**Introduction To Java (Day 01)**

**Need for the High-Level Language:**

Machines – Computers can understand only binary languages which are in the form of 0s and 1s which are nothing but electronic signals.

In order to automate things, we need to develop software which are nothing but a Group of programs which are again a group of instructions which are to be executed by machines to fulfil our tasks , be it any Process like Banking, Insurance, HR Mgmt. etc.

Therefore, we cannot write instructions in Binary languages which could become very complex, therefore we need a medium in other words a language to instruct the computers to do our tasks.

The Binary languages are termed as low-level languages and the High-level languages are the ones like Java, C, C++, C# etc.

Java is one of the most powerful High-Level languages that have been in the Software Engineering world and also a very popular language among the industry.

**Object Oriented Programming Language vs Procedural Programming Language:**  
There are different Programming Methodologies like

1. Non Structured Programming
2. Structured Programming
3. Procedural Programming
4. Object Oriented Programming

**Non Structured Programming:**

Earlier days when Programming was very new, people did not have a systematic mechanism to write Programs like grouping the instructions, commenting & documenting the codes – the lines of instructions.

Also some of the mechanisms like, taking the course of action, based on users inputs, for eg. if the user wished to continue booking tickets or to stop booking and also mechanisms like repeating the task for n number of times and all these mechanisms were to be manually defined in a step by step manner These led to huge size of the programs and also Debugging and maintaining the programs became very difficult

**Structured Programming:**

Then came the Structured Programming methodology where people started programming in a more systematic way. Where, to overcome the above disadvantages constructs like if … else , switch cases were used and to loop a particular task , mechanisms like while, do while loops were introduced.

Apart from this Documentation and commenting about the codes were mandatory. All these led to easier debugging and maintenance of the code.

Organizing the Codes became more systematic in this methodology.

**Procedural Programming :** Procedural Programming was utilizing all good features of Structured Programming and almost Structured Programming and Procedural Programming overlapped with each other.

Procedural Programming is more Process Centric, i.e the Application is written for a certain process like HR Process , Banking Process or Insurance Process, which means the activities that take place in these processes are automated.

But entire application is a single monolithic component where internal elements of the application which could be nothing but Procedures – which are again group of instructions which can perform a particular task say, checking the date , if it is greater than current date etc.,

Procedures can be reusable, but the entire application once developed would be so much monolithic, that small parts of the application cannot be used across the other applications , whereas they are all intertwined so much that they become very much interdependent and not independent , therefore this leads to debugging and maintenance issues along with other issues like not reusable across apps and so on.

Other disadvantages include not being able to represent Real Life systems and data security not being implemented

**Object Oriented Programming:** The advent of Object-Oriented Programming is one of the greatest breakthroughs in the field of Programming.

In this methodology, The System for which the software is being developed, is broken down into sub level Objects and these Objects are represented by Program specific Code units, which are called as Classes, In other words An application is made up of several elements ,each of these elements represent the real time entities of the System.

For eg. If you would like to control the entire Refrigeration System through a software, the individual components like Compressor, Controller, Switch, Compartments are all represented by Classes with all the characteristics and behaviours of those components being encapsulated in these classes.

Therefore in Object Oriented Programming, these objects are the building blocks of a Program.

In high level Design all these building blocks are termed as Objects in general, In the low level i.e the Code level Classes represent these Objects and Instances of these classes are called Objects more specifically. **Classes & Objects**

For Eg “Employee” can be a class with all the characteristics like EmployeeCode, EmployeeName, EmployeeAddress, EmployeeSalary etc and the Behavioural aspects could be , goToWork, performDuty etc.

Hence Classes are code units of a Program which encapsulate the characteristics and behaviours of real time entities.

So to visualize and experience the benefits of these classes instantiation is needed, i.e when we create an instance of this class, we would be able to experience the benefits, for example

Surya, Rajan, Rahul are all actual Employees who would exhibit the characteristics and behaviours that we have defined in the class Employee. These are instances

So Classes form a design , blue print of real time entity which can be experienced through the instances in other word Objects

Car could be a class MarutiKA01 3457 , MarutiKA02 5647,MarutiKA03768 could be instances.

Classes are also called user defined types, because we define the nature of the class and when a class is instantiated, an instance has to conform to the type that is the class we defined.

The Main features of OOP are

Inheritance

Polymorphism

Encapsulation

Abstraction

**Inheritance :** Is a feature by which we can derive a new Class from an existing class, The New class is called as “Derived” class and the class from which it is derived is called as “Base” class,

Base class is also called as “Super”,”Parent” classes

Derived class is also called as “Sub”, “Child” classes

The Derived class is supposed to inherit all the Properties and behavioural aspects from the Parent class automatically.

It helps in reusability, for example In automobile mgmt. system, when one is trying to deal with a class say “Vehicle” which has some characteristics like ChasisNo, Brakes, Wheels, engine etc, later on if somebody wants to represent Two wheeler, he need not start from the scratch, he can derive a class called TwoWheeler from the class Vehicle,

Now the TwoWheeler class will have all the features of Vehicle class and also it can have some specialized features like 2 Wheels and HandleBars.

Somebody , when needed to represent FourWheeler can again derive from Vehicle class a class called FourWheeler which again inherits all features from Vehicle class and can specialize features like 4 Wheels and Steering.

Inheritance is also called “Specialization” where the Super classes are derived into subclasses for specialization of the features along with the retention of base class features.

**Polymorphism :** Polymorphism is a feature by which an entity can exhibit itself in multiple forms. Very simple example is that we can use the operator + for adding integers as well as to concatenate 2 strings like Hello + World, Polymorphism plays a very major role in Object Oriented Programming. Polymorphism can be of 2 forms

Static Polymorphism

Dynamic Polymorphism

which you will learn in detail later.

**Encapsulation :** Encapsulation is a feature by which we can bind the characteristics and the behaviour of a real time entity within a class, characteristics are represented through Data Members or variables and behavioural aspects are represented through methods or functions.

Encapsulation is to be understood in different dimensions, Encapsulation is also a feature by which we can hide all the non essential characteristics from the user’s accessibility.

**Abstraction :** Is a feature by which we can provide accessibility to the essential characteristics of a system to an end user, abstraction implementation by default leads to Polymorphism.

**JAVA as an OOP Language**

Java is one of the most powerful OOP languages that is available in the market. Java was developed by James Gosling when he was associated with SunMicrosystems.

Though Java is an Open Source language, which means anybody can contribute to the Java API Source code, officially it is owned by Oracle Corporation which acquired Sun Microsystems

**Features of Java:**

**Simple Language:** It is one of the simplest languages to use.

**Platform Independent:** It can run on any software as well as any hardware platform**,** since it is architecture neutral.

**Complete Object Oriented Programming Language but not Pure OOP Language:**

Java Program can exist only through classes and without classes Java Programs cannot be created hence it is Complete Object Oriented, but still Java allows us to use primitive data types like int,char,float etc , therefore we can not consider Java as a pure Object Oriented Programming Language.

**Supports Multithreading**

**Supports Networking**

**Supports DeskTop , Distributed , Web Apps , Mobile Apps , Web Services & MicroServices Development**

**Portable** hence supports WORA (Write once run anywhere)

**Highly Secured**

**Supports all DBMS/RDBMS**

**Supports GUI & CLI based apps , supports Event Driven Programming.**

**Robust.** It has a very powerful Exception handling mechanism and Java apps would not easily crash.

JAVA ARCHITECTURE





JDK Provides the complete suite of tools to Develop and run Java applications.

JDK consists of JRE & Development Tools

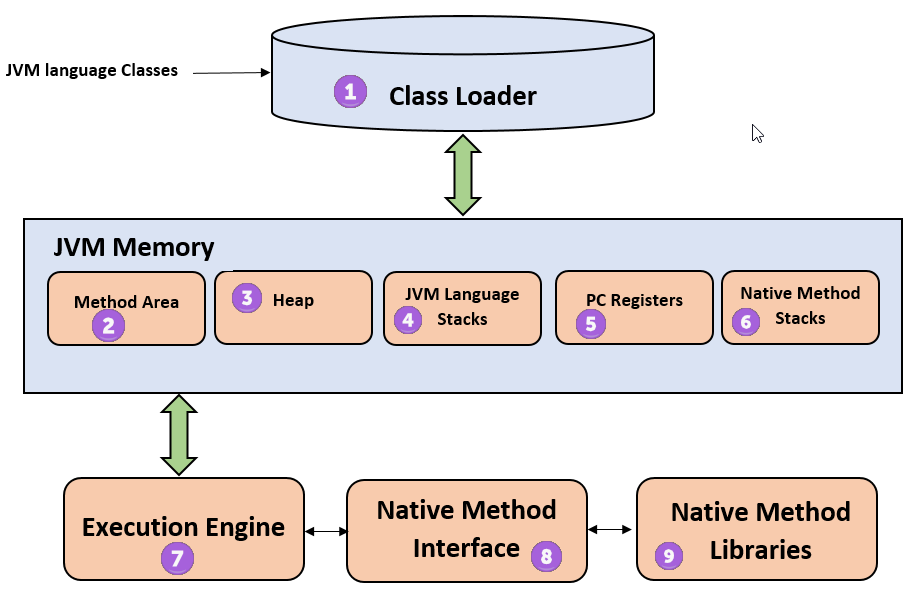
Development Tools include the Compiler (javac), Executor(java), Documentation tools and Archiver.

JRE provides an environment to execute the Java programs

JRE consists of Java API (Libraries),dependencies required to execute Java programs and JVM .

JVM is responsible to execute Java Programs which are in ByteCode format

JVM loads all dependent classes using class loader and checks the code using ByteCode Verifier and converts to the native format using JIT compiler and executes using Execution Engine.



JVM uses the above components while executing the ByteCode

1. ClassLoader : is a subsystem used by the JVM for loading the class files, it performs 3 major tasks i.e loading, Linking & initialization.
2. Method Area stores class structures like metadata , the constant runtime pool and the code for the methods
3. Heap : All the Objects, their related instance variables and arrays are stored in the heap. This memory is common and shared across multiple threads.
4. Java language stacks store local variables and its partial results. Each thread has its own JVM stack, created simultaneously as the thread is created. A new frame is created whenever a method is invoked and it is deleted when the method invocation process is complete.
5. PC registers store the address of the Java virtual machine instruction which is currently executing . In Java, each thread has its separate PC register.
6. Native method stacks hold the instruction of native code depending on the native library. It is written in other languages, not necessarily in Java.
7. Execution Engine : It is a type of software used to test hardware, software or complete systems, this is the component responsible to execute the java code .
8. Native Method interface is a programming framework. It allows java code which is running in a JVM to call libraries and native applications
9. Native Method Libraries: is a collection of the Natice Libraries(C,C++) which are needed by the execution engine
10. JIT Compiler is used by the JVM to convert the bytecode into the native format

Compiler : Used to convert the High level language into native machine code - Byte Code in Java

Linker : Used to combine different program files reference in the main java program together

Loader - To load the files from secondary storage like Hard Disk into RAM

Execution : Actual execution of the code , JVM executes it with the help of OS

**Keywords:** Keywords are specific words which have a specific meaning and are reserved for a particular purpose. Every language has its own keywords , which should not be used for user defined identifiers like variables,method names,class names etc.

**Java Keywords**

| abstract | continue | for | new | switch |
| --- | --- | --- | --- | --- |
| assert | default | goto | package | synchronized |
| boolean | do | if | private | this |
| break | double | implements | protected | throw |
| byte | else | import | public | throws |
| case | enum | instanceof | return | transient |
| catch | extends | int | short | try |
| char | final | interface | static | void |
| class | finally | long | strictfp | volatile |
| const | float | native | super | while |

In addition to the above **true, false & null** are also reserved words which should not be used for the identifiers.

**Variables & Data Types:**

Variable is a container to hold data or a specific value that is to be used by the program.

For example employeeName is a container to hold the names of the employees, when a variable is created , the type of data/value that is to be stored in it, has to be specified, for which one can use “DataType” ,

Data types are predefined terms which decides the type of data and the size of data that can be stored in a variable.

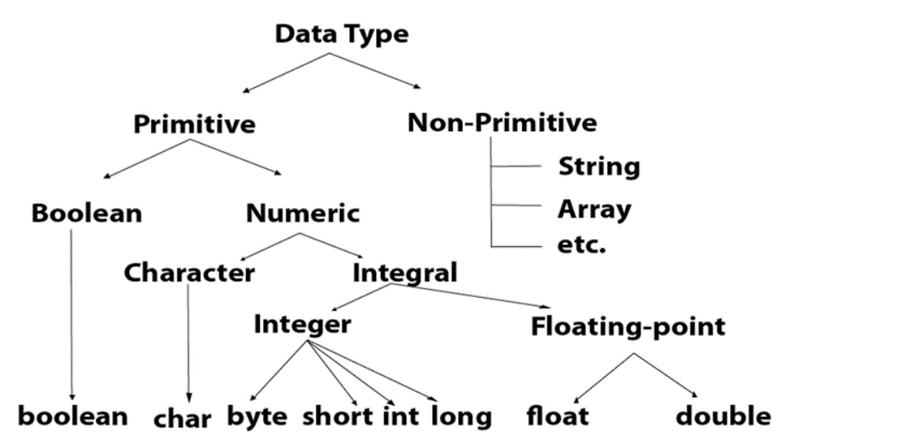
When a variable is created it gets reserved with some amount of memory depending upon the data type used for the variable.

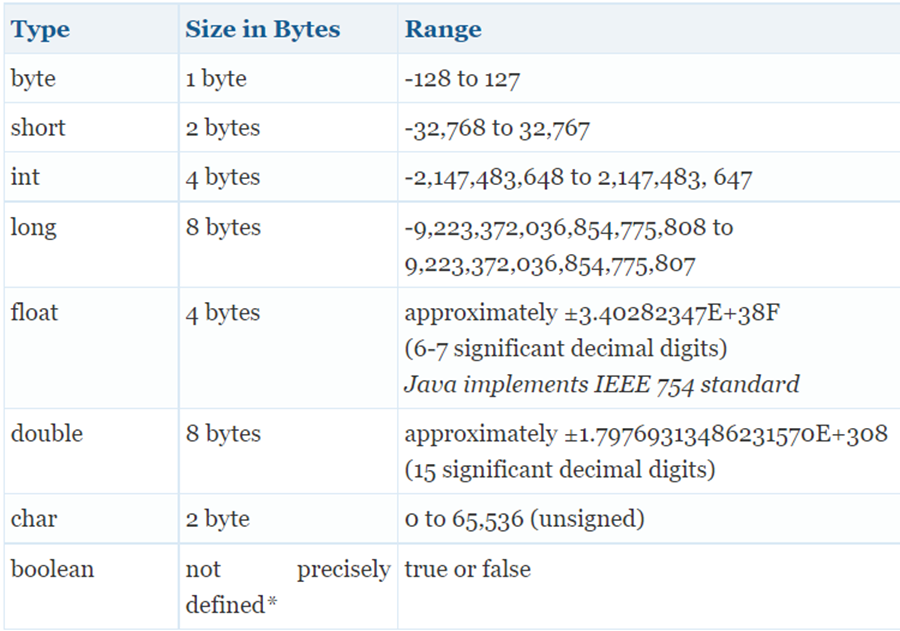
For example, when we need to store the scores of a student, scores have to be numeric. We do not have to store it as a string of characters, therefore java provides a data type called “int” which can be used for numeric/integer type of data.

When we need to store employee salaries for example , we may want it to be of decimal type, therefore java provides a data type called float to store decimal values or floating point numbers.

When we need to store employee names , we want it to be of collection of characters which we call it as a String

Data types allowed in Java:





String is a class , which will be covered in detail later.

Syntax to create a variable:

data type variable;

int studentScore; initialization : studentScore = 87

char reply; initialization : reply = ‘Y’

RULES FOR NAMING VARIABLES:

1. Must start with a letter , underscore or a dollar sign.
2. It can have combination of letters, numbers , @ and $
3. No whitespaces are allowed
4. No Keywords are allowed
5. It is case sensitive : employeeName is not same as EmployeeName

**TypeCasting :**

Sometimes we may have to transfer the data from 1 variable of a certain type to another, for example from **long** type variable to **int** typed variable.

this situation has to be carefully dealt with, as it is understood that **long** variable can store 8 bytes of data and **int** variable can store 4 bytes of data, therefore when long typed variable data is transferred to int typed variable, there is a possibility of losing the data,

This process of conversion from one type to another type is called TYPECASTING.

Let us see an example from

**long longVar = 20000;**

**int intVar = 245;**

when we do the following conversion

**IongVar = intVar** there is no issue risen, since wider typed variable is going to hold narrow typed variable data, data is also not lost, but overwritten

This type of casting takes place implicitly and is called IMPLICIT WIDECASTING

But the reverse is dangerous .

for eg. **intVar = longVar** is not implicitly done

it has to be done as in the following way

**intVar = (int)longVar;** This is called NARROWCASTING and EXPLICIT.

WIDECASTING is always implicit and NARROWCASTING has to be done explicitly

Therefore while casting 2 aspects are to be taken care of

1. Are the types compatible with
2. Are the sizes compatible? in the sense, smaller size can fit in larger sized data type but reverse is not possible, therefore for the reverse , explicit typecasting is to be done and we may have the chance of losing data

Another example and the logic of casting:

short shortVar = 124; size of short is 2 bytes and range is -32768 to +32767

byte byteVar = 100 ; size of byte is 1 byte and range is -128 to +127

Conditions

1. types are compatible
2. LHS is smaller RHS is wider so the casting is NARROWCAST and it is EXPLICIT

If we cast byteVar = (byte)shortVar

shortVar holds 124 and is within the range of byteVar, so byteVar’ s new data will be 124 (earlier it was 100)

On the other Hand

suppose short shorVar = 129 and byte byteVar = 100;

now if casting is done byteVar = (byteVar)shortVar,

what is stored in the byteVar would be surprising, new byteVar value is -127

it does 129 - 256 (the total value stored in byte) hence -127

Suppose short shortVar = 256

byte byteVar = 100

Now byteVar = (byte)byteVar , byteVar would store 256 - 256 = 0

Suppose short shortVar = 512

byte byteVar = 100

Now byteVar = (byte)byteVar, byteVar would store 512 % 256 i.e 0

Suppose short shortVar = 514

byte byteVar = 100

Now byteVar = (byte)byteVar, byteVar would store 514 % 256 i.e 2

Same logic is applicable for other data types casting.

Valid Castings allowed are

Widening Cast-Implicit

**byte ->short ->char ->int ->long->float->double**

Narrowing Cast-Explicit

**double->float->long->int ->char->short->byte**

Also when we do float type casting it has the following effect:

for eg.

float floatVar = 2.645f float is 4 bytes

int intVar = 200; int is also 4 bytes and float to int is considered narrow casting as in the above mentioned rule

intVar = (int)floatVar

Now intVar will just store 2 since it is int, please note no rounding off is done

suppose we have

float fVar = 100.0f;

int iVar = 20;

now fVar = iVar which is wide casting will result in fVar holding 20.0

**Operators**

Operators are the tools which are used to perform operations. The Operators available in Java are

**Arithmetic Operators :**

**+ For addition for eg. 200 + 300 gives 500**

**- For Subtraction for eg. 400 - 200 gives 200**

**\* For Multiplication for eg. 10 \* 20 gives 200**

**/ For Division for eg. 200 / 50 gives 4**

**% For Modulo for eg. 200 % 6 gives 2 reminder**

**Assignment Operators:**

**= For eg. iVar = 200**

**+= For eg. iVar+=20 is equivalent to iVar = iVar+20**

**-= For eg. iVar-=20 is equivalent to iVar = iVar - 20**

**\*= For eg. iVar\*=20 is equivalent to iVar = iVar \* 20**

**/= For eg. iVar/=20 is equivalent to iVar = iVar / 20**

**%= For eg. iVar %= 10 is equivalent to iVar = iVar % 10**

**Relational Operators**

Also called as Comparison Operators which are use to compare after comparison the result will be true or false based on the values that are being compared

**> , >= , < , <= , == , !=**

for eg. int a = 100, int b = 200 ; Now a < b will return true and so on

**Logical Operators**

**&& , || and !**

Logical operators help us in comparing conditions for eg

int var1 = 100 , var2 = 200

if ( var1 >= 100 && var2 >= 100) returns true if both the conditions are satisfied

if (var1 >= 100 || var2 >= 100) returns true if atleast one condition is satisfied

if(! var1 >= 100) returns true when the condition is not satisfied else returns false

**Unary Operators**

**++ , - - ,**

int var=100; var++ is used in post increment operations

int var=200; ++var is used in pre increment operations

In Post Increment/decrement, first the value is assigned and then operation takes place

whereas in PreIncrement, first the increment/decrement operation takes place then the assignment takes place

for example

int varx = 10 ; int vary = 20;

if we do the operation as in below statements

int result = varx++ \* vary++

System.out. println(“result is ”+result) ; will display 200

System.out.println(“varx value is “+varx) ; will display 11

System.out.println(“vary value is “+vary); will display 21

whereas

if we do the operation as in below statements

int result = ++varx \* ++vary

System.out. println(“result is ”+result) ; will display 231 which is 11 \* 21

System.out.println(“varx value is “+varx) ; will display 11

System.out.println(“vary value is “+vary); will display 21

**Bitwise Operators**

Bit Operators operate upon Bits

**~**  Negation Operator; For eg. for a value 101; ~101 will results in 010

**<<** Left Shift Operator; For eg. for a value 010; 010 << 1 results in 100

**>>** Right SHift Operator; For eg. for a value 010; 010 >> 1 results in 001

**<<<** Unsigned RightShift Operator For eg.

int num1 = 6 and int num2 = -6

num1 >>> 1 results in 13

num2 >>> 1 results in 12147483645

The Unsigned Right Shift Operator moves the bits of the integer a given number of places to the right. The sign bit was filled with 0s.

**&**  Bitwise AND operator ; 010 & 011 results in 010

**^** XOR Operator ; 0101 ^ 1100 results in 1001

**Ternary Operators**

**Condition ? statement1 : statement2**

The Condition is checked if it is satisfied it performs Statement1 else it performs statement2

int age = 23;

age >= 25 ? display “Eligible for Job” : display “Not Eligible for Job”

Here it displays “Not eligible for Job”

**Comments**

Comments are used to provide documentation-al information about the Code that is written, it is a sort of MetaData.

Comments will not be compiled , the compiler understands that commented lines for extra information about the code and therefore commented lines are ignored for the compilation.

Comments are useful for better readability and understanding of the code, it also helps during debugging and maintenance of the codes.

It is always a good practice to give meaningful comments for the codes we write.

**Types Of Comments:**

There are basically 3 types of comments

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment

**Single Line Comment**

Eg.

public class DocumentSample

{

public static void main(String[] argos)

{

// This is a single line comment

System.out. println(“Hello World”);

}

}

**Multi Line Comment**

Eg.

public class DocumentSample

{

public static void main(String[] argos)

{

/\*

This is a Multi line comment

We can have n number of lines

\*/

System.out. println(“Hello World”);

}

}

**JavaDoc Comments**

JavaDoc comments can be converted to JAVADOC which acts as a very good documentation for the java applications

Eg.

public class DocumentSample

{

public static void main(String[] argos)

{

/\*\*

This is a Multi line comment

We can have n number of lines

\*/

System.out. println(“Hello World”);

}

}

**Constructs & Loops**

Constructs & Loops play a very major role in Structured Programming languages.

Constructs are helpful in carrying out mechanisms, which involves in taking the course of action of a Program, depending upon a certain condition.

These features got descended to the Procedural and Object Oriented Programming Languages from Structured Programming languages.

Primary Constructs being used in Java are as follows:

1. If .. else Construct
2. if .. else if Construct
3. Nested If
4. Switch Case construct

**If .. else**

If else Construct is used to perform tasks based on a condition, If a condition is satisfied, corresponding activity is performed or else a different activity is performed.

For eg. A scenario where, if the score of a student is greater than 50 , a message “Good!! Passed “ is to be displayed otherwise a message “Sorry Try again” is to be displayed

Java Code snippet for the above scenario is as follows:

public class IfConstruct

{

public static void main(String[] argos)

{

int score = 65;

if (score >= 50)

{

System.out.println(“Good!! passed and Score is “+score);

}

else

{

System.out.println(“Sorry! Try again”+score);

}

}

}

The above code would give an output as follows:

Good!! passed and score is 65

In fact, after the if condition , It is mandatory to enclose all the statements within a pair of curly braces when there are more than 1 statement.

But if there is only one statement after the if condition, then the pair of curly braces is not mandatory, but it is always a good practice for the same.

**If else if:**

When multiple conditions are to be checked and corresponding actions are to be taken, then if else if constructs can be used.

Java Code snippet for the above scenario is as follows:

public class IfElseIf

{

public static void main(String[] argos)

{

int score = 65;

if( (score >= 50) && (score < 60))

{

System.out. println(“Good Passed”);

}

else if((score >= 60) &&(score < 70))

{

System.out. println(“V Good Secured First Class”);

}

else if((score >= 70)&&(score <= 100))

{

System.out. println(“Excellent Secured Distinction”);

}

else

{

System.out. println(“Sorry Try again”);

}

}

}

The above code would give an output as follows:

V Good Secured First Class

**Nested If:**

In the above example even if the score is 120 , it gives the output as “Sorry Try Again”, therefore in such circumstances, nested If can be used as shown below

public class IfElseIf

{

public static void main(String[] argos)

{

int score = 65;

if((score >= 0) && (score <= 100))

{

if( (score >= 50) && (score < 60))

{

System.out. println(“Good Passed”);

}

else if((score >= 60) &&(score < 70))

{

System.out. println(“V Good Secured First Class”);

}

else if((score >= 70)&&(score <= 100))

{

System.out. println(“Excellent Secured Distinction”);

}

else

{

System.out. println(“Sorry Try again”);

}

}

else

{

System.out.println(“Sorry Valid Range of Score is 0-100”);

}

} // Main function ends

} // Class ends

In the above snippet, when a value of 120 is entered for score, it gives the output as

“ Sorry Valid Range of Score is 0-100”

This is how a nested if can be used.

**switch.. caseConstruct:**

In the above examples we saw , if a variable value is in different ranges of values and at a point a variable can hold one of the ranges, we can perform activities accordingly using **if.. else** or **if else if** constructs, but when we want to check and take the course of actions for a particular value for a variable and when this particular value may differ and accordingly when the course of action is to be taken, we can use “switch” case construct.

Java Code snippet for the above scenario is as follows:

public class Switcher

{

public static void main(String[] args)

{

int day = 4;

switch(day)

{

case 1: {

System.out. println(“The Day is Monday”);

break;

}

case 2: {

System.out. println(“The Day is Tuesday”);

break;

}

case 3: {

System.out. println(“The Day is Wednesday”);

break;

}

case 4: {

System.out. println(“The Day is Thursday”);

break;

}

case 5: {

System.out. println(“The Day is Friday”);

break;

}

case 6: {

System.out. println(“The Day is Saturday”);

break;

}

case 7: {

System.out. println(“The Day is Sunday”);

break;

}

default: {

System.out. println(“Valid Range is 1-7”);

break;

} // Default Closing

} // Switch Closing  
} //Main closing

} // Class closing

In the above example , **cases** can take int, char or String values;

And though **break** is not mandatory, it avoids checking other subsequent conditions, when a condition is met already and breaks out of switch construct and **default** is also not mandatory though it is helpful to check the values which are out of range.

**Packages:** Packages are the ways to organize the related classes, it acts as a container to group and place all the related classes.

Java has a wide number of predefined packages where the predefined classes are placed.

For eg. all Database related predefined classes are all placed under a package called

java.sql.\*

* \* Represents multiple classes.

to utilize the features/functionalities of a class one has to import in the respective user defined classes.

for example to use the java.sql.\* in a class the following would be the syntax.

import java.sql.\*;

class UserClass

{

}

If we want to individually import the classes we can do as follows:

import java.sql.Connection;

import java.sql.ResultSet;

This is just an example, you will be learning more on java.sql.\* in forthcoming chapters.

Usual convention to name an user defined package would be as follows

com.orgname.packagename

all lower cases

Various Packages available in Java are

java.util.\*

java.lang.\*

java.io.\*

java.awt.\*

javax.swing.\*;

java.sql.\*

java.net.\*

We shall understand more on these packages in future sessions

**Introduction To Java (Day 02)**