, using linked list we can store these integers scattered all over the places in memory. Along with the 1st integer we can store information about the next integers stored in memory. In this way we can store information about the next integer stored in memory. After 60 no integer is present. So along with 60, we can store null value. Any linked list consists of two parts; one is data part, another is link part. Link is reference to the next node. Node is combination of data and link. Linked list is extensible. In array elements are stored in adjacent locations. If we create an array of 1,00,000 elements then so many adjacent locations may not be available to us. In that case, linked list becomes handy because elements in a linked list are scattered all over the memory. Addition, deletion and insertion of elements is easy in case of linked list as compared to an array. If number of elements is not known then we make use of linked list rather than an array. So the 1st approach to take 'n' different variables is a tedious job. With an array the problem is that, we may allocate too much or too little space for an array. Resizing of an array is time consuming. Linked list is a better idea than an array. But the linked list is difficult to actually implement. So, no point in implementing and creating a linked list ourselves. Because these are so common requirements then there are standard classes present in the library which permit us to do this.

Philosophy

There are three cornerstones to these collections. One is, we must be able to store the data in these collections. We should also be able to access the data properly. We must also be able to carry out certain operations on that data. When we try to store a data, it might be linearly stored data or it might be non-linearly stored data. For an array, elements are stored in adjacent memory locations. For a linked list, elements are not stored in adjacent memory locations. So, linear and non-linear storage are the possibilities of storage. When we try to access then there would have methods like set which allows to set data in a collection, get method which allows to retrieve a data from any part of the collection. We should also be able to add new elements to the collection, remove existing elements or insert element in between existing elements that are already present in a collection. In addition to that we should also be able to find out subset of a collection. We might be able to sort the elements, search a particular element in a collection. We can also be able to reverse the elements in a collection. We can copy one collection into another collection, find the frequency of a particular item within the collection, find the minimum and maximum element in a collection, store 0 in place of all the data items in a collection, fill new data into the collection. All these are implemented in the form of static methods. These operations are commonly known as algorithms. Collection is a solution for accessing, storage and operations.

Solution - Collections

Collection is a group of more than 1 elements stored in an object. When we want to store the data, retrieve the data, modify the data or transmit the data, a collection is useful. We may want to send the data present in an array over the network. This transmission capacity is present in a collection. We may want to make use of collections because if linked list, binary tree are required then we do not have to implement these. There are readymade classes present in library which we can use. So there is no need to reinvent the wheel. Reuse is also possible. Effort in building linked list, etc. should be reduced. Performance of the collections is fine tuned to a great extent because these collection classes are time tested. For searching an efficient algorithm is used such that the search becomes faster. Collections can be extended to take care of new user-defined objects. The way we perform operations on different collections should not be too different. If we go through the array from 1st element to the
last element methods that we use to do so should be exactly same as the methods that we use to go through a set of nodes present in a linked list. So there must be similarity the way we perform operations on these collections and the way we access the data present in the collections. Libraries which are otherwise unrelated, if they want to communicate with one another if both of them have used the right set of collection classes, reuse becomes possible to a much greater extent.

**Collection Types**

There are different types of collections that can exist; ordered collection, indexed collection and a mapping collection. The ordered collection is one in which input / output is order based. Means we cannot insert an element in the middle of a stack or a queue. Stack is a Last-In-First-Out collection whereas Queue is a First-In-First-Out collection. The collection can be indexed based. Example of this is an ArrayList and a Vector collection. Means based on the index value, we decide to insert an element. If we want to retrieve a value from an ArrayList at position 5 or we may want to set a value at position 2, then this type of operation is indexed based rather than order based. Third type of collection is a mapped collection where input / output is based on a key. Example of this is a HashTable and TreeMap. This always contains a key / value pairs. For this particular key, this is a value. We insert using a key and retrieve also using a key.

**How To Represent & Manage Collections**

To represent and manage collection, Java Framework provides lots of readymade interfaces. Interfaces are declarations of methods without the body. These interfaces are then implemented in classes which are collection classes. So collection classes implement certain interfaces. Suitable interface for a class is implemented. We also have algorithms which are static methods. Methods like sort( ), binarysearch( ), copy( ), fill( ), min( ), max( ), etc. All are static methods declared in different classes.

**Interfaces**

There is an Iterable interface, an Iterator interface and a map interface. If a particular class implements an Iterable interface; it means we can iterate through the items of this particular class whenever they are being managed in a collection. Not every class objects would be iterable. If a class has implemented the Iterable interface then that class we can iterate through its objects. How to carry out iteration is depends upon whether the Iterator interface stands implemented or not. Iterator will allow to iterate but Iterable will allow to figure out whether that class is iterable or not. If Iterable is implemented then we can iterate through that. To actually carry out the iteration, we have the Iterator interface to be implemented in that class. A Collection interface is derived from an Iterable interface and from Collection interface Set, List and Queue interfaces are inherited. From Set, a SortedSet interface is inherited whereas from Queue interface, a Dequeue interface is inherited. Dequeue interface is one in which there is a double ended queue i.e. addition and deletion can takes place at both ends of the queue. ListIterator interface is inherited from an Iterator interface whereas SortedMap interface is inherited from a Map interface.Iterable interface contains only declaration of iterator( ) method which always return a Iterator object. Using this object, we can then call methods like, hasNext( ), next( ) and remove( ) to either iterate through the interface or remove the element from that collection. hasNext( ) method try to find out whether there is next item present in the collection and if it is there then we can extract that item using the next( ) method. Map interface allow to maintain key / value pairs. Example is a dictionary, address book of a mobile phone, etc. ListIterator allows to iterate from left to right as well as from right to left.

**Collection Classes**

At the base we have AbstractCollection class from which three abstract classes are derived; AbstractSet, AbstractQueue and AbstractList. From the AbstractSet class two classes are derived;
TreeSet and HashSet. PriorityQueue class is derived from the AbstractQueue class. From the AbstractList class ArrayList, AbstractSequencialList and Vector classes are derived. A LinkedList class is derived from AbstractSequencialList class and Stack class is derived from Vector class. This gives the hierarchy of collection classes present in the Java Framework.

**Algorithms**

Algorithms allow to carry out operations on the collections. There are two kinds of algorithms; one is Arrays and other is Collections. Arrays is a class which has methods like fill(), sort(), binarySearch(), copyRange(), copyRange() indicates to copy part of the array into some other array. All these are static methods. We also have copy(), min(), max(), reverse() and shuffle() methods. shuffle() method allows to shuffle all the elements present in a collection in some random fashion.

**Vector**

Vector is an array of variable length. It can grow in size as per the demands of the program. It is resizable so it is changing the size of the array. Vector is an ordered collection. In a particular order we can store elements in the array. Vector is a thread-safe collection. Whenever we run a program, that running is known as a thread. From one thread, we can launch multiple more threads. Name of the package is collections and we import java.io.* and java.util.*. In a class VectorTest, we have main(). Inside main() we create a Vector object by using new Vector(). Then we add different elements to this vector i.e. we add b and 4. Then we create one more Vector object v2 and to this 2nd Vector object we add elements like Hello World. Then we call the method addAll() where all the elements of Vector v2 get added to Vector v1. Then we add few more elements 8.7, Rahul. We are adding int, char, string, double, etc. All these get stored in the form of objects rather than just as primitive values. Then we say v1.add(1, "Deepti"). Means we add an element whose value is Deepti at position 1. Then we print the size of the 1st vector which is turned out to be 7. Then we print the capacity of the vector. The default capacity is 10. The moment we add 11th element, the capacity will grow by 10. So that is a grow size. Then we call contains() method to find out whether Deepti name is present in a collection or not. If it is present then we print the message Found Deepti. We can remove one of the elements present in a collection. Then we can iterate through this collection to find what is present now. For that we call the iterator() method of the Vector class by saying Iterator e = v1.iterator(). iterator() method return the address of the Iterator object which is collected in Iterator reference e. We can now call two methods; hasNext() and next(). hasNext() method continue to return true value till the time there is indeed a next element present in the collection. If there are not any elements then it will return false. To actually get the elements in a collection, we call the next() method. Output is shown in the slide.
In this lecture you will understand:

* How to use ArrayList Collection
* Collection of user-defined types
* Relationship between Inner Classes and Collections
* How to use key-value collections
ArrayList Collection

ArrayList tries to implement variable length array which was resizable. ArrayList is an ordered collection of items. ArrayList is not a thread-safe collection. ArrayList can be synchronized. Name of the package is collections and we import java.io.* and java.util.*. Inside the class ArrayListTest, we have main( ). Within main( ), we create an ArrayList object by using ArrayList arr = new ArrayList( ). Then we call the add( ) method of the ArrayList collection class to add variety of elements to the ArrayList collection. We add elements like a, 43, 6.7, Rahul, true. Then we add element Deepti at 1st position in the ArrayList. If we try to print the size then we get the size as 6. Then we remove one of the element i.e. Rahul by calling remove( ) method. We add different types of elements into an ArrayList. All these elements are boxed into their respective classes. Then ArrayList will maintain a collection of these objects.

Slide 4

We can find the position of element 6.7 present in the ArrayList. We can find the position of 6.7 in the ArrayList using indexof( ) method. We can also find the value of element present at the 0th location using get(0) method. So we can get the index by providing a value or get the value by providing the index. So we get output as 3 and a respectively. Then we iterate through this collection. We call iterator( ) method in the ArrayList class. We get the reference to this iterator and using this iterator we call methods hasNext( ) and next( ). The values present in the collection are displayed and are shown in the slide. Iterator is an interface which has only one method. This interface permits to access all the elements present in the ArrayList collection. It allows to modify the collection. For example, we can remove some elements present in an ArrayList using the Iterator interface. If we want to carry out modification and access simultaneously then we get ConcurrentModificationException. We can reverse all the elements present in the collection. To do so, we use algorithm class Collections which has static method reverse( ). We pass the ArrayList to the reverse( ) method and it will reverse all the elements present in the ArrayList collection. Once it is reversed, we can once again iterate through them using same method. All the elements get reversed in the ArrayList collection and the output obtained is shown in the slide.

TreeSet

TreeSet collection is also an ordered collection. TreeSet is going to be sorted in the natural order i.e. if we take integers as an example then they are sorted by value, if we take names then they are sorted alphabetically, if we have dates in a collection then they would be sorted chronologically. Duplication of items never happens in the TreeSet collection. Name of the package is collections and import java.io.* and java.util.*. Inside the TreeSetTest class we have main( ) and within main( ) we create an object of TreeSet collection; TreeSet ts = new TreeSet( ). Then we add elements to this TreeSet. We add a Product object to the TreeSet collection. We create a Product object by saying new Product( ) and in that we try to store id and name. In the 1st element id is 3 and name is J2EE. This is key / value pair. In this way we create more Products and add them to the TreeSet collection. Then we iterate through the TreeSet. Get the Iterator reference using Iterator e = ts.iterator( ). After that use hasNext( ) and next( ) methods and print the elements in the TreeSet.

Slide 6

Class Product implements Comparable interface. TreeSet maintains a sorted collection. So whether one Product is greater than the other Product cannot be decided by the TreeSet. What is the natural order of sorting for our Product is we have to decide. Hence Product will have to implement the Comparable interface to be able to do this. Within Product we have private int id and private String name. When id and name is passed to the constructor, we collect those values into these private variables by saying this.id = i and this.name = n. We have to implement toString( ) method such that
if we do System.out.println( ) on an object of the Product class, would be able to print that object correctly. Within the toString( ) method we return the string formed by id and name and that string is enclosed between <> brackets.

Slide 7

The Comparable interface forces us to implement the compareTo( ) method. In compareTo( ) method, we receive an object and then we try to find out whether this object is an instance of the Product class or not. If it is not then we will throw an error using assert keyword. When assert will throw an error that time execution will come to halt. Then cast the Object o into Product and collect that in p2. Then try to compare what is present in p2’s name with the name which is available to us as a private variable. For that we call the compareTo( ) method of the String class and pass 2nd name’s string to it. Whatever is the result of comparison is returned from the compareTo( ) method. If 0 is returned, it indicates that names are same. If non-zero value is returned then whether it’s a +ve or –ve, it will decide whether alphabetical order of these names is the correct dictionary order or it is a incorrect order. That way the TreeSet collection would be able to insert the elements in a sorted manner. We also have equals( ) method. We have to implement this method because it is declared as abstract in the base class. Base class of the Product class is Object class. We again check whether o is an instance of the Product class. If so, cast o into a Product, collect that in p2 and then find out whether name is name and the id is also matching. Finally return the result of comparison. If both are matching then we say that both objects are equal. So, one method came from interface and one from the base class. We have to provide the equals( ) method because there is no way to figure out whether two objects are same or not.

Inner Classes

Suppose we have a class Outer. Within the Outer class, we have private int x = 100. Then we have a class defined inside the Outer class as Inner. Inner class contains int y = 50 and we have a display( ) function. Inside display( ), we are displaying y of the Inner class as well as x of the Outer class. Now we have a Sample class which contains main( ). Within main( ), we try to create an object of the Inner class by using Inner i = new Inner( ). Then we try to call the display( ) method by saying i.display( ). We try to print out the value of y saying i.e. when we run the program, we get error as these three statements cannot get executed because we cannot create an object of the Inner class from a place that is physically outside the Outer class. So this is not permitted. To create an object of the Inner class we need to call some method of the Outer class. To call that method, first we need to create an object of the Outer class. So we say Outer o = new Outer( ). Once the object is created, we call the method inside the Outer class. Suppose the name of the method is fun( ). So, we call a method fun( ) which belongs to the Outer class. We define the method fun( ) in the Outer class. Inside the fun( ) method we can create an object of the Inner class by using Inner i = new Inner( ). Then we call the display( ) method of the Inner class and we can print value of y. Inner class’s object will have to be created through an Outer class function. Inner class can access Outer class fields as well as methods.

One More Access

We create an interface test which contains declaration of method display( ). Then in the Outer class, we create an Inner class and the Inner class implements the test interface. Inside the Inner class, we define the display( ) function and displays a message Hi. Sample class contains main( ) and inside main( ), we create an object of the Outer class by saying Outer o = new Outer( ). Then we say test t = o.fun( ). o.fun( ) indicates that fun( ) is a method in the Outer class. fun( ) method is going to return an Inner reference. So the prototype of fun( ) is Inner fun( ). Inside the fun( ) method, we create a new Inner object and return its reference. Then inside main( ), we can call the display( ) method of the Inner class saying t.display( ). Inner class can be accessed from outside the Outer class using an interface pointer.
How It Works

We have been doing Iterator \texttt{e = ts.iterator()}, if \texttt{ts} is a TreeSet reference. So call to Iterator, \texttt{hasNext()} and \texttt{next()} is done in all the collections like ArrayList, Vector and TreeSet even though all the three are independent of one another. This has been facilitated because of Inner classes. TreeSet implements the Iterable interface. Iterable interface contains the \texttt{iterator()} method. Class TreeSetItr implements Iterator interface. So TreeSetItr becomes inner class of the TreeSet class. When we do \texttt{ts.iterator()}, we are trying to call method of the outer class because \texttt{ts} is reference of the TreeSet. So outer class method \texttt{iterator()} is shown in the slide. \texttt{iterator()} method returns an Iterator reference. An object of the Inner class is created and once this object is created, its reference is returned. This reference is collected in \texttt{e}. If we say \texttt{e.hasNext()}, we can imagine that the \texttt{hasNext()} method has to be a TreeSetItr method i.e. the inner class method. This method returns the boolean by 1st checking whether there is a subsequent element present in this collection or not. If it is there then it will return true and if it is not then it will obviously return a false. If another element is there then we call the \texttt{next()} method. \texttt{next()} is also a method of the Inner class. \texttt{next()} method returns the TreeSet object in the form of Object. TreeSet contains a Product object. So, a Product object is returned in the form of an Object.
In this lecture you will understand:

- Need for generalized Collections
- What are Generics?
- How to use Generic Collections?
- How to create user-defined generic collection?
Requirement

Suppose we have a task where we want to manage a collection of some objects of the type Product. If we want to manage such a collection then we can think of using either a LinkedList of Product objects or an ArrayList of different Product objects. We can make use of ArrayList to store collection of Products.

Product Class

We will first create a Product class, create objects of the Product class and then store them in an ArrayList collection. We have to declare two private variables int id and String name inside the Product class. Then we have a constructor which receives an int and a String and stores them in id and name respectively. We have a display( ) function which will display the id and name on the screen.

Dump Into ArrayList

Now we want to build objects of the Product class and store them in an ArrayList. Name of the package is templates and we import java.util.*. Name of the class is Templates1 and inside main( ) we create an object of the ArrayList. Now we can add objects to the ArrayList using ts.add( ). We are trying to add different Product objects to the ArrayList. The id and name is passed to the constructor of the Product class. In this way we can add multiple objects in the same way. We also try to add string to the ArrayList. This is grammatically perfect. Now if we want to iterate through an array of Product objects which are present in the collection then we can use the standard way. We first call the ArrayList class’s iterator( ) function because ArrayList implements the Iterable interface. The reference is present in i. Now we can iterate through the ArrayList collection using a while loop and hasNext( ) method. We can retrieve the objects by using i.next( ). ArrayList is a heterogeneous collection which can store anything like, int, float, double, char, etc. i.e. objects of different types can go into ArrayList. ArrayList is a collection of objects and all classes in Java are derived from Object. So not only a Product object but anything primitive data can be stored in an ArrayList. next( ) method returns the value in the form of an Object, we have to cast it suitably i.e. Product. Its reference is stored in p. Then using p, we can call the display( ) method. For Product P1 and P2, there is not any problem. Problem will come when we try to extract string from the ArrayList because string cannot be cast into an item of the Product class. The error is an exception has occurred and name of the exception is ClassCastException. This exception has occurred because we try to create a heterogeneous collection of Product strings unless and until we try to create an array of ArrayList objects containing only objects of the same type. Homogeneous collection is better idea then heterogeneous collection and if at all it is a heterogeneous collection then at least into that collection there must be objects which are part of an inheritance chain. The code is buggy means during compilation the code is parsed but during execution we get an exception. The code is slow because ArrayList collects only objects. It never collects the primitives. Primitives go on stack whereas objects go on heap and heap works slowly. It is ugly because it unnecessarily do the type casting.

Solutions

- We can create different ArrayList classes for different types. When we use this solution means we are trying to specialize the code based on the type.
- We can create one Generic ArrayList class for all types. This is a better solution. This indicates that we share the same code no matter what the type is. This solution is implemented by generics.
- Generics were invented for implementing generic collections.
- Many other J2SE libraries use them.
• Generics refer to Generic types and Generic methods.

• Generics types and methods differ from regular types and methods in that they have type parameters.

**Generic ArrayList**

We have the same package templates and name of the class is Templates2. Within the class we have main( ) and inside main( ), we say ArrayList <Product> pl = new ArrayList <Product>( ). This indicates that we are proposing to create an ArrayList of Products. Using this we invoke the constructor of the generic ArrayList class. Now we can add products to the ArrayList using pl.add( ). When we say pl.add ( "hello" ), compilation error will be displayed because we are saying that the ArrayList is a homogeneous collection of Products and we cannot add strings to that. While iterating through it, we create an iterator which works only on Products. So Iterator reference i is created which allows to iterate only through Product objects. Then actual iteration is carried out. i.next( ) return the Product object and we will collect it is p. Here no casting is required. The ArrayList class is a generic ArrayList class. Other collections are also generic. Generic collection will also have type parameters. The type parameter here is Product and it is the actual argument. When we say pl.add ( "hello" ), it indicates the homogeneity and produces a type-safe collection. Typesafe means the program would compile without any errors and there will not be any ClassCastException that will be thrown at runtime.

**Generics - Tips**

• There are three ways to use generic collection types:
  o **Concrete instantiation** where we use concrete type arguments for the generic collection.
    
    ArrayList <Product> pl = new ArrayList <Product>( ) ;
  
  o **Raw instantiation** where we do not provide any type arguments.
    
    ArrayList pl = new ArrayList( ) ;
  
  o **Wildcard instantiation** - with wildcard arguments
    
    Beyond scope

• Advantages of Generics:
  o Increased expressive power
  o Improved type safety
  o Implicit type checks and type conversions where needed

• Downside:
  o Only object types can be added to the collection. No primitive types can be added to the collection.

**User-Def. Generic**

Within main( ) we say MyCollection <Integer> ti = new MyCollection <Integer>( ). This indicates that MyCollection is our generic class to which int type parameter is provided. We are trying to create a collection of integer objects. Then we add integers by using ti.add( ) i.e. 5 and 10 are added to the collection. Using get( ) method of the collection, we can retrieve the values that has been added to the collection. So we get the output as 10 and 5. Then we can use MyCollection to create Product objects instead of integer objects. So the generic MyCollection can take care of integers as well as...
Products. We create Product collection and add Product objects by first creating Product objects and then adding them to the collection using add( ) method. We can retrieve the values from the collection using get( ) method and display( ) method. So the output obtained is 1 P2 and 1 P1. Now we say MyCollection td = new MyCollection( ) and we add 5.7f and Hello to the collection using add( ) method. Then we extract the values and print them out. We do not make use of type casting because it is not required. The 3rd part that we had done i.e. td is a heterogeneous collection whereas the 1st two parts ti and tp are homogenized collections. Heterogeneous collections are unsafe. The 3rd way is a raw way of creating the collections. First two are concrete ways.

MyCollection

We say class MyCollection<E>. E is the generic type. Inside class we say private ArrayList<E> items means we are trying to create items of type E. Within MyCollection constructor we say items = new ArrayList<E>( ). Means from within the MyCollection, we are trying to invoke the generic ArrayList class’s collection object. add( ) method just receives the object and add it to the ArrayList using items.add( obj ). get( ) method returns something of the type E using return items.get( i ). We can use any other name instead of E. E is just a placeholder for a type. E is similar to a type but not really a type. On the E there are several restrictions. We cannot say new E within the MyCollection constructor. It is not acceptable. From type E, we cannot derive a new type. Product class contains a 2-argument constructor and a display( ) method same as before.
User Defined Generics

In this lecture you will understand:

* How to create user-defined Collections?
* How to create user-defined Iterators?
* How to create a generic printList( ) algorithm?
* How to create a Function Template?
User-Defined Iterator

We create a new project Templates4 which is a class name. Inside it we have main( ). Within main( ), we create new TCollection of the Product type. Then we add Product objects to this collection by saying pl.add( ). We add 6 products to the collection. In the last two products, there is a duplication of ids. To iterate through this, we can say for ( Product p : pl ). Each time through the loop, we want to call the display( ) method. Output for the program is shown in the slide. Again the Product class contains a 2-argument constructor and a display( ) method. We can also use the old method of iterating through the collection i.e. Iterator<Product> i = pl.iterator( ). TCollection has implemented the Iterable interface. Address of inner class is returned and it is collected in i. Using i, we can call methods like hasNext( ) and next( ). Outer class implements the Iterable interface whereas inner class implements the Iterator interface. We can use Product p to call the display( ) method in the Product class. for loop internally also makes call to the i.hasNext( ) and i.next( ).

Iterable Collec.

class TCollection<E> implements Iterable<E>. E is a generic type. In this case E is replaced with Product. We are trying to create collection of Product objects which implements the Iterable interface. Interface and a method can also be generic. Iterable is a generic interface whereas TCollection is a generic class. Within the class we have ArrayList<E> of items. ArrayList itself is a generic class in a JCF. In the constructor of the TCollection class, we create a new ArrayList object by using new ArrayList<E>. Whenever a new item is added to the TCollection, control would land into the add( ) method which inturn will call the ArrayList’s add( ) function and add the object that it receives to the ArrayList. When we make a call to get( ), it will return a Product object. So E represents Product object in the get( ) method. get( ) method will retrieve the items from the ArrayList by saying items.get ( i ). Then we have an iterator( ) function which will return TIterator ( this ). new TIterator ( this ) will create an object of the inner class i.e. TIterator. this contains address of the object using which we are landed into this function i.e. pl objects address is present in this reference. TIterator’s constructor will receive this.

Inner Class

Class TIterator implements Iterator<E> interface. Within class, we have Iterator<E> itr. Whenever the constructor of TIterator is called, this is passed to it. It will be collected in TCollection<E> c. Inside the constructor we say itr = c.items.iterator( ). Then we implement the hasNext( ) method which returns boolean value and inside hasNext( ), we say itr.hasNext( ). c.items.iterator( ) returns the address of the inner class object of the ArrayList class. Then we implement the next( ) function which will return itr.next( ). Again the ArrayList class’s next( ) is called. In addition we have a remove( ) function which calls the remove( ) of the ArrayList class.

Generic printList( ) Algorithm

Through the printList( ) method, we want that all the List<T> compatible classes can be applied to this algorithm. Whichever classes are of the type List<T> those classes we should be able to apply the printList( ) algorithm. Name of the class is Templates5 which contains main( ). Inside main( ), we create a new Vector collection and we propose to store only integer objects in this collection. Then we add integers to it using add( ) method like 1, 4. When we add 1 and 4 to the add( ) method, it receives an integer but immediately box it into an Integer object. Now we create an ArrayList collection. Again to the ArrayList collection we add integers like 2 and 6. Now we propose to apply the algorithm function printList( ) to the Vector and ArrayList collection. printIntList( ) is a generic function which allows to print integers present in the list. This is a generic algorithm which will work on Vector and ArrayList. printList( ) is a more generic algorithm. It will work on all such classes which are compatible with List collection class. The output is shown in the slide. If we apply
printList( ) on a Vector collection, ArrayList collection then can we apply printList( ) on a collection that we had created? Yes, provided it implements java.util.List interface within it.

Slide 7

The printIntList( ) function returns nothing and it receives List<Integer> tl. Inside the function, we can run the for loop, saying for ( int a : tl ). Each time through the loop, we print whatever is present in a. Likewise we can have a more generic printList( ). Inside this function also, we run a similar for loop and each time through the loop, we print the value of a. a is not of the type int but it is of the type T because we do not know that all integers are present here. It might be all doubles, floats or any objects. Hence the for loop is, for ( T a : tl ).

Function Templates

We have a class Templates6 which contains main( ). Within main( ), we would create two integers a and b with values 4 and 5 respectively. Then we would try to print out maximum out of a and b where max( ) is a function template which will report maximum out of a and b. We propose to have max( ) as generic because we want to find out maximum out of characters 'z' and 'y'. We also want to find out maximum out of two Product objects having values like 1, "de" and 2, "bc". We implement max( ) in such a way that we can compare Product objects based on the names rather than ids. After seeing the output 1, "de" is bigger than 2, "bc". Then we would create an ArrayList of integers by invoking generic ArrayList class. Then we will add several integers to this ArrayList. The integers are 10, 20, 30 and 20. Then we create a function template duplicate( ) which will return the information whether in my collection are there any duplicate objects or not? In the ArrayList of integers 20 is being duplicated. Ten we create an ArrayList of Product and when we create an ArrayList of Product, we will add few products to it. 1, "P1" is added twice. So this is a duplicated entry in the ArrayList. For this the same duplicate( ) function is used. So duplicate( ) should be a generic method. max( ) and duplicate( ), both of them are generic algorithms which we apply on some standard collection.

Slide 9

public static <T extends Comparable> T max ( T a, T b ). By saying <T extends Comparable>, we are saying that we propose to use in the max( ) function such a T which implements the Comparable interface. In case of generics, we have to use a keyword extends and not implements. It indicates that max( ) can work on all such classes which are going to implement the Comparable interface. max( ) works with objects of classes which are ready to implement the Comparable interface. Whenever we create the Product class to compare two products, we have to see that it implements the Comparable interface. max( ) function compares a with b using a method compareTo( ). If a.compareTo ( b ) is less than 0 then it would return b, otherwise it would return a. Then we provide a duplicate( ) function which has a prototype public static <T> boolean duplicate ( List<T> coll ). We can take one entity from a collection and compare it with all others. If there is a match then it is a duplicate. This is implemented by running two for loops. The outer loop is for ( T i1 : coll ). Within the outer loop, we begin with count = 0 and run another for loop saying for ( T i2 : coll ) i.e. for each item present in the collection, go to every other item in that collection. Inside the inner for loop, we start comparing. If two found to be equal then increment the count by 1. Outside the inner for loop, we check if ( count > 1 ) and if it is then return true. Outside the outer for loop, return false. Object contains equals( ) method which compares references of the two objects. But we want to compare values present inside the objects. So we will override equals( ) in T.

Product Class Revisited

Class Product implements Comparable interface. Inside class, we have private int id and private String name. Constructor receives id and name and stores them in class’s private variables. Then we
declare the display( ) function which displays id and name. Then we have the compareTo( ) function which receives an Object, cast that object into a Product and then from that Product we are trying to access the name. This name is compared with name.compareTo(). The name is compared with the name of the argument passed to the compareTo( ) function. If names are match then 0 would be returned. If they do not match then a non-zero would be returned.

**Program Continued**

Then we provide the equals( ) function which is declared in the Object class and we are overriding it here. Once again cast the Object into Product and then check whether names are equal or not using name.equals( p2.name ). If these are equal then true would be returned otherwise false would be returned. We also have a hashCode( ) function. This is used because whenever we override equals( ) we also have to override hashCode( ). This function computes the hash code which is nothing but a number. This hash code is generated to ensure that whenever we input Products into a collection, they are stored in a sufficiently sparse manner such that there are no collisions. Hence the hash code has to be computed for every Product added into the ArrayList. When we use NetBeans 5.5, this hashCode( ) function the moment we try to override it, it gives us hint and we accept that hint. It provides a empty hashCode( ) function. NetBeans 6.0 is much better. It provides the hashCode( ) function, the code within it. While providing the code, it asks us which fields to use to compute the hash code. When we say use name, generate the suitable hash code for it. Inside the function, suitable hash code is generated for id and name and we return that hash code.
Networking Today

In this lecture you will understand:

* What an App needs to achieve network communication?
* Network programming model
* How to get HTTP headers?
* How to download a file using HTTP?
* How to download a file using FTP?
Network Communication

To communicate over a network, we need to know three things; name of the machine, port number through which we want to carry out the communication and the underlying set of the agreed upon rules i.e. a protocol. Machine name internally gets mapped into Internet Protocol Addresses i.e. IPAddress. Port number is able to distinguish between different applications running on a machine. To distinguish one from the other, we need port number. Suppose on a machine an email program is running like Outlook or Outlook Express. We also started a browser like Internet Explorer or Firefox on a machine. Both the applications Outlook and Internet Explorer trying to communicate with some application on the other side of the network like downloading mails. So when our applications are trying to communicate with a Web Server or an Email server then it should not happen that the Email server should start stocking to IE and the Web Server start stocking to Outlook. This leads to miscommunication. So if two different applications are trying to make use of network to communicate with some other application on the other side of the network then how does the application on the other side know that we want to communicate with such and such application on our machine? For that reason a port number is used. Once we start using port number, one application will use one port number; another application will use on the same machine another port number. In this way there will not be any confusion with communication between our application and applications running on the other side of the network. Protocol would indicate how to send the request, what are the rules governing that and how to receive the response? For that there are certain rules. According to those rules if the communication happens, then it would become a meaningful communication. There are protocols in a network communication about how to send a request from our application to the application running on other side of the network. When that application sends a response, how to accept that response? The set of rules governing this network communication is a protocol. IPAddresses are 32-bit addresses. In each 32-bit address, the 4-bytes are separated using . (dot) and the value present in each byte is taken as a number. 198.168.100.2 is an IPAddress. Out of these 4-bytes either 1st byte or 1st two bytes or 1st three bytes will represent the network number and the balance byte will represent the host number. In 198.168.100.2, 198.168.100 will represent the network number and 2 will represent the machine in that network. 1st byte will represent the network number or 1st two bytes will represent the network number or 1st three bytes will represent the network number, according to that the IPAddress is classified as class A, B, C. In Class A, 1 byte is for network number and 3-bytes for host. In Class B, 2-bytes are for network number and 2-bytes for host and in Class C, 3-bytes are for network number and 1-byte for host. While using port number, standard applications will always make use of standard port numbers. Port number allows to distinguish between different applications and different applications will have to make use of different port numbers. Standard applications means like Email client or a browser. They always try to make use of standard port numbers because they always make use of standard protocol. If we are using IE or Firefox browser, that time we are using the HTTP protocol which is always to make use of port 80. If it is a File Transfer protocol then port number 20 or 21 is used. One out of these two is used for sending the data and one is used for sending the command. For sending mails we make use of SMTP protocol i.e. Simple Mail Transfer Protocol and when we use this protocol then the application carry out communication with port number 25 whereas if we use the POP protocol which is used for downloading mails then the application make use of port number 110. These port numbers do not have relations with serial and parallel ports. These are only logical numbers and not physical numbers.

Network Programming Model

The Network programming model always involves the two machines; one is client and another is server. Communication will always happen between client and server. Client will send the request and server will send the response to that request. For example, when we look at email clients like Outlook, Eudora client, they always send mails and receive mails. While doing so they are
communicating with server software known as Microsoft Exchange server. They would be using POP protocol for downloading mails and SMTP for sending mails. When we try to browse certain sites, we might use IE or Firefox browser which is a client piece. On the server side there would be a Web Server which might be an Internet Information Server from Microsoft or it might be Apache Web server. It’s a client server model where we send a HTTP request and we get back a HTTP response. If we want to send a file for getting it store on a FTP server or trying to download a file from a FTP server, it’s a client server model at work. We can use CuteFTP software as a FTP client and then we configure the IIS from Microsoft to make it work like a FTP server. Networking programming model is basically a client-server model. What request is send accordingly we get a response from a server side.

How HTTP Works

If we want to build client software which makes use of HTTP then we have to understand how the HTTP works. From a browser or a client application that we write, we want to send the request to the server and the server would want to respond to that particular request. When we send the request, the request is first send to ISP ( Internet Service Provider ) saying that we want to visit www.yahoo.com. ISP machine may not be aware of where is www.yahoo.com hosted. If it knows it then it will return the IPAddress of that machine on which yahoo’s site has been hosted. If it does not know that then it will try to contact a Domain Name Server ( DNS ). A DNS would resolve the site name that we have passed into an IPAddress and would try to return the IPAddress of the machine where yahoo.com stands hosted. If DNS also does not know about this then in that case it will try to contact the root DNS. There are several root DNS present at important sites like NASA or International Security Institutes. Once the information is obtained, this information is returned back in the form of IPAddress where the site that we wish to visit stands hosted. Once that IPAddress is known then the browser would try to communicate with the server whose IPAddress is returned. When we establish the connection between the client and the server, in HTTP terms people will say that a session has now been established. Once the session stands established, then from the browser application we can make a request saying that we need a file that would be one of the requests. We want to download the 1st page of yahoo.com. So we would try to contact yahoo server saying that get me the home page file such that we can render it in a browser. Once we send the request, a response will come back from the server and then the client will take appropriate action, the moment the response is received. There are different types of HTTP request that we can send. Most common HTTP requests are GET and POST. GET is used for getting a file from a Web server. If we wish to fill up a form and submit it to a particular website then we would be required to make a POST request. There are other requests like OPTIONS, HEAD, PUT, DELETE. Likewise, when a response is send back from a server to a client, response is come back in several different forms. It might come back in the form of some file that we request for downloading or in the form of HTML + JavaScript response or ActiveX control or Java Applet.

Sample Header

The slide shows some sample header which we got when we visited quest.ksetindia.com through the application. This is a list of key : value pairs. In place of one value, there might be multiple values as well. : is used to separate key from the values. Address is a key whereas http://quest.ksetindia.com/ is a value. If we make a request to get some index page from quest.ksetindia.com then the header shown in the slide is likely to get. The value in the Address key indicates the site we had visited when we got this response. [HTTP/1.1 200 OK] indicates that we are using a protocol called HTTP of version 1.1. 200 indicates success. OK is human readable response that has been send. When we get a response as page cannot be displayed that time error number is 404. 500 indicates the server error, 301 indicates the file has been moved permanently and 302 indicates file has been moved temporarily. Then the date on which the request was made is shown. Following that we had a Transfer-Encoding which means that the response that is come back to us is chunked means broken
into different chunks. Expires indicates the esoteric date. Then a cookie number is given. Cookie is
send by the server to the browser and gets stored on the machine. Cookie is the means to identify us.

domain indicates that cookie came from quest.ksetindia.com. Any third party cookies can be
disabled. Connection : [close] means server will close the connection once the response has been
send to us. Then content-type is given. Following that server-name is given. X-Powered-By means
some extensions can be given to HTTP. Then we have Cache-Control entry which controls how
proxies may cache the object. This is the sample header following that there would be contents.

**HTTP Headers**

Begin with package networking and import java.io.*, java.net.*, java.text.MessageFormat and
java.util.*. Then we have a class PrintHeaders inside which we have main( ). Inside main( ), we say
printHeaders ( "http://quest.ksetindia.com/index.php" ) which is a file for which we want to get the
header for. Likewise, we also try to get the header for printHeaders ( "http://microsoft.com/windows"
) and printHeaders ( "http://quest.ksetindia.com/dl/myfile.zip" ). To the printHeaders( ) function, we
are passing a URL or Uniform Resource Locator. In this URL, at the beginning a protocol is present.
Following that protocol we have // and then location number. quest.ksetindea.com is a location which
gets converted into a suitable IPAddress. From this IPAddress we try to go to a particular path on that
machine inside which there is a file. The location quest.ksetindia.com might optionally be followed
by a port number. Following that we will give the path and myfile.zip is the file for which we want to
get the header for. The resource can be a file, might be a directory or might even be a query. The
URLs can be HTTPs, FTP or File.

**Slide 8**

Within printHeaders( ) function we receive the String and collect it is addr. Then we print a message
using MessageFormat.format( ). This method allows to format the text. Then we can create a new
URL object where URL is a class present in networking library present in Java. Once this object is
constructed, using this object we call a method called openConnection( ). openConnection( ) returns a
HttpURLConnection reference which is collected in conn. HttpURLConnection is derived from a
more generic class URLConnection. The object that we had created i.e. URLConnection object
represents a communication link that exists between the application and HTTP URL. To represent
that the HttpURLConnection object is created. Using the conn reference, we can read / write to the
resource that is being referenced by the URL. Using conn, we can make calls to those methods.
openConnection( ) will open the connection with that HTTP URL and it will also make the GET
request. If openConnection( ) returns the code HttpURLConnection.HTTP_OK means everything
went ahead smoothly. Means the request was acceptable. Status code are returned by
getResponseCode( ) method of the HttpURLConnection class. If the condition is satisfied means the
request is successful. Now we need to extract the headers. To get the header information, we use a
Map interface. We call a method called getHeaderFields( ) and it returns a <key, value> pairs. The
object which represents this <key, value> pairs that object’s reference is returned and it is collected in
headers variable. Now we can call a keySet( ) method which will keep returning a <key, value> pair.
It is collected in h and we print the key along with header by saying headers.get ( h ) which will give
the value present against that particular key. Finally say conn.disconnect( ).
In this lecture you will understand:

* HTTP and FTP Download
* What is MD5?
* Where is it used?
* What are its benefits?
* How to use it programmatically?
File Download

The application ultimately communicates with the server by sending request and receiving responses. So we may have on the server a software running as a HTTP Server. On that HTTP Server we have a file called quest.ksetindia.com/dl/myfile.zip. To begin with, we have the location within the URL where the file is present. So quest.ksetindia.com is get converted into suitable IPAddress. dl is the directory and within that directory there is a file called myfile.zip. We also have a FTP server running on quest.ksetindia.com where there is another directory called pub where myftp.zip file is present which is we want to download through the client. For downloading 1\textsuperscript{st} file, we have to make a HTTP Request and for the 2\textsuperscript{nd} we have to make a FTP Request.

HTTP & FTP Download

Name of the package is networking and import java.io.*, java.net.* and java.util.*. HttpDownload is the class name within which we have main( ). We define static final int BUF_SIZE = 100. Inside main( ), we define a variable u of the type URL. Then we say u = new URL ("http://quest.ksetindia.com/dl/myfile.zip"). Then we try to open the HttpURLConnection using u.openConnection( ). Once the connection has been established, a request will be send. Once the request is send, we will try to check the response code to figure out whether the request has been made was successfully accepted and the response was returned to us. If the status code is HttpURLConnection.HTTP_OK means the request is successful. If it is successful then we got to download the file. To carry out the downloading of the file, we will call a function downloadFile( ) which is user-defined function. To this function we pass getInputStream( ) means the connection object will have within itself a InputStream where we can extract the file from where we can read the bytes to that stream. Bytes from myfile.zip file. We will also create one more URL object u = new URL("ftp://quest.ksetindia.com/pub/myftp.zip") and this time we had given FTP URL. myftp.zip file we wish to download from a pub directory. Once again we call downloadFile( ) and pass to it u.openStream( ) and myftp.zip. myftp.zip is a local filename. u.openStream( ) will not only create the FTP connection, it will also make a request to download myfile.zip or myftp.zip. For both HTTP and FTP downloads, we make use of common function downloadFile( ). openStream( ) also returns the InputStream which we can use for reading.

downloadFile( )

This function receives a InputStream ins and a String name i.e. local file name. We first print the name of the file that we want to download. Then we put the InputStream within a BufferedInputStream object. Then we open a FileOutputStream to which we pass the name of the target file. Using FileOutputStream, we try to write this target file. Then we create a byte[ ] array of size BUF_SIZE and then define int len which can be used to figure out how many bytes have been read when we make a call fis.read( ). We are trying to read data from offset 0 and we are trying to read data.length amount of data. It is collected in the array data. Each time we read, we write the data that is present into FileOutputStream that we have opened using fos.write (data, 0, len). Finally we close BufferedInputStream and FileOutputStream and then print the name of the file whose downloading is complete.

MD5 Checksum

- MD5 stands for Message-Digest algorithm 5.
- It was designed by Ronald Rivest in 1991.
- It converts variable length content to a 128-bit fixed length value.
- MD5 algorithm function is a one way hash function.
It is used in a wide variety of security applications:

- Check integrity of files
- Store passwords

MD5 hash is expressed as a 32-character hex number.

A small change in the content gives a different 32-char hex number.

**MD5 Checksum Logic**

From main(), we intend to call two functions; httpVerifyMD5( ) and ftpVerifyMD5( ). We propose to download HTTP request file similarly we want to make a FTP request to download a file and then verify the checksum of the downloaded files. httpVerifyMD5( ) inturn will call downloadHTTPFile( ) which inturn will call downloadAnyFile( ). ftpVerifyMD5( ) proposes to call only downloadAnyFile( ). Once the files have been downloaded then the httpVerifyMD5( ) as well as ftpVerifyMD5( ) will call a function called verifyChecksum( ) and verifyChecksum( ) inturn will call getChecksum( ). Then it will verify whether it matches the original MD5 sum or not. If it is then it will report that MD5 sums are same. Hence the files which are downloaded using HTTP or the file that we download using FTP has been indeed downloaded correctly byte by byte.

**Program**

Package name is networking and we import java.util.*, java.io.*, java.net.* and java.security.*. Inside class VerifyMD5 we have main( ). Within main( ), we call a function httpVerifyMD5( ) and ftpVerifyMD5( ).

**HTTP Download And Verify**

In httpVerifyMD5( ) function, we call a downloadHTTPFile( ) and pass a hard coded URL http://quest.ksetindia.com/dl/myfile.zip to it. It means that download a file myfile.zip which is present in dl directory from quest.ksetindia.com. Similarly, download a md5sum.txt file from the same URL. Once the files have been downloaded, we call verifyChecksum( ) and ask it to find out whether myfile.zip MD5 checksum matches with the MD5 checksum stored in md5sum.txt. This functionality is done by verifyChecksum( ). Then we write ftpVerifyMD5( ) inside it we first create a URL object and then call downloadAnyFile( ) function. This is a common function which is not only called from ftpVerifyMD5( ) but also called from downloadHTTPFile( ). To downloadAnyFile( ) function, pass the u.openStream( ) and string to it. Likewise, for downloading md5sum.txt, we will first create a URL object and once again call downloadAnyFile( ). Then we verify checksum of the downloaded file with the checksum of the file that is present in md5sum.txt.

**downloadHTTPFile( )**

Within downloadHTTPFile( ), we receive the HTTP URL in String addr. Create a URL object using it. Then using the HttpURLConnection reference conn, open the connection by calling u.openConnection( ). It opens a connection and sends the request. Then we can check whether the request was accepted successfully or not by using HttpURLConnection.HTTP_OK. If the response code returned is HTTP_OK means the request is successfully accepted. If it is successfully accepted then we would call downloadAnyFile( ) function. Means for HTTP as well as FTP the actual download will happen through downloadAnyFile( ) function. We will pass the InputStream for the HttpURLConnection as well as the name of the file is passed to it. Name of the file would be either md5sum.txt or myfile.zip.

**downloadAnyFile( )**
Within this function, we first print the name of the downloaded file. The name that is obtained is first split using /. All the parts are collected in an array of strings. Then obtain the last string which is present in the / separated several strings which is the name of the file. Then we create BufferedInputStream and pass InputStream to it. Using the name that we have just now extracted create the FileOutputStream. Once that is over we can now create a byte[ ] array data of BUF_SIZE i.e. 100 bytes. Then we start reading the file from the InputStream. So long as we do not reach the end, whatever is present in the data buffer is written to the FileOutputStream. Then we close the input as well as output stream.

**verifyChecksum( )**

Whenever verifyChecksum( ) receives the name, collect it in a String and then create a BufferedReader object. While creating the BufferedReader object, provide the FileReader object to its constructor. Pass name of the file md5sum.txt to the constructor of the FileReader object. Then we read the contents of the md5sum.txt, store it in a string origSum. Then we close the BufferedReader. So, only objective of BufferedReader and FileReader to read the string that we have in md5sum.txt. Then call a function called getChecksum( ) to which name of the file is passed i.e. the origName which we have downloaded. getChecksum( ) will actually compute the sum and return that sum and collect in mySum. Then find the two checksums are equal are not. If they are found to be equal then we report that checksum is correct.

**getCheckSum( )**

We receive the name of the file on which we want to do the MD5 checksum. First we create a File object. Then we have a DigestInputStream and its reference is set to null. Within the try block, we would try to create an instance of a MessageDigest. MessageDigest.getInstance( ) indicates that this is a class factory function which will create an instance and return the reference to us in m. Then we create a new DigestInputStream object and pass FileInputStream objects reference is passed to it. While creating the FileInputStream object, pass MessageDigest m to it as well as srcF. Then we use DigestInputStream to go on reading the contents of the underlying stream byte by byte. As we keep reading the byte, the MessageDigest will get updated. A update( ) function will be internally called by the read( ) function to update the MessageDigest. Outside the while loop, we get 128-bit number by using m.digest( ) which is collected in byte[ ] md. We use a StringBuilder object to convert the byte[ ] into a String and then for each byte present in the md array, pick a byte, convert it into a String and append it to the existing string present in the StringBuilder. Means using a for loop, we convert a byte array into a String. Finally we can return this string back to whichever function called the getChecksum( ) function.

**Result**

Then we write the finally block. Inside this block, if ( dis != null ) then close DigestInputStream. For the two files that we had downloaded along with their MD5 checksum files, we will find that the result is obtained and it is shown in the slide. The result clearly shows that first we downloaded myfile.zip then we download md5sum.txt. Once that is done, the checksum is shown and that checksum will be exactly same as the checksum that is present in md5sum.txt. This checksum is turned out to be same if the files are downloaded correctly. Exactly same results are shown for myftp.zip. This got downloaded, its MD5 checksum was computed and it did match with the checksum present in the md5sum.txt present in the pub directory. So we downloaded 4 files, computed 2 checksums, match them and found them that they are exactly same.
Network Serialization

In this lecture you will understand:

* What is network serialization?
* How to achieve it programmatically?
* What is multithreading?
* When can it be used?
* How to use it in network serialization?
Requirement

We have a Client machine and a Server machine. On the Client machine we have a Product object having id of the Product is 1, name of the Product may be C Programming, price of the Product may be 500.0 and within it we have also stored the time of creation of the object i.e. 12:45:40. We wish to send this object to the Server. When we send it to the Server, we have to serialize it over the network. So stream of bytes will actually be send to the Server. On the Server there will have to be a code which will try to convert this data received in the form of 0s and 1s back into an object. Means here we would try to reconstruct the object from the bits that we received and then we will try to modify this object. We will try to change the price from 500.0 to 750.0. This price we would try to store in the same object. We can notice that when from the bits on the Server side the object got created, the time of creation of object is not same as the time of creation of the object on the client. On the client side it was 12:45:40 whereas on the Server when the object got reconstructed, it was 12:45:42. Means we will put an artificial delay of 2 seconds such that over a network when stream of bits go and gets recreated in the form of an object then there will be some time lack. So, 2nd object on the Server will be at 12:45:42. We multiply the object’s price with 1.5 to get 750. Then we try to send this object once again back to the client. When we do so, it will once again get converted into 0s and 1s. Once this is done, on the client side we will try to convert that 0s and 1s back into an object and once reconstructed into an object, we would print it just to verify whether the object has been created properly or not. There is a time delay between sending the stream of bytes over the network from the server to the client and then reconstructing that object on the client side. When it is reconstructed on the client side time is 12:45:44.

Product Class

Name of the package is networking and we import java.util.* and java.io.*. The Product class implements the Serializable interface. Within the class we have private int id, private String Name, private float price and transient private Date creationTime. We marked creationTime as transient such that it does not get serialized. In the constructor of the Product class, we setup the creationTime with the current time and print the message that >>Product.constructor1<<. We also have a 3-argument constructor here which receives id, name and price. Set these fields along with the creationTime. Again print the message that >>Product.constructor2<<.

Product Class Cond.

getPrice( ) function will return the current value of price and setPrice( ) function which allows to set the value of a price. Whatever value is passed to the setPrice( ) function is set to the object field called price. Then we have a toString( ) function which allows to convert the object into a string which we can then print using System.out.println( ). We will combine the four private fields. In creationTime, we are converting 1st it into a string using String.format( ) function. We are passing %1$tT to it. %1 is the positional parameter i.e. the 1st parameter is creationTime. $t indicates that we are interested in printing out the time. T indicates that we are interested only in time part of it and not int date and day. The whole string is combined and put in [ ] and that string is returned from the toString( ) function. Then we provide a readObject( ) function. When we say transient creationTime, we have to provide this readObject( ) function which is called during serialization. Whenever its time to serialize this, we say in.defaultReadObject( ) within this. It is responsible for reading out the id, name and price whereas date we setup ourselves by using this.creationTime = new Date( ).

Network Server

Package would be same as networking and we import java.net.* and java.io.*. We have a class SerializationServer and within this we initialize a value portNumber = 10000 and it is unchangeable. Whenever we try to communicate over a network that time we know the IPAddress and port number.
Both client and server make use of the same port number. Then we declare the constructor of the SerializationServer class. Within the constructor, we will create an object of the class ServerSocket and pass portNumber i.e. 10000 to this constructor. Socket is a bidirectional communication channel. Each socket will have two underlying streams within it; an input stream and an output stream. Using input stream we can read the data that is send to our machine from some remote machine whereas to the underlying output stream we can ourselves manage to send the data to the remote machine. When we say new ServerSocket( ), we try to create a socket on the server and whenever we create a socket that time not only we need the port number we also need the IP Address of the machine. But here we did not provide the IPAddress because the ServerSocket class takes the IPAddress of the Server machine i.e. the current machine and uses the port number that we pass to it and creates a ServerSocket. This socket is a listening socket. On the server side, we create a listening socket to which a client socket connects. Once the listening socket is created, we would call a function accept( ) within the ServerSocket class. accept( ) listens for a connection means it is going to wait for an incoming request that is going to come from client and once that request comes, the accept( ) function accepts that connection request and come out of a loop. So accept( ) is a blocking function means unless and until control comes back from accept( ) further execution cannot happen. accept( ) will create a new communication Socket on the server side and that communication socket’s address would be returned is collected in cSock variable. Once the communication socket is created, the listening socket is free to accept any other connection. Now we would setup ObjectInputStream is to null and ObjectOutputStream os to null. Within the try block we create a new ObjectInputStream object and provide cSock.getInputStream( ) to it. Then put the current execution thread to sleep for 2 seconds. This is to simulate the network delay. Then we read the object using is.readObject( ) and cast it to Product and store in Product p.

Server Continued
Then we print whatever Product had been received. When we do System.out.println( ), the toString( ) method of the Product class will be invoked. Then we manipulate the price by multiplying it with 1.5f. We use getPrice( ) to find the current price of the product and set the new price for that product using setPrice( ). Then we send this object back to the client. To be able to send it back to the client, we would use same OutputStream for the communication socket. Wrap it in a ObjectOutputStream object and then we say os.writeObject ( p ). Modified product is written to the underlying stream. In the finally block, we close the streams i.e. os.close( ), is.close( ), cSock.close( ) and lSock.close( ). All the streams are closed.

Network Client
We again begin with package networking and import java.net.* and java.io.*. Inside the class SerializationClient we initialize portNumber = 10000. In the constructor of the SerializationClient, first set the communication socket to null and ObjectOutputStream os = null as well as ObjectInputStream is = null. Within the try block, we create a new Socket. The new socket we are creating is a communication socket. So we are using a Socket class. We are using the IPAddress and a portNumber. In place of the IPAddress we are using local machine name which is always the localhost. Then create a new ObjectOutputStream object and wrap an output stream of the communication socket that we have created within that.

Client Continued
Then we create a new Product. After creating a new Product, we will print that out using System.out.println( ). Once again toString( ) method of the Product class would be invoked. Then we write object p to the ObjectOutputStream. When we do writeObject( ) the object would be streamed over the network. Then we would also try to create a new ObjectInputStream object which wraps around the communication socket’s input stream. Then put Thread.Sleep ( 2000 ). Then make a call to readObject( ) and object that is read is casted into Product and collected in p. Lastly print the
Product p. In the finally block, we will close all the streams by using if ( cSock != null ) then
cSock.close( ) and if ( os != null ) then os.close( ).

Multiple Units Of Execution

Suppose we browse the net while using yahoo.com then as the contents of page being downloaded,
we can use scroll bar to move the page up and down. Whenever we use Word while typing, in thread
we do tying and in another thread the spelling check and in the third thread grammar check is also
being done.

- Thread is a unit of execution.
- An application can contain more than one unit of execution. Every application can have at
  least one unit of execution.
- Once a thread is created within an application, it starts getting time slices.
- The amount of time slice allocated is a Thread Quantum.
- Scheduling is done on a per thread basis.
- Multithreading improves application's responsiveness.
- Simplify program organization. When copying of files is done, we see two folders and flying
  of files from one to another. This is a AVI file which is played. For that two different threads
  are launched within the copying application.

Multithreaded Driver Program

We have a class NetSerialTester. Within main( ), we declare variables serverT and clientT of Thread
type. Create a new Thread object and pass address of an object of the type ServerRunner to it. To a
Thread class’s constructor, we have to provide an address of an object which implements Runnable
interface. So ServerRunner class will have to implement Runnable interface. Then we call a start( )
method in it. The start( ) method will actually call the method in the ServerRunner class which will
actually launch the thread. Then sleep for next 400 milliseconds. Create a clientT thread reference
and using it launch the thread by first creating a Thread object, pass to it the address of the
ClientRunner object. ClientRunner will also have to implement the Runnable interface. Then
clientT.start( ). This time client thread will start. Then we will wait till the time both client and server
threads both do not die. It is important to call the join( ) method because join( ) will ensure that
application does not close before the server and client threads are over. So we say clientT.join( ) and
serverT.join( ). This ensures that main thread waits for client and server threads to terminate. When
we do Thread.sleep( ), we are waiting for the server thread to bind because in real life server and
client will run on different machines. But here we are trying to run it on same machine. So we use
Thread.sleep( ).

ServerRunner

ServerRunner class implements Runnable interface. This interface has only one method run( ).
Within the try block create a new SerializationServer object. If some exception occurs then we can
print it in the catch block.

ClientRunner

Similarly ClientRunner class implements Runnable interface. Within the run( ) method inside thirty
block, create a new SerializationClient object. If some exception occurs then we can print it in the
catch block.
In this lecture you will understand:

* What is zipping & unzipping?
* Program to
  * Zip files
  * List entries in zipped file
  * Unzip files
ZIP

• ZIP format is a data compression and archival format.
• A ZIP file contains one or more compressed files.
• The format was originally designed by Phil Katz for PKZIP compression and decompression utility.
• ZIP files generally use the file extensions ".zip" or ".ZIP".
• ZIP format compresses every file separately.
• This allows individual files to be retrieved.

Requirement

In whichever directory we want to write this program that directory itself we want to zip. If the name of the project we give as ZipUnzip then in folders like /build, /dist, /nbproject, /src, /test all these get created. In test folder there are few more files are present as shown in the slide. We want to zip all these folders and files. For that purpose, we want to define a function called zipAll( ). So the Current directory we would try to zip in one file called s.zip. Once we zipped this, we also want to show all these zip entries on the screen by using a function showZipEntries( ). It will simply pick up the entries present in the .zip file and list them out on the screen. Then we see how to make use of unzipAll( ) function which will unzip all the files and folders present in s.zip into some temp directory. We want to write zipAll( ), showZipEntries( ) and unzipAll( ) functions. All these functions are called from main( ).

Program

Name of the package is zipunzip and we import java.io.*, java.util.* and java.util.zip.*. Inside class ZipUnzip, declare variable static final int BUF_SIZE = 100. Within main( ), define ZipOutputStream zos. We make use of this by using new ZipOutputStream. Create a ZipOutputStream object and wrap inside it a new FileOutputStream object. Pass the name of the zip file inside which we wish to zip the current directory i.e. s.zip to the constructor of the FileOutputStream. Then we will call the zipAll( ) function which is a user-defined function and pass the directory which we wish to zip to it. . ( dot ) indicates the current directory. .. means the parent of the current directory. We wish to zip the current directory. So first create a file object by saying new File( ) and pass name of the current directory to it. We also pass the ZipOutputStream reference to the zipAll( ) function. Then we close the ZipOutputStream. Then we call another user-defined function showZipEntries( ) which shows all the entries present in the s.zip file which is created through the zipAll( ) function. Once all these entries have been listed out, we will then create a File object temp inside which we wish to unzip all the entries present in s.zip. Then check whether such a directory exists or not. If it does not exists then try to create that directory by calling a method mkdir( ). So a temp directory gets created in the current directory. Then we call unzipAll( ) function which is a user-defined function and pass source file s.zip to it. We also pass the destination folder where we want all the files and folders present in s.zip to get unzipped. To pass the target, we first create a new File object using new File ( "." ).getAbsolutePath( ). So the entire path will get created upto the current directory. At the end add /temp/.

zipAll( )

zipAll( ) function will zip all the files and folders present in the current directory. It receives File f and a ZipOutputStream zos. Within this function, we will call File class’s static method listFiles( ). This will return a list of files and it is collected in array of Files i.e. list. Then iterate through the list using for ( File ch : list ). One by one we will extract the file present in the list array. We will check
that each file we extract is a directory or not using if ( !ch.isDirectory( ) ). If it is not a directory then its time to zip a file. To carry that out, we will call a function addZipEntry( ). Pass ZipOutputStream that we had created to it and we say ch.getParent( ) + File.separator + ch.getName( ). ch.getParent( ) will give the parent of the current entry to that we will add File.separator which is usually a / and then attach the name of the entry using ch.getName( ). Means we are rebuilding the entire path. Then once again we iterate through the same list. Each time through this for loop we would check whether it’s a directory or not using if ( ch.isDirectory( ) ). If it’s a directory then we can recursively make a call to zipAll( ) function because within a directory there might be files present. Once again pass ch andzos to zipAll( ) function.

**addZipEntry( )**

The addZipEntry( ) function receives the ZipOutputStream zos as well as the String that represents the file that we wish to zip. Within this function, we first declare the BufferedInputStream reference fis. Then create a new BufferedInputStream object and pass an object of new FileInputStream to its constructor. While creating an object of the FileInputStream, pass the file that we wish to zip. Create a new ZipEntry object and pass the file that we wish to zip to it. Before passing it to ZipEntry, if \ present in the entire path then replace that \ with / and then pass it to ZipEntry. We can print the name on the screen by using ze.getName( ). Then we call a function putNextEntry( ) which is a member function of the ZipOutputStream class. This function begins writing a new ZIP file entry and also positions stream to start of entry data. Create a byte[ ] data of size BUF_SIZE. Now we can start reading from the source file by saying fis.read( ). fis.read( ) will always able to return the number of bytes it is able to read successfully which is collected in count. Whatever number of bytes we have read, those numbers of bytes we have to write using zos.write( ) i.e. count number of bytes are written to zos.

**showZipEntries( )**

Within this function create a new ZipFile object. Through name s.zip is passed. Address of the ZipFile object is collected in a reference zf. Now we want the list of entries present in the s.zip. For that we have to make use of a variable e which is of the type Enumeration <ZipEntry>. It indicates the Enumeration is a generic interface or a templated interface. Then iterate through this by calling a function called entries( ). entries( ) is a function of the ZipFile class. It returns the reference of the enumerator and once we get that reference, we can iterate through these different entries. Using e.hasMoreElements( ), go on running the loop till the time e.hasMoreElements( ) return a true. To get the element we have to call a function called nextElement( ). hasMoreElements( ) is called to check whether there are more elements or not. If there are elements then retrieve those zip entries objects reference using nextElement( ). The ZipEntry object’s reference that we get is checked whether it is a directory or not using if ( !ze.isDirectory( ) ). If it is not a directory then we print that entry. While printing that entry, we want to print three things; name of the entry, compressed size of the file and original size of the file. So we are using %1$s %2$d ( %3$d ). $s indicates that 1st entry is a string and $d indicates that 2nd and 3rd are integers. The original size of the file is printed within ( ).

**unzipAll( )**

Within this function, we have source and destination strings. Source is s.zip which we want to unzip now and destination is temp directory present in the current directory. Create a ZipInputStream reference and for that we create an object of ZipInputStream and wrap that with FileInputStream. src is submitted to FileInputStream object constructor because we want to read from this file. We define variable ZipEntry e and using e we try to call a function getNextEntry( ). Using this function, we get next entry from s.zip. So long as there are entries present in s.zip, we should try to create a new File object for each entry that we extract. Name of the entry is obtained by using e.getName( ). Create a File object for the extracted entry. Then check whether that entry is a directory or not. If it is a directory then check out whether f.exists( ) is true or not. If it does not exist then we create nested
directories if required. So we use function f.mkdirs( ). So, nested directories also get created. Then if
it is not a directory then we go to the else block. In this block, we get the size of a file. If its a file
then we need to unzip that file. Maximum number of bytes that we read is stored in max variable
initiated using e.getSize( ). Then we create a new File object. dest is a temp directory. Its entire path
is available. At the end of it we would add a / and add f.getParent( ) to it i.e. for the current file entry,
the parent is added.

Slide 10

Once the parent object has been created then we need to check out whether such an entry is already
there or not. If such an entry does not exists then we create nested directories by using parent.mkdirs( )
then we create FileOutputStream reference and using dest, / and name of the entry, create
FileOutputStream object. Once the FileOutputStream object is created, we carry out reading and for
that we first create a byte array data. We run a loop from while ( size <= max ). Each time through
the loop, we should read the content of the file into the buffer and once we had read it into the buffer
then we check that whatever is returned by read( ) is -1 or not. If it is -1 then we break the loop.
Otherwise we should write the data to the FileOutputStream. Then we increment the size by size +=
len. Then close the FileOutputStream.
Multithreading And Synchronization

In this lecture you will understand:

* What is multithreading?
* How to get properties of current thread?
* How to launch multiple threads?
* Pros and cons of different launching methods
* Need for synchronization
* How to achieve it programmatically?
Current Thread

Multithreading is the ability of a program to run several units of execution simultaneously. A thread scheduler, which is a part of JVM manages all the threads running. A thread scheduler schedules one thread after the other. Scheduling threads means giving time slices to each of the threads that are running in the JVM. Scheduling is done on a Round Robin basis, which means if there are three threads that are running in a memory then the thread scheduler will allot $\frac{1}{3}$rd of time to each of these three threads unless some thread has a higher priority than the other thread. If a thread has a higher priority, then it will get more time slots as compared to the lower priority thread. Name of the package is mainthread and class name is MainThread. Within the class we have main() and inside main(), we would try to access the properties of the current thread. By default, the program runs in a thread which is commonly known as a main thread. If we want to get the properties of the main thread, then we have to make use of a static method inside the Thread class. Once we call the current threads static method, it returns the reference of the current thread object which is collected in t. Once we get that reference, we can print it out. While printing the object, the toString() method of the Thread object will get called. The properties are printed as main 5 main. The default name given to the main thread is main. This is different than the name of the function. 5 indicates that the thread has normal priority. We can make several threads belong to a particular group. So name of the thread, priority and group name of the thread is printed out. We can give a new name to the thread and for that we have to call a setName() function of the Thread class. We are proposing to give mythread as a name to the current thread. After the name of the thread had changed, we once again print the contents of t and it is mythread 5 main. t.getName() gives the name of the thread which is stored in String s and we print that name which results in mythread.

Launching A Thread

Suppose we have a package sample which contains class Sample and within it we have main(). Within main(), we run a loop 10 times and print out a message Main thread. We would create a new class Ex which is extended from the super class Thread. Ex class contains a constructor. So within main(), we say Ex t = new Ex(). When we say Ex t = new Ex(), the object of the Ex class would get created and its constructor would be invoked. Ex is extended from Thread class hence base class’s constructor would bound to get called. When we invoke the constructor of the Thread class, it stores the address of the Ex object in one of its private variable. The constructor also registers the thread with JVM. Now if we say t.start(), so start() method is searched in the Ex class. If it is not present in the Ex class then control passes to the base class i.e. Thread. Thread class contains a start() function. The moment the start() gets called, it signals the thread scheduler that our thread is now ready to run. So we are signaling through the start method that the JVM’s scheduler can now start scheduling this new thread on a Round Robin basis. So whenever its time for the scheduler to give time slot to our thread the new thread that we have launched it would try to call the run() function. The run() function, we have to provide in a Ex class. We call the start() function. start() of the Thread class’s member function would get called and the start() function would instruct the JVM that the thread is ready to run and schedule time slot for it. Whenever the thread scheduler schedules a time slot for our thread that we have launched, the run() function within our thread will get called. Within the run() function, we print out some message New thread 10 times. Output we might get is shown in the slide.

Multiple Threads

Within the Sample class, inside main(), we would try to launch at least three threads apart from the main thread to which our main() belongs. Within main(), we have a for loop which prints Main 10 times. Ex class extends the Thread class. We create a new Ex object and call start() method using reference of Ex object i.e. t1.start(). Then we give the name to the thread by using t1.setName ("First"). Then we launch one more thread by saying t2.start() and give name to it by using t2.setName ("Second"). Similarly we would create one more object and give name to it as Third. We
had a run( ) function within the Ex class and inside the run( ) function, we would try to get the current thread object because run( ) gets called for t1, t2 as well as t3. We first get the object of the current thread and then try to extract its name by using t.getName( ). Collect it in String s and print that string within a for loop 10 times. Same run( ) function can get executed for different threads. We want that the main thread would not die when other threads are running. Because if main thread die then other threads would also die since all other threads are launched from the main thread. So we have to make a call to the join( ) method on each one of this thread. Control will wait within the join( ) method unless and until that particular thread dies. Output is shown in the slide.

One More Way

We have class Sample and inside main( ) of the class, we would try to create a new Thread object. As we construct this Thread object, we will pass an Ex object’s reference to its constructor and for that we would have to create a new Ex object. We would pass One to the constructor of the Ex object. We have to define Ex and while defining it we have to make sure that Ex implements a Runnable interface. If we want that the program would have an ability to launch a new thread then whenever we create a Thread object, pass address of another object whose class implements a Runnable interface to it. A run( ) method is present within the Runnable interface. In the constructor of the Ex class, we receive the string in s and set up in a private String str. We do t.start( ) in main( ) which will invoke start( ) method of the Thread class. start( ) method inform the JVM scheduler that the thread is ready to run now, you can schedule time slot for it. Whenever it schedules the time slot and that particular thread’s term comes, during that time slot, the run( ) method within Ex class would get called. Inside run( ) method, we would run a loop 10 times and each time through the loop, we would print str. Then in main( ) we can print out a message Main thread 10 times.

Multiple Thread Using Runnable

We provide the Sample class. Ex class would remain same i.e. it would contain the constructor and the run( ) method. Within main( ), we would create a new Thread object and to the Thread object, we will pass the address of the new Ex object and pass parameter as First to the constructor of the Ex class. Then we would do t1.start( ). Means we are trying to store information of the Ex object within the Thread object. Similarly we say Ex t2 = new Thread ( new Ex ( "Second" ) ) and launch this thread as well by saying t2.start( ). Same can be done for the third thread i.e. Ex t3 = new Thread ( new Ex ( "Third" ) ) and launch this thread also by using t3.start( ). Then within main( ), we run a loop and print Main thread 10 times. When we run the program and kept Ex class same then the four messages First, Second, Third and Main thread will get completely mix with one another telling that we have an ability to launch multiple threads not only by extending the Ex class deriving it from the Thread class but also by creating Ex class which implements the Runnable interface.

Multithreading Tips

- There are two methods to launch threads:
  - Creating a class that extends Thread class
  - Defining a class which implements Runnable interface

- In both cases a thread object must be created
  - object ex gets known to Thread

- start( ) is a Thread member function
  - It registers ex object with scheduler.

- run( ) must be within our class
  - It is called by scheduler.
• When we use extends Thread that time we cannot use in Multiple Inheritance situation.
• extends Thread - Thread related functions can be overridden.

**Requirement**

From the Main Thread we intend to launch new threads; Thread1 and Thread2. In both the cases we wish to call a method dowork( ). Within dowork( ), we call two more methods dowork1( ) and dowork2( ). When dowork( ) is called, unless and until dowork1( ) and dowork2( ) has completed in that particular thread may be Thread1 or Thread2, till the time Thread1 working with dowork( ) method if Thread2 gets time slot it should be made to wait. Once Thread1’s job of dowork( ) is over then only Thread2 should really made a call to dowork( ) and get its job done. We are trying to synchronize the activities of Thread1 and Thread2.

**Program**

Name of the package is synctest and class name is SyncTest. Within the class we have main( ) and inside main( ), we would create a new Work object. Work is a user-defined class. Then we would try to launch two new threads. We are not using an extending thread method for launching a thread. We are trying to make use of Runnable interface implementation within the class. So we would try to store new ThreadRunner object within the new Thread object. We passed a Work object that we had created to the ThreadRunner object. Same is done for the 2nd thread. Then we would start both the threads by using t1.start( ) and t2.start( ). Unless and until t1 and t2 die till that time main thread should continue to run. So we call join( ) on t1 and t2 i.e. t1.join( ) and t2.join( ).

**Work Class**

Within the Work class we have int count = 0. Then we define the doWork( ) method. doWork( ) prints current value of count along with the name of the current thread and a message doWork. From doWork( ) method, we call doWork1( ) and doWork2( ) methods. Within doWork1( ) method, we print current value of count, name of current thread and along with that we print the name of the method doWork1. Then we would go for sleep for next 200 milliseconds. Within doWork2( ) also we print value of count along with that we print name of current thread and a method name doWork2. We increment the count in the doWork2( ) method. When doWork( ) or doWork1( ) is in progress that time value of count is 0. Then in doWork2( ), we would go for sleep for next 100 milliseconds.

**ThreadRunner**

The ThreadRunner class implements the Runnable interface. Inside it we have a private Work obj. Whenever the ThreadRunner object is created, it will store whatever it receives in Work o into obj. Within the run( ) method, we run a loop two times and each time through the loop, within the try block, we call a doWork( ) method. If any exception is thrown, we should catch it. When we catch the exception, we should print the message associated with it using System.err.println( ). When we execute the program, the messages that get printed clearly show that all the threads got time slot and all of them got mixed up as shown in the slide. But we do not want such an output. We want that the activities of these threads should be synchronized properly.

**Solution**

For that we need to attach a keyword synchronized before public void doWork( ). doWork( ) remains same. Now when we execute this program, this time the output come exactly as per our requirement. The output is shown in the slide. For all doWork( ), doWork1( ) and doWork2( ) the count is printed as 0. For the second time count is printed as 1 for all doWork( ), doWork1( ) and doWork2( ) and next it is printed as 2 and finally as 3. First 6 messages are for Thread0 and next 6 messages are for Thread1.
In this lecture you will understand:

- Where are web services?
- What is WSDL?
- Accessing web services in NetBeans
- Using web services programmatically
- Crux of Web Services
Create New Project

Click on New Project | Java | Java Application. Then click on Next.

Slide 4

The screen gets displayed when we click on Next. We are required to give name of the Project, Main class for that Project and the package to which it belongs. We are going to create a Project name as currencyclient. Click on Finish.

Where Is The Web Service?

We have written this Web service in .Net and we want to consume it through a Java program. With the advent of Web Services, two cross platform communication is possible. Cross platform means a Java program should be able to communicate with a C# or a VB.Net program or a VB.Net or C# program should be able to communicate with a Java program. So we create a currency web service in .Net and shown to you how to consume it through a Java program. Currency service is a web service and its present in http://quest.ksetindia.com/CurrencyService/CurrencyService.asmx. asmx is an extension used for web service in .Net. If we type it in the browser and hit the Enter key, the CurrencyService page is shown. When it is shown, it clearly tells that there are five different methods present in CurrencyService. One of the methods is GetCountryCurrencies( ). Then we have GetCountryNames( ), GetCurrency( ), GetRate( ) and GetRateByCountry( ). GetCountryCurrencies( ) means for different countries in the world what the currencies that each one of the country uses. A list of that would be returned when we call GetCountryCurrencies( ). When we say GetCountryNames( ), we get names of the countries that exists in the world. When we say GetCurrency( ), we will be required to give the name of the currency and equivalent currency in which we want the equivalent of it. For example, if we wish to find out 1 US Dollar = how many number of Indian rupees? We have to pass USD and INR as the currencies to the GetRate( ) and we get for 1 USD how many Indian rupees are there. Rate given is different on different Dates. So, five different methods are present in a CurrencyService web service. If we want to know the description of these methods then we click on service Description.

WSDL

When we click on Description, we get the description of the CurrencyService in a Web Service Description Language i.e. WSDL. WSDL explains the web service in the form of XML. If we go through the XML, we will find that it indicates what different methods are present inside the CurrencyService, type parameters we need to pass to each of these methods and return types. All of that are written in the form of XML. XML is use to describe many things in a orderly fashion. The place where WSDL file is present that URL is shown in the slide. We need this URL when we try to consume or use this web service.

Add Web Service Client

We right click on the project currencyclient. Select New from the menu that pops up. From this menu select Web Service Client…

Add WSDL URL

When we do this, another dialog shown in the slide will appear. Within this dialog, we are required to give the WSDL URL. We have to provide http://quest.ksetindia.com/CurrencyService/CurrencyService.asmx?WSDL URL. Give the package name as currencyclient and the Client Style is selected as JAX-WS Style. JAX-WS means Java API for XML Web Services. Then click on Finish.
Slide 9
When we click on Finish, lots of compilation will happen within the window shown at the bottom. Behind the screen, new classes have been created for us which will allow to have an access to this web service. The end result of this is shown in the currencyclient project. Click on Files menu.

Slide 10
When we click on Files menu, the tree expands and we see the lot of files created and all these files are created the moment we clicked on Finish. All the classes present in these files are created automatically for us. Once the classes had been created, we need to add code within main( ) to consume the web service.

Add Code
We say CurrencyService srv = new CurrencyService( ). The CurrencyService class has been created on our machine within our project and it is derived from javax.xml.ws.Service. So from the Service class the CurrencyService class had been inherited. This code is commonly known as a stub code. Actual web service is present at quest.ksetindia.com. Since we wish to use that service, some code that is generated on the client that is known as the stub code. There is an interface CurrencyServiceSoap which is implemented in one of the wrapper classes. For us to be able to use the methods of this interface, we say srv.getCurrencyServiceSoap( ). getCurrencyServiceSoap( ) method is present in the CurrencyService class. A CurrencyServiceSoap objects reference is returned which is collected in sa. Once it is available to us, we can now try to call the methods inside the CurrencyService.

1. Get Conversion Rate
We say sa.getRate( ). We pass USD and INR to the getRate( ) method. Means we wish to find out 1USD is equal to how many Indian national Rupees. The value that would be returned to us is collected in float and we would print that out using System.out.println( ). If we execute the program, in the output we get 1 US Dollar = 39.325 Indian Rupees. getRate( ) receives a from String and a to String. Means from USD to Indian Rupees we want to carry out the conversion. This is a live web service. So, on different days we execute this program, we get different rates i.e. current rate is displayed.

2. Countries And Currencies
We can also get the list of the countries and the currencies that are prevalent in those countries. For that in main( ), we need to add suitable code. But before that we need to import java.util.List. We are adding the code shown in the block. We call a function getCountryCurrencies( ). Whatever it returns, it is collected in List. A list of countries along with their currencies is returned to us. Each currency internationally is represented by three letter word. For example, for Indian Rupees we have INR, for American Dollar we have USD, etc. We get not only the name of the currency and the country, we will also get the way it is represented using a three letter word. We are iterating through that list using a for loop. Each time through the loop we will print out c.country, c.currency and c.symbol i.e. the name of the country, currency that it uses and the symbol that is used for that currency.

Output…
When we execute the program, we would see that in the output we get list of countries, the currency that is prevalent for each of these countries and lastly we have the names that are acceptable in the International market for different currencies. For example, for America its USD, etc. as shown in the slide.
Other Methods…

There are three more methods. How to make use of the two of them is shown in the slide. For example, we have a call to a method called getRateByCountry( ). In this we pass name of the country instead of name of the currency. When we pass Australia and India, we come to know 1 Australian Dollar is equal to how many Indian Rupees. The answer likely to get is shown in the slide. We might get different answer because it's a live web service. We also have a method getCurrency( ). Means we pass the name of the country and we will get to know what is the currency that is used in this country. For example, we wish to find out for Hungary then say sa.getCurrency ( "Hungary" ) and its currency is HUF that will get printed out. Similarly we can call sa.getCountries( ) method and get a list of countries.

What Are Web Services

Web Services are services that are offered on the Internet which anybody can consume. It does not matter whether web service was written in Java, C# or VB.Net. If it is written in Java then we can consume it is Java program. We can also consume the exactly same web service, if we write a client in J2ME. Once we create a web service, we can consume it from multiple clients. Web service is available all the time. Web service can be used for HTTP / HTTPS for data transfer. XML format has a universal appeal whenever its time to represent any idea. So web Service description if it is given in XML in that case it is very easy to read XML, parse it and understand it. Lots of XML parsers are available. Through XML, we cannot make calls to functions like getRate( ), getRateByCountry( ), etc. For that we need SOAP envelope. SOAP is a Simple Object Access Protocol. Using this protocol, we can actually make calls to functions. When this protocol indicates that such method is called, to indicate that once again XML is used. We have two sides to a web service; one is service requestor or a consumer and another is a service provider. From the consumer a SOAP envelope has to be send to the Service Provider. Through that SOAP envelope the call would be made to the service, whatever is the returned value that returned value once again would be put into the SOAP envelope and it will be returned to the client who is trying to consume that web service. All of these are text based. SOAP is a message envelope and its basic job is to map Java types to XML before sending it over the network and vice versa. There are two ways; one is JAX-WS stands for Java API for XML Web Services either that can be used. Now it is super cited by JAX-RPC stands for Java API for XML RPC. We are consumers so we can use any one of them.
Using XML

In this lecture you will understand:

- Different data usage scenarios
- What is XML?
- How to create XML document, DTD, SAX handler classes?
- How to parse a XML file using SAX handler classes?
Data Usage Scenarios

The four major scenarios are as follows. One situation could be, we are using an application which is trying to read data or write data to an Oracle database. Another application might be reading data or writing data from an Indexed sequential database like a COBOL database. If these two applications who are handling data in two different ways if they want to carry out communication amongst one another then we need some common ground according to which these applications can communicate with one another indicating that, that is how we represent a data. If one application is to send data to another, second application to send data to the first one then what is the common ground that can be used for both the applications to understand data without any confusion. This is one scenario where we need to exchange data between different applications. In another application a Solaris application using FTP uploads certain files on a Web Server. Then a Web Client that has been written either in Java or in .Net or in any such language or technology tries to send request and get response form a Web Server. This Web Server is connected to a database. How the data retrieval would happen when the Web Client makes a request and how that data would be send to the Web Client such that the client can then make use of that data. This is another scenario where the data will have to be used in such a manner that the Web Server and the Web Client agree on some ways of representing the data. Third scenario could be where we have centralized backup manager which is going to back up data every now and then for example, thrice everyday from different machines which are connected to the backup manager in a network. Backup agent will have to some how the other communicates to the backup manager. That data needs to be backed up from this particular agent. So again data exchange will have to be happen between backup agent and backup manager. Backup agent will have to communicate that these are the files that need to be backed up and then the backup manager will have to actually carry out the backup and then communicate through the agent that the backup is over. Once again its time to communicate between backup agent and backup manager there ah has to be a common ground agreed upon by both the parties for this communication to happen. The fourth scenario could be we may have been created a .Net application, a Java application and an application created in VC++ using COM technology. .Net, Java and COM are competing technologies which will allow to create distributed applications. If three different companies created three different applications and in a business-to-business scenarios applications has to communicate with one another then we can realize that there has to be a common agreed upon way according to which data exchange can happen between these three distributed applications which are written using different languages, which implements different technologies. For such scenarios there has to be a common ground.

Moral

- There is a need to have an easy way to describe and exchange data between two different applications.
- There is a need for an easy way to make function calls in Distributed applications.
- All needs are satisfied using XML - Extended Markup Language.
- XML contains user defined tags whereas HTML tags are Standard tags.
- The slide shows the example of XML document. The document describes a data about a book. Name of the book is VC++ COM and Beyond which has number of pages as 460. Name of another book is Effective COM and has 510 pages. books, book, title, etc. are XML tags.

XML Reader – Steps Involved
We want to write a XML Reader program which will read a XML document, parse it properly and based on whatever is the tag present in that document perform some action. Steps for that are as follows:

- Create new Java Application.
  - Project name – Contacts
  - Package - contacts
  - Main Class - XmlContacts
- Create new XML file – Contacts.xml.
- Generate DTD file – Contacts.dtd. DTD stands for Data Type Definition.
- Add its reference in Contacts.xml.
- Generate SAX document handler class. SAX stands for Simple API for XML.
- Add Contact class.
- Modify SAX handler class generated by NetBeans.
- Invoke SAX parser and print contacts.

**Create Contacts.xml**

XmlContacts is the name of Main Class. Contacts is the Project name and contacts is the package name. Now we wish to addContacts.xml file to this project. Go to the Project tab, right click the Contacts Project. A menu will pop up. Select New from the menu. After that one more menu would pop up and select XML Document… from here.

**Give Filename**

After that we would be asked to give name for the XML document. So we give the name as Contacts. So Contacts.xml would now get created. Then click on Next.

**Select XML Document Type**

When we click on Next, we get the XML document type that we want to create. A dialog would pop up indicating the type of document we wish to create. There are three options available. We will choose Well-formed Document i.e. the XML document which is properly formed. It is not constrained by either DTD or XML-Schema. On the basis of the XML document we create the DTD document. Then click on Finish.

**Result**

When we do so, we will see the actual XML file created as shown in the slide. The only tag shown is the \texttt{<root>} and \texttt{</root>}. / indicates the end of that tag. \texttt{<root>} is the starting and \texttt{</root>} is the ending of the tag. We can put any number of tags that we want within this.

**Modify Contacts.xml**

During modification, we propose to create four different tags; Contacts, Contact, Name and Telephone. We provide name as Sachin Tendulkar whereas Telephone number is given as +91 22 45889900. Once we modified Contacts.xml, save the file.

**Create DTD – Contacts.dtd**
Once that is done, we want to create DTD for the XML document. Go to Contacts.xml, right click on it. A menu will pop up, select Generate DTD… from the menu that pops up. DTD generator cannot imagine what kind of Data Type Definition it is supposed to build. For it to be able to understand that we gave it a sample XML, Contacts, Contact and Telephone. On the basis of this, it will generate the DTD document. When we select Generate DTD…, it will ask for the name of the document. Say Contacts.dtd we wish to create. Then click on OK. When we do so, the DTD document will be created.

Copy DTD Reference

From this DTD document, we need to copy the DTD reference present in this DTD document. Arrow in the slide indicates the DTD reference and it is written within < !DOCTYPE Contacts SYSTEM "Contacts.dtd" >. This Contacts.dtd’s reference we have to add to the XML document. So we are copying this line and go to the XML file. Contacts within the DOCTYPE is a symbolic name given to the Contacts.dtd.

Paste & Modify

The line we copied is pasted in the XML file. Go to the top in the XML file and paste the line copied from the dtd document. We need to modify the line copied and pasted i.e. < !DOCTYPE Contacts SYSTEM "src/contacts/Contacts.dtd" >. We have to give the complete path here. So while pasting the reference we also have to modify it suitably.

Invoke SAX Document Handler Wizard

SAX Document Handler Wizard allows to create the Handler class. Go to Contacts.dtd, right click on that, a menu will pops up. Select SAX Document Handler Wizard… from the menu that pops up. Once we select SAX Document Handler Wizard, we are required to go through four different steps to create the Document Handler class.

Step 1 – API Version

In this step, we are required to provide the version for JAXP and SAX Parser. JAXP is a superset of SAX Parser. JAXP represents whole set of classes that are really associated with everything related with the XML technology. Out of which we are going to generate a class for only SAX Parser. So we select SAX Parser Version number as 2.0. Then click on Next.

Step 2 – Element Mappings

In this step, it will ask to provide mappings for different elements. As shown in the slide, in the table, 1st column indicates Name, telephone, Contact and Contacts. It suggests the Handler Type and the Handler Method. Name and Telephone is described as Data whereas Contact and Contacts are described as Container. Contact is a Container for Name and Telephone. Contacts is a super container for several such Contact. It suggests the name of the Handler Method that is created in the SAX class. We keep names of these handlers as they are. Then click on Next.

Step 3 – Data Converters

In this step, it will ask that do we want any data conversion to happen whenever these handlers get called. Since we do not want any data conversion from string to some int, Date, boolean. We can see that as the XML document gets parsed each tag that it comes across is going to take to a particular handler to process that tag. While doing so, we might want to convert the String to an int or String to a Date because ultimately a String is going to be read from the XML document which is a text document. If we want that these Strings to be converted into some meaningful data type like int, boolean, etc. in that case we need to provide data converters for it. But we are going to read only
Strings i.e. Name and Telephone. So we do not need any data converters for this project. So we click on Next.

Step 4 – Output Filenames

After that it will suggest to give the names of the files where these handler classes interface and the handler class’s implementation is going to get stored. So ContactsHandler is the interface whereas ContactsHandlerImpl is the actual implementation of that interface. Then we click on Finish. Now the Handler class would get created.

Create Contact class

Package name is contacts and the class name is Contact. Within the class we have private String name and private String telephone. Then we provide Contact() zero-argument constructor which is kept as empty. Then we provide a getName() function which will return the name. We will also provide a setName() function which will receive a String and setup in name. Similarly we have getTel() function which returns the telephone and setTel() which receives String t and sets the value of t in telephone. We also provide a toString() function such that if somebody tries to print a Contact object this toString() function would be invoked and it would return the telephone and name written within a pair of < and >.

Modify ContactsHandlerImpl

We import org.xml.sax.* inside which all SAX related classes are present. We also import java.util.ArrayList. ContactsHandlerImpl has already been created by us through the four steps. This class implements the interface ContactsHandler. Within this readymade methods have also been created for us. We need to modify these methods. We will provide a list object of the type ArrayList and the ArrayList is going to maintain a list of Contact. So we create a list from generic ArrayList class. We also say Contact current = null. There is start_Contacts() method. When we start reading XML file, this start_Contacts() would get called as soon as the Contacts tag is encountered. final Attributes meta indicates that this particular tag, if it has attributes associated with it then those will be available to us in variable meta. Inside this method we would try to create a new ArrayList object i.e. list = new ArrayList<Contact>(). Then we also have start_Contact() which is going to get called whenever the Contact tag is encountered. Within Contacts, we had a Contact tag. At that time start_Contact() is invoked. Inside this we create a new Contact object i.e. current = new Contact().

Modification Continued

As soon as Name is encountered handle_Name() is going to get called. When this is called, the string which is obtained i.e. name is setup by saying current.setName(data). Then we have handle_Telephone() method which is invoked when Telephone tag is encountered. Within this, we say current.setTel(data) which setup telephone number in the private variable. Following that we have end_Contact() method which is going to get called whenever we encounter a </Contact>. When we encounter this method, we add the current object to the ArrayList i.e. list.add(current). When we reach end of Contacts we have nothing to do. So we had not done anything in end_Contacts() function.

Modification Continued

Within the getList() method, we are going to return the ArrayList list. Then there is also a finalize() method and we are not going to do anything within this. This is auto-generated by NetBeans within this handler class.

XML Reader
Package name is contacts and we import java.io.FileReader and import org.xml.sax.InputSource. We had XmlContacts class. Within main(), we declare a reference handler of ContactsHandlerImpl. We also have ContactsParser p. We first create an object of ContactsHandlerImpl, collect the reference in handler variable. Then create a new ContactsParser object and pass the handler to it. Its reference is collected in p. Then we say p.parse ( new InputSource ( new FileReader ("src/contacts/Contacts.xml") ) ). Inside this we created a new InputSource object which wraps around a new FileReader object. We pass the name of the XML file to the FileReader object. Then we run a for loop. handler.getList() returns the ArrayList that we had created within the implementation class ContactsHandlerImpl. We can iterate through the list and each time through the loop we can print each Contact using System.out.println ( c )The toString() method would be called and within < and > the data is printed.
Reflection And Annotations

In this lecture you will understand:

* What are annotations?
* Different scenarios in which it can be used
* What is reflection?
* What does it facilitate?
* Writing a Test harness using Annotations & Reflection
Annotations

- Annotations are more mature comments.
- They provide data about a program that is not part of the program itself.
- Annotations can be applied to classes, fields, methods and packages.
- 3 uses of annotations:
  - We can think of using annotations during compilation. Usage of annotations during compilation is to detect errors. It will also allow to suppress the warnings.
  - We can use annotations during deployment process. Annotations can be used during deployment to generate code and to create XML files.
  - We can use annotations during execution process. We can use annotations during execution to test the program.

Error Detection

Suppose we have a class base and inside the base class we have a public void fun( ) function which receives a String and an int and print that on the screen. From the base class, we propose to derive a new class der. Within the der class, we propose to override the function fun( ) which in the base class. To do so, we put an annotation @override. Then we can follow it with actual function fun( ). Annotations always begins with the @ symbol. Once we marked the fun( ) function with @override annotation, now an error would be reported because we are saying that fun( ) is being overridden from the base class but the parameters are not matching with the parameter of fun( ) in the base class. fun( ) in base class is a 2-argument fun( ) whereas fun( ) in the der class is a 1-argument fun( ). If we say that these are overridden functions, the signatures will have to match. If the signature do not match then the compiler would report an error.

Suppress Warning

Suppose we have a package sample. Inside the package we have a Sample class and inside this class, we have main( ). Within main( ), we are trying to create new Ex object e and using that we call a function fun( ) within the Ex class i.e. e.fun( ). Class Ex and within this class we have an annotation called @Deprecated. This is a standard annotation. Deprecated means no longer used. Suppose we have a fun( ) function which displays a message Hello and it is marked as Deprecated, when we try to mark it as Deprecated and we also try to use it that time a warning would occur saying that this is marked as Deprecated; we are not suppose to use it. If we wish to suppress the warning that comes then we can do so. @SuppressWarnings( value = "deprecation" ) is used to suppress the warning. An annotation can also have a value. value indicates which value we want to suppress. The way there is a deprecation warning that we can suppress; there is also an unchecked function that might be there, that warning we can also suppress. For that we say @SuppressWarnings( value = "unchecked" ). In this way we can suppress the warnings.

Annotation With Multiple Elements

If we want to declare an annotation which is a user-defined annotation then we have to say @interface. So @interface Author is a user-defined annotation that we want to create. We can have different elements within this annotation. For example, one element can be name of the company which is a String. So we say, String Company( ). This is syntax for that. So it’s like an interface having different methods. Company can be thought of as a method. We can also give default value for this String which is "KSET Pvt Ltd" using default "KSET Pvt Ltd". Then we have String Coder( ) is used for a person who is writing the code for this program. String Reviewer( ) is used to put the
name of the reviewer. String[] Testers( ) is used to store the names of the testers who will test this program and then we have int VersionNumber with default value 1.0. So the way we declare an interface on similar line we can declare an annotation. Once the annotation is created, we might want that these values present in the annotations should be given at the beginning of each class. So we can apply these annotations to a class. Once we create the annotation, we can indicate that these annotations should be applied to what. For that we are using an enum known as ElementType.METHOD and we are also using an enum ElementType.CLASS. @Target is an annotation but it is a standard annotation. We are using @Target and then using @interface Author, indicates that this can be applied to a method as well as to a class. However it cannot be applied to a field within a class. If we want that this should also be applicable to a field then we should mention ElementType.FIELD in the @Target annotation. We can also indicate that this annotation can be available at what particular stage. To indicate that we have to declare its retention policy and for that we use @Retention annotation. We pass to it RetentionPolicy.RUNTIME indicating that this annotation author should be available even during the time of executing the program. There can also be a retention which is available at source level or at the class level. Class Cricket is shown in the slide. We can use the user-defined annotation as shown in the slide. Before class Cricket we say @Author ( Coder = "Rahul", Reviewer = "Ranjit", Testers = { "A", "B" } ). So this annotation is applied to the Cricket class. We had not given the company name, so it is taken as default.

**Reflection**

Reflection is a mechanism which allows to get the information or use the information. Using reflection, we can get information about an object, about a class or about an interface. We can come to know that this particular object has been created from which particular class. If we know the class, we can find the modifiers used by the class. Fields that this class have, different constructors defined within this class, different methods declared within the class, the class from which this class has been derived, parameters we need to pass the methods that we intend to call, etc. all these information we can get about a class using reflection. If we know the interface, we can figure out different constants used in it, methods present inside the interface using reflection. So all the information we can obtain about a class, an object and an interface using reflection API. We can use this information to create an object. Using the set or get method present inside the object, we can modify the object as well. We may call some other methods that may have been implemented on that object through the program. All this can be done at runtime in a programmatic fashion.

**Using @ At Runtime**

Name of the package is annotationsreflection and we import java.lang.annotation.* and java.lang.reflect.*. Then we declare annotation. First we mention the Retention policy for it using @Retention ( RetentionPolicy.RUNTIME ). We also want that this annotation can be applied only to a method so we say @Target ( ElementType.METHOD ). Then we declare a user-defined annotation by @interface TestMethod {  }. We do not want any elements within this annotation. Then we declare a class ClassUnderTest. Inside this class we have different testcase methods; testcase1( ), testcase2( ) and testcase3( ). To some methods we apply the @TestMethod annotation. In testcase1( ) method, we print name of the method indicating that we are in testcase1. Then we have testcase2( ) method and applied @TestMethod to it. Inside this method we display testcase2 and we throw a RuntimeException within this method. This exception is caught be a class which makes use of the class ClassUnderTest.

**Program Continued**

Then we have testcase3( ) method. We are not applying the annotation to this method. Whichever testcase we actually wish to check to that alone we should apply the annotation @TestMethod. Within this method, we display a message testcase3.
Test The Class

We have a class TestHarness within which we have main( ). Inside main( ), we propose to have two variables passed and failed with initial value as 0. Then we create a String className having value annotationsreflection.ClassUnderTest where annotationsreflection is a namespace and inside it we have class ClassUnderTest. So we store packagename.className in the String className. Then we would run a for loop. We say Class.forName ( className ). Class is a readymade class. For every single class that runs in a Java Runtime Environment, a Class object is maintained for it. So for ClassUnderTest, a Class would also be maintained. We are trying to use Class object which contains information about that particular class. When at execution time this TestHarness main( ) would get executed, we are going to find out, what Class object for such a class whose name is annotationsreflection.ClassUnderTest. Once we get the Class object for this particular class, we would try to call a method on it called getMethods( ). Then we get different methods present in the class ClassUnderTest. Once we get different methods, we will feed it to the Method m. Means we iterate through the methods one after the other. getMethods( ) returns an array of Method objects. For each Method m present in the getMethods( ), we propose to check whether the annotation TestMethod had been applied to it or not. @TestMethod is an annotation. When we say TestMethod.class it gives the instance of the TestMethod class. We had applied TestMethod to testcase1( ) and testcase2( ) methods. We are trying to find that for the method that is in m, for that particular method is the annotation TestMethod applied or not. If that annotation has been applied then we want to call that particular method. For that within the try block we call m.invoke ( null ) and then we increment the value of passed by 1 using passed++. if gets failed for testcase3( ) method.

Program Continued

In catch block we print the message and call the getCause( ) method to figure out the cause of the exception. Then we increment the value of failed by 1 using failed++. We can also print the current value of passed and failed. The output when we execute the program is look like as shown in the slide.