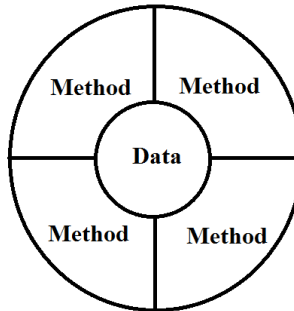


UNIT-I

FUNDAMENTALS OF OOPS

Object Oriented Paradigm:

- OOPs treats data as a critical element in the program development and does not allow it to flow freely around the system.
- It ties data more closely to the functions that operate on it and protects it from unintentional modification by other functions.
- OOP allows us to decompose a problem into a number of entities called objects and then build data and functions(Methods) around these entities.



- The data of object can be accessed by the methods associated with that object.
- Some of the features of Object Oriented Paradigm are:
 - @ Emphasis is on data rather than the procedure.
 - @Programs are divided into Objects.
 - @Data structures are designed such that they characterize the objects.
 - @Methods that operate on the data of an object are tied together in the data structure.
 - @Data hidden cannot be accessed by external functions.
 - @Objects may communicate with each other through methods.
 - @New data and methods can be easily added whenever necessary.
 - @Follows bottom-up approach in program design.

Object Oriented Programming is an approach that provides a way of modularizing by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

Basic Concepts of OOP:

OBJECT:

1. An object is a software entity that combines a set of data with a set of operations to manipulate the data.
2. An object is an instance of a class.
3. An object is known by a name and every object contains a state.
4. The state of is determined by the values of properties.
5. The state of an object can be changed by calling methods on it.
6. The sequence of states represent the behavior of the object.

CLASS:

1. A class is defined as the blue print of the object.

2. A serves as a plan or a template.
3. Description of a number of similar objects is also called a class.
4. A class is a user-defined data type.
5. A class contains two things i.e., properties and methods.

ENCAPSULATION:

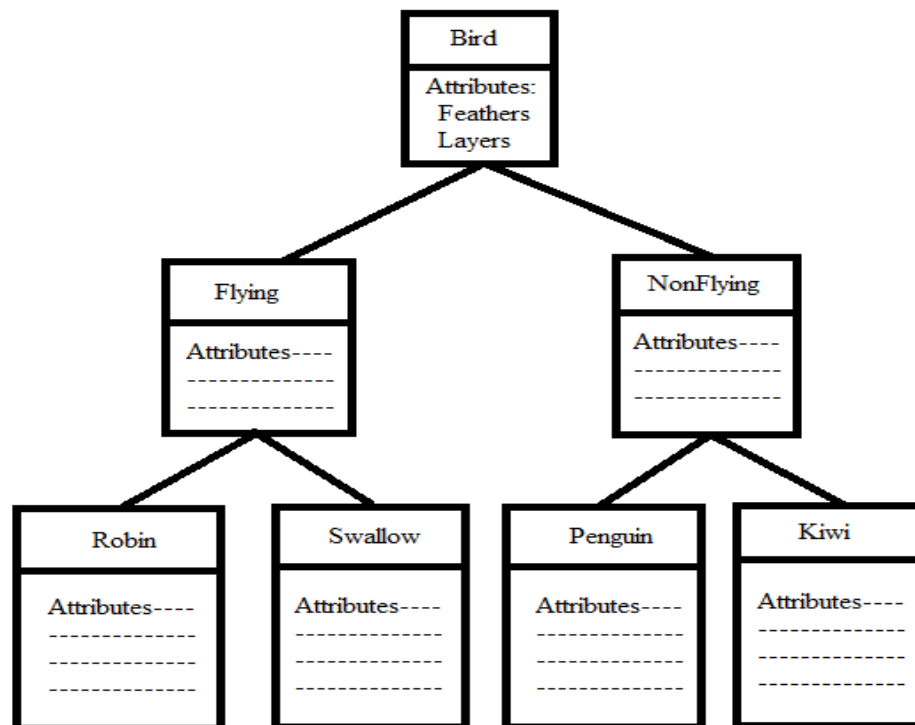
1. Encapsulation is one of the features of object oriented methodology.
2. The process of binding the data and methods that operate on data into object to hide them from the outside world is called as encapsulation.
3. Encapsulation is also known as data hiding. Data encapsulation is the striking feature of a class.

ABSTRACTION:

1. Abstraction refers to the act of representing essential features without including the background details or explanations.
2. Classes use the concept of abstraction and are defined as a list of properties and methods to operate on these properties.

INHERITANCE:

1. Inheritance is the way to adopt the characteristics of a class into another class.
2. Here we have two types of classes. One is base class(super class) and the other one is derived class(sub class).
3. There exist a parent-child relationship among the classes.
4. A sub class inherits all the properties of a base class, in addition to this it can add its own features.

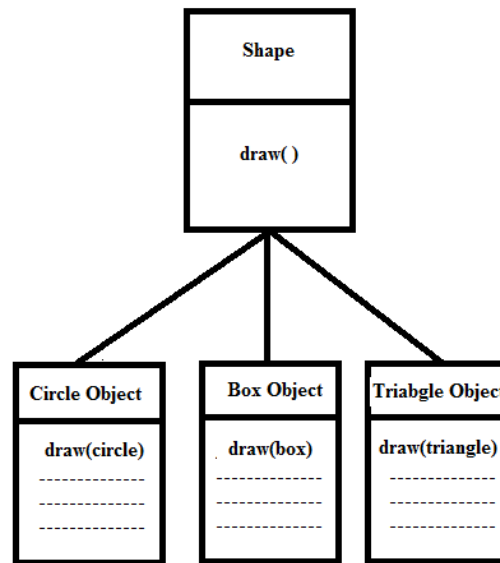


In the above example Bird Robin is a part of the class Flying Bird, which is a sub part of the class Bird. The concept of inheritance provides the idea of reusability.

This means that we can add additional features to an existing class without modifying it.

POLYMORPHISM:

1. Polymorphism means the ability to take more than one form.
2. An operation may exhibit different behavior in different instances.
3. The behavior depends up on the type of data used in the operation.
4. Consider the operation of addition for two numbers, the operation will generate sum, if the operands were strings, then the operation will produce a third string by concatenation.



In the above example, a single method name can be used to handle different number and different types of arguments.

This is something similar to a particular word having several different meanings depending on the context.

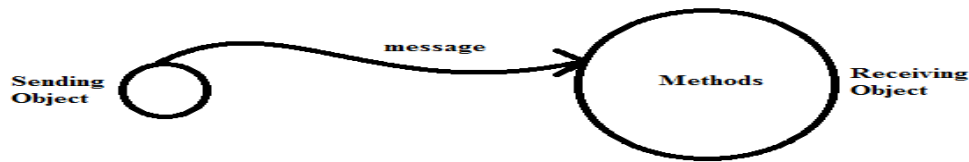
DYNAMIC BINDING:

1. Binding refers to the linking of method call to the method.
2. Dynamic Binding means that the code associated with a method call is not known until the time of the call at runtime.
3. It is associated with polymorphism and inheritance.

MESSAGE PASSING:

1. An object oriented program consists of a set of objects that communicate with each other.
2. The process of programming in an object oriented language involves the following basics.
 - a. Creating classes that define object and their behavior.
 - b. Creating object from class definitions.
 - c. Establishing communication between objects.

Objects communicate with one-another by sending and receiving information much the same way as people pass messages to one-another is called message passing.



Message passing involves specifying the name of the object, the name of the method(message) and the information to be sent. Consider the following example:

Employee.salary(name); Employee→Object ; salary→message ; name→information

BENEFITS OF OOPs:

OOPs offers several benefits to both the programmer and the user.

1. Through inheritance we can eliminate redundant code and extend the use of existing classes.
2. We can build programs from the standard working modules that communicate with one another rather than having to start writing the code from scratch.
3. The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
4. It is possible to have multiple objects to co-exist without any interference.
5. It is possible to map objects in the program domain to those objects in the program.
6. It is easy to partition the work in a project based in objects.
7. The data centered design approach enables us to capture more details of a model in an implementation.
8. Systems can be easily upgraded from small to large systems.
9. Message passing techniques for communication between objects make the interface descriptive with external systems much simpler.

APPLICATIONS OF OOPs:

1. Real time systems
2. Simulation and modeling
3. Object-Oriented database.
4. Hypertext, Hypermedia
5. Artificial Intelligence and Expert Systems
6. Neural Networks and Parallel programming
7. Decision support and office automation systems.

OVERVIEW OF JAVA LANGUAGE

INTRODUCTION:

1. Java is a general purpose Object Oriented Programming. It was developed by James Gosling and Sun Micro Systems in 1990.
2. The original name of the language was “Oak”.
3. In 1990, Sun Microsystems intended to develop a software, that can be used to program electronic devices.
4. In 1991, exploring all possibilities of the languages like C and C++ they announced the requirement of platform independent language.

5. In 1992, the Green Project team demonstrated the applications of OAK with electronic devices and controlling home appliances.
6. In 1993, the WWW appeared on the internet and text based internet can be converted to graphical based internet. The green project team came up with an idea of developing web applets.
7. In 1994, the web browser “Hot Java” was developed. That will locate and execute applet applications.
8. In 1995, OAK was revised and renamed as “Java”.
9. In 1996, Java 1.0 was released.
10. In 1997, Sun Microsystems developed another version of Java i.e., JDK1.1(Java Development Kit).
11. In 1998, JDK 1.2 was released which is a standard edition.
12. In 1999, Sun releases Java 2(J2) enterprise edition.
13. In 2004, Java 5.0 was released and in 2007 Java 6.0 was released.

FEATURES OF JAVA:

The invention of Java wanted to design a language which could offer solutions to some of the problems encountered in modern programming. They wanted the language to be not only reliable, portable, and distributed but also simple, compact and interactive. Sun Microsystem describes Java with the following features:

- Compiled and Interpreted
- Platform-Independent and Portable
- Object-Oriented
- Robust and Secure
- Distributed
- Simple and Small
- Multithreaded and Interactive
- High Performance
- Dynamic and Extensible

These features have made Java the first application language of the World Wide Web. Java will also become the premier language for general purpose stand-alone applications.

Compiled and Interpreted:

1. Generally a computer language is either compiled or interpreted. Java combines both these approaches thus making Java a two-stage system.
2. First, Java compiler translates source code into **bytecode** instructions. Bytecodes are not machine instructions.
3. Therefore, in the second stage, Java interpreter generates machine code that can be directly executed by the machine that is running the Java Program.
4. Thus we can say that Java is both a compiled and an interpreted language.

Platform-independent:

1. The most significant contribution of Java over other languages is its portability.
2. Java programs can be easily moved from one computer system to another, anywhere and anytime.
3. Changes and upgrades in operating system, processor and system resources will not force any changes in Java Programs.

Portable:

1. We can download a Java applet from a remote computer onto our local system via internet and execute it locally.
2. Java ensures portability in two ways. First, Java compiler generates bytecode instructions that can be implemented on any machine.
3. Secondly, the size of the primitive data types is machine independent.

Object-Oriented:

1. Java is a true object-oriented language. Almost everything in Java is an Object.
2. All program code and data reside within objects and classes..
3. Java comes with an extensive set of classes, arranged in packages, that we can use in our programs by inheritance.

Robust:

1. Java is a robust language. It provides many safeguards to ensure reliable code.
2. It has strict compile time and run time checking for data types.
3. Java incorporates the concept of exception handling which captures series errors and eliminates any risk of crashing the system.

Secure:

1. Security becomes an important issue for a language that is used for programming on Internet.
2. Threat of viruses and abuse of resources are everywhere. Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet.
3. The absence of pointers in Java ensures that programs cannot gain access to memory locations without proper authorization.

Distributed:

1. Java is designed as a distributed language for creating applications on networks.
2. It has the ability to share both data and programs.
3. Java applications can open and access remote objects on Internet as easily as they can do in a local system.
4. This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

Simple and Small:

1. Java is a small and simple language.
2. Java does not use pointers, preprocessor header files, goto statement and many others.
3. It also eliminates operator overloading and multiple inheritance.
4. Familiarity is another striking feature of Java. Java uses many constructs of C and C++ and therefore Java code looks like a C++ code.
5. Java is a simplified version of C++.

Multithreaded and Interactive:

1. Multithreaded means handling multiple tasks simultaneously. Java supports multithreaded programs.
2. This means that we need not wait for the application to finish one task before beginning another.
3. The Java runtime comes with tools that support multiprocess synchronization and construct smoothly running interactive systems.

High Performance:

1. Java performance is impressive for an interpreted language, mainly due to the use of intermediate bytecode.
2. Java architecture is designed to reduce overheads during runtime.
3. The incorporation of multithreading enhances the overall execution speed of Java program.

Dynamic and Extensible:

1. Java is a dynamic language. Java is capable of dynamically linking in new class libraries, methods, and objects.
2. Java programs support functions written in other languages such as C and C++. These functions are known as native methods.
3. Native methods are linked dynamically at runtime.

Java Program Structure:

1. A Java program may contain many classes of which only one class defines a main method.
2. Classes contain data members and methods that operate on the data members of the class. Methods may contain data type declarations and executable statements.
3. To write a Java program, we first define classes and then put them together.

Documentation Section
Package Statement
Import Statement
Interface Section
Class Definition
Main class { Main method() { } }

Documentation Section:

1. This section consists of commenting lines. The comments can be used anywhere in the program.

2. Generally, the comment section includes the documentation of the java program. Comments can help us to understand the code to non-programmers.
3. In general, comments include the following data.
 - a. Name of the program
 - b. Purpose of the program
 - c. Author of the program
 - d. Copy right information
4. Java supports different kinds of comments.
 - a. Single line comments(//): ex: // This is my first java program.
 - b. Multi line comment: If comment runs over a number of lines we use multi line comment (/* */)Ex: /* This program was developed at Aditya Degree College
By Aditya students */
 - c. Java document comment(** ... */)

Package statements:

1. The first line allowed in java program is package section. It contains package name.
2. Package name informs the compiler that all the classes are belonging to this package.
3. General Syntax: package package_name;
4. Package statement is optional when our classes must not have to be the part of the package.

Import Statement:

1. This statement is used to add a specific class or classes of a package to the current java program.
Ex: import java.lang.System;
2. The above statement imports the class “System” from java.lang package to our program.
Ex: import java.lang.*;
3. The above statement imports all the classes form the package java.lang to our program. The import statement instructs the interpreter to load the classes from the specified package.

Interface section:

1. The interface section is exactly similar to the class but it includes a group of method declarations.
2. It is used only when we are implementing “Multiple Inheritance”.

Class Definition Section:

1. A java program may contain any number of class definitions.
2. Classes are primary and essential components of a java program.
3. A class is used to create user-defined data-types.
4. Every class contains a set of properties and methods that operate on the properties.

Main Class Definition:

1. Any java program contains a main method, and it is the starting point of execution of the program.
2. Main() method creates the objects of various classes and established communication between them.
3. Once controller reaches the boundary end of main program, the program terminates and the controller pass back to operating system.

Differences between Java and C:

1. Java does not include the C unique statement keywords **sizeof**, and **typedef**.
2. Java does not contain the data types **struct** and **union**.
3. Java does not define the type modifiers keywords **auto**, **extern**, **register**, **signed**, and **unsigned**.
4. Java does not support an explicit pointer type.
5. Java does not have a preprocessor and therefore we cannot use **#define**, **#include**, and **#ifdef** statements.
6. Java adds new operators such as **instanceof** and **>>>**.
7. Java adds labeled **break** and **continue** statements.
8. Java adds many features required for object-oriented programming.

Differences between Java and C++:

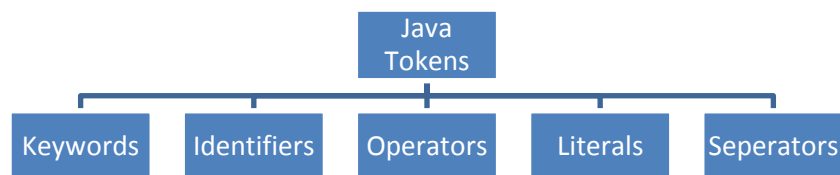
1. Java does not support operator overloading.
2. Java does not have template classes as in C++.
3. Java does not support multiple inheritance of classes. This is accomplished using a new feature called “interface”.
4. Java does not support global variables. Every variable and method is declared within a class and forms part of that class.
5. Java does not use pointers.
6. There are no header files in Java.

JAVA & INTERNET:

1. Java is strongly associated with Internet.
2. The Major principles behind Java and HTML is same i.e., platform independent.
3. The first program written in Java for the internet is “hot Java” i.e., a web browser.
4. Internet users can create an applet application(a java program to be displayed on a web browser) and execute that application in our local computer with java enabled web browser.

JAVA TOKENS:

The smallest individual unit in a language is called tokens. The compiler recognizes the tokens for building up expressions and statements. There are five major categories of Java tokens available in Java Programming.



Keywords:

- ➔ These are also called as reserved words of the language.
- ➔ Keywords have a specific meaning.
- ➔ We cannot use the keywords as identifiers.
- ➔ All keywords are written in lower case letters only.
- ➔ In Java, we have more than 50 keywords.

abstract Assert boolean Break byte case catch

Char	Class	const	continue	default	do	double
Else	Enum	extends	Final	finally	float	for
Goto	If	implements	Import	instanceof	int	interface
Long	Native	new	Package	private	protected	public
Return	Short	static	Strictfp	super	switch	synchronized
This	Throw	throws	transient	try	void	volatile
while						

Identifiers:

- ➔ These are the programmer designed tokens.
- ➔ Those are used for giving names to variables, names to classes, names to packages, names to interfaces and to programs.
- ➔ There are certain rules and conventions, we must follow while using identifiers.

Rules:

- ➔ They have alphabets, digits and an underscore(_).
- ➔ It must begin with alphabets and followed by digits.
- ➔ It must not be a keyword.
- ➔ Other than the underscore no special symbols are permitted.
- ➔ Spaces should not be allowed in variable names.
- ➔ Upper case letters defer from lower case letters.

Conventions:

- ➔ Class names begin with upper case alphabet.
- ➔ If method name contains two words, we merge them by capitalizing the first letter of second word.
- ➔ Ex: getData()

Literals:

- ➔ Literals are the sequence of characters that represent constant value i.e., to be stored in a variable.
- ➔ Java supports five categories of Literals.
 1. Integer Literals (25, +10, -23,...)
 2. Float Literals (3.4, -3.4,...)
 3. Character Literals ('A', 'a',.....)
 4. String Literals ("Moon", "JAVA",.....)
 5. Boolean Literals (true,false)

Operators:

- ➔ An operator is a symbol that is used to manipulate Data.
- ➔ An operator takes one (or) more inputs (operands) and produces result by operating them.
- ➔ If the operator takes only one operand that is called unary operator (+a,-a,...).
- ➔ If the operator takes two operands then it is called as binary operator(a+b, c=a/b,...).

Separator:

- ➔ These are the symbols used to indicate where the group of code is divided and arranged.
- ➔ The following are the distinct separators.

; ➔ Used to separate statements or terminates the statement.

,	→	Used to separate continues identifiers in variable declaration.
.	→	Used to separate package names from classes and also used to separate the variable names with objects.
[]	→	Used declare an array and specifying the size.
{ }	→	Used to indicate starting of a class, interface, method and also used to initialize the values for an array.
()	→	Used in method definition and calling of methods.

JAVA STATEMENTS:

1. Empty Statement:- These do nothing and are used during program development as a place holder.
2. Labeled Statement:- Any statement may begin with a label. Such labels must not be keywords.
3. Expression Statements:- Most of the java statements are expression statements. Java has 7 types of expression statements.
4. Selection Statements:- These are used to select one of several control flows. There are three types of selection statements in Java.
5. Iterative Statement:- These specify how and when looping will take place. There are three types of iterative statements in Java.
6. Jump Statements:- Jump statements pass control to the beginning or end of the current block, or to a labeled statement.
7. Synchronization Statement:- These are used for handling issues with multithreading.
8. Guarding Statements:- Guarding statements are used for safe handling of code that may cause exceptions.

IMPLEMENTING A JAVA PROGRAM

Implementing a Java program involves three steps.

1. Creating a Java Program
2. Compiling a Java Program
3. Running a Java Program

1. Creating a Java Program:

Creating a java program means writing a java program in any one of the text editors like notepad. Before writing a java program, first we need to check whether the java software installed or not.

If installed, then you can start writing the java program.

```
Ex: import java.lang.*;
    class FirstProgram
    {
        Public static void main(String args[])
        {
            System.out.println("Welcome to Java World");
        }
    }
```

Save the above program with the name FirstProgram.java.

2. Compiling a Java Program:

By using java compiler we can convert the source program into byte code. This byte code is generated in any machine. This byte code is system dependent.

General Syntax: C:\>javac FileName.java

C:\>javac FirstProgram.java

It will generate dot class file, i.e., FirstProgram.class.

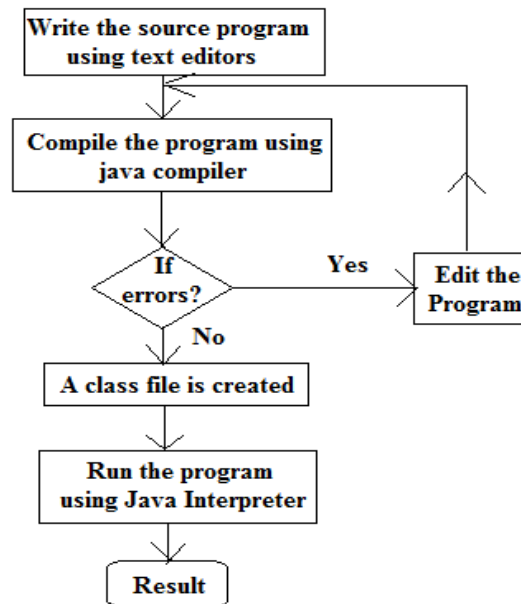
3. Running a Java Program:

The byte code is not directly executed by any machine. The java interpreter can take the byte code and generate machine executable instructions.

General Syntax: C:\>java ClassName

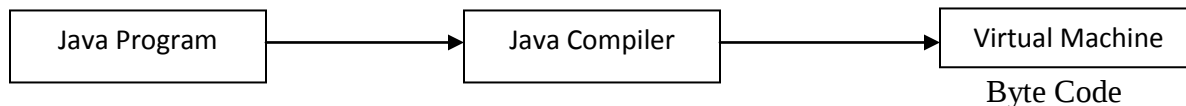
C:\>java FirstProgram

The following flow chart represents a sequence of steps involved in implementing a Java program.

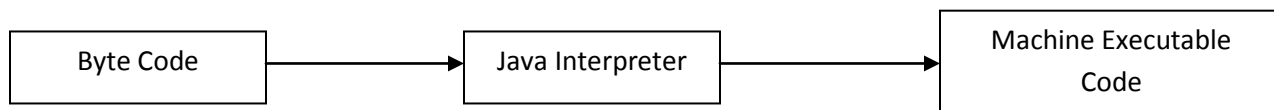


JAVA VIRTUAL MACHINE (JVM)

Java compiler produces an intermediate code called bytecode, it is taken by a machine that does not exist. This machine is called Java Virtual Machine and it exists only inside the computer memory.

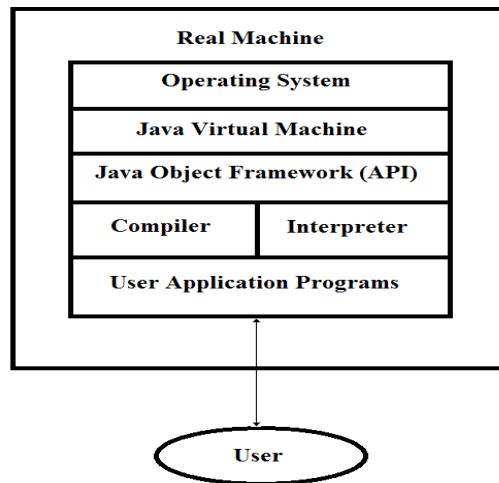


The Byte code generated by the java compiler is not machine specific. So it can be executed on any machine. But the executable code is machine specific, which means executable code is different from one operating system to another operating system.



Java Object Framework (API) acts as the intermediary between the user programs and the virtual machine.

However the virtual machine acts as the intermediary between the operating system and Java API.



COMMAND LINE ARGUMENTS:

Command Line Arguments are the values passed to the Java program at the time of program execution. Any arguments provided in the command line are passed to the array **args** which is defined in main method. The arguments are passed to the main from the command prompt by the interpreter.

`public static void main(String args[ ])` is the general form of main method in java. Here `args[ ]` is an array of string type which can store arguments passed to from the command prompt. The first argument will be stored at `args[0]` and the `n`th argument will be stored at `args[n-1]`.

The following program demonstrates command line arguments.

```

import java.lang.*;
class Demo
{
    public static void main(String args[ ])
    {
        int c=args.length;
        for(int i=0;i<c;i++)
        {
            System.out.println(args[i]);
        }
    }
}

```

- ➔ Save the above program as Demo.java.
 - ➔ Compile the program as `javac Demo.java`
 - ➔ Run the program as `java Demo Aditya Degree College`
- The output is
- ```

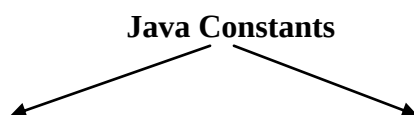
Aditya
Degree
College

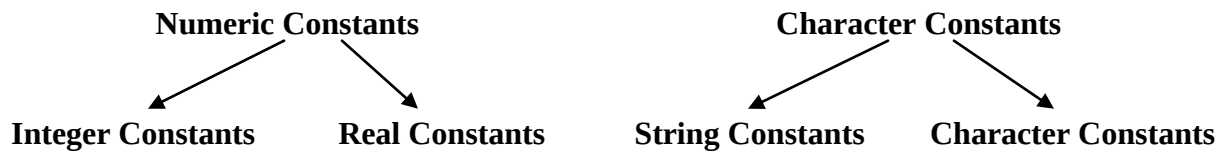
```

## CONSTANTS VARIABLES & DATA TYPES

### CONSTANTS:

Constants in Java refers to fixed values that do not change during the execution of a program. Java supports several types of constants. They are





#### Integer Constants:

1. An Integer constant refers to a sequence of digits. There are three types of integers, namely decimal integer, octal integer and hexa-decimal integer.
2. Decimal integer consists of a set of digits 0-9. Embedded spaces, commas, and non digit characters are not permitted between digits.  
Ex: 45342, 2344 // valid constants  
Ex: 45 43 1,233 \$123 // invalid constants
3. An octal integer constant consist of any combination of digits from the set (0 to 7) with a leading 0.  
Ex: 037, 045, 0146, 0234
4. A sequence of digits preceded by 0X or 0x is considered as hexa-decimal integer constant. They may also include alphabets A to F or a to f. Letters A to F represents the numbers from 10 to 15.  
Ex: 0X123, 0X145 0X23D

#### Single Character Constants:

1. A single character constant contains a single character enclosed within a pair of single quotes.  
Ex: 'S', 'X', 'A' etc.

#### String Constants:

1. A string constant is a sequence of characters enclosed within double quotes. The characters may be alphabets, digits, special characters and blank spaces.  
Ex: "Aditya", "1989"

#### Symbolic Constants:

Sometimes we don't want to change the value of a variable until the compilation of the program. So that at that time we use symbolic constants.

These are very much useful in a program for the following reasons.

1. Understandability:  
It is very clear that  $\pi * r * r$  is better than  $3.14 * r * r$ . Here the value of  $\pi$  is 3.14 and if we declare  $\pi$  as symbolic constant, that will be useful any number of times in the same program.
2. Modifiability:  
If the value of 3.14 is used several number of times in a program and we need to change the value to 3.141516 without disturbing any line of the code.  
The symbolic constants must be declared inside the class and should not be inside the method.  
The symbolic constants must be declared by using the keyword final.

Syntax:

```
final datatype identifier=value;
```

Example:

```
final float pi=3.141;
```

#### VARIABLES:

1. A variable is an entity whose value may vary during program execution.
2. A variable is an identifier that can be used to store values. Variables store different values but one at a time.
3. A data-type is always associated with a variable. The data-type of the variable decides what types of value it can store.

#### Variable Declaration:

We must declare a variable before it is used in program. The declaration of variable contains data-type which indicates what type of value it is going to be contained and what is the range of values.

Syntax: datatype variable1, variable2, ... ;

The place of declaration decides the scope of variable.

### Initialization of a Variable:

When a variable is declared, it is commonly contains some default values. We can assign some initial values to the variable while declaration, it is called as initialization.

Ex: `int a=10;` variable initialization

### Assignments:

Storing a value or result of an expression in a particular variable is known as assignment. There are three types of assignments in Java programming.

- a. Chained assignment: One value can be assigned to more than one variable.

Ex: `x=y=z=5;`

- b. Embedded assignment: If one assignment consists another assignment.

Ex: `x=(y=50)+10;`

- c. Compound assignment: An assignment involves arithmetic expression.

Ex: `x=y+50;`

### Scope of Variables:

Depending on the place of declaration of a variable, the variable holds some properties like life time. There are three types of variables.

#### 1. Local variables:

- a. The variables that are declared and used inside the method are called as local variables. These can be accessed from starting of the method to ending of the method.
- b. The value of local variable is not visible to outside of the method. Local variables can also declare inside a block.
- c. When the controller comes out of the block or method, the value of local variables exit.

Ex: `void add()`

```
{
 Int a=5,b=3;
 C=a+b;

}
```

Here a,b,c are local variables.

Ex: `for(int i=0;i<10;i++)`

```
{
}
```

Here “i” is the local variable for ‘for’ loop.

#### 2. Instance variables:

- a. These variables are declared inside a class. Instance variables are created when the objects are instantiated and then they are associated with objects.
- b. They take different values for each object. The memory is not common for all the objects, memory to each object will be individually allocated.
- c. Instance variables should not be preceded with the keyword “static”.

Ex: `class Student`

```
{
```

```

int rollno;
String name;
void getData()
{

}
.....
}

```

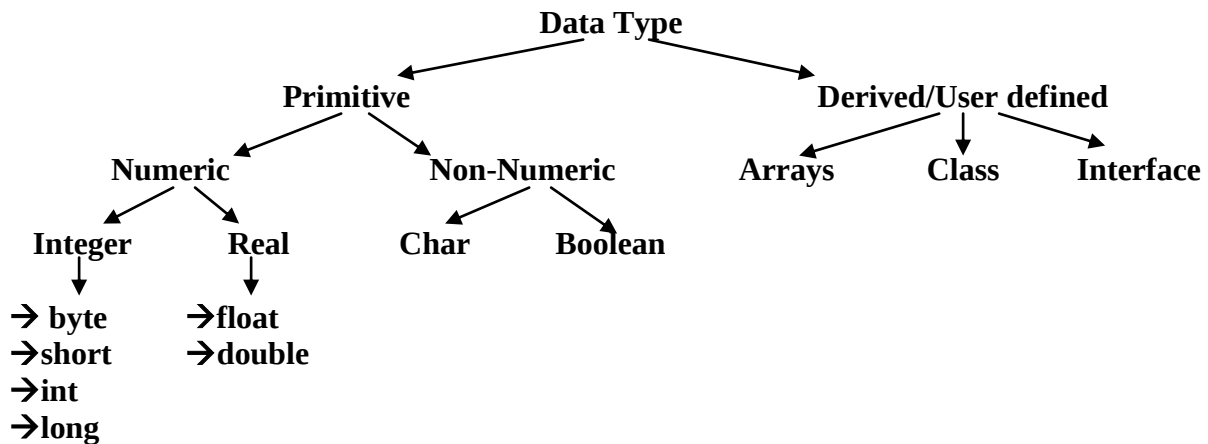
Here rollno and name are instance variables.

### 3. Static Variables:

- If a variable is declared inside the class and it is shared by all the objects of that class then that variables are referred to as “static” variables.
- These are called as class variables.
- The keyword “static” is used to declare static variables.  
Ex: static int a;
- Common memory is allocated for static variable for all the objects and this object share the value of the static variable.
- The static variables are called with class name not with object.

### DATA TYPE:

- Data type is used to specifies the type of storing values in a variable. Java supports different kinds of data types.
- For storing value in a variable, each variable in java must have a data type.
- Data type specifies the size and the range of values that can be stored in a variable.
- The following are the list of data types available in java.



### INTEGER DATA TYPES:

- Integer type can hold whole numbers like -50, 20, ... etc. The size of the value that can be store depends on the integer data type.
- Java supports 4 different types of integer data types. They are byte, short, int, long.
- Java doesn't support the concept of unsigned integers. The size and range of integer data types are listed below.

| TYPE  | SIZE    | RANGE                                                |
|-------|---------|------------------------------------------------------|
| Byte  | 1 byte  | $-2^7$ to $2^7-1$ (-128 to +127)                     |
| Short | 2 bytes | $-2^{15}$ to $2^{15}-1$ (-32768 to +32767)           |
| Int   | 4 bytes | $-2^{31}$ to $2^{31}-1$ (-2147483648 to +2147483647) |

|      |         |                         |
|------|---------|-------------------------|
| Long | 8 bytes | $-2^{63}$ to $2^{63}-1$ |
|------|---------|-------------------------|

When we want to declare a variable as long the value must be suffixed by 'e' or 'L'.

Ex: long a=23l;

#### REAL DATA TYPE:

1. Integer data can hold only whole numbers. Therefore to store fractional parts we use floating type data types.
2. Floating point precision values are double precision numbers. When we declare a variable with float data type by default the value treated as a double data type. So the value must be suffixed by 'f';

| TYPE   | SIZE    | RANGE                |
|--------|---------|----------------------|
| Float  | 4 bytes | 3.4E-38 to 1.7E+38   |
| Double | 8 bytes | 1.7E-308 to 3.4E+308 |

Floating point supports a special value known as "Nan" (Not a number).

#### CHARACTER DATA TYPE:

1. In order to store the alpha numeric character constants into memory java supports one data type i.e., char.
2. The character data type can hold only single character. It occupies 2 bytes of memory.
3. Java follows Unicode character set. Since java is an instrumental language.
4. The Unicode character set contains 65536 wide varieties of characters from different languages including ASCII.
5. The value of character variable must be enclosed within ' '.

Ex: char ch='A';

#### BOOLEAN DATA TYPE:

1. When we want to check conditions in the program execution the boolean data type is used.
2. The keyword boolean is used to declare a variable of boolean data type.
3. It takes only two possible values. Those are
  - a. True → when condition is true
  - b. False → when condition is false
4. The boolean data type occupies only one bit of memory. The result of a relational expression belongs to Boolean type.

Ex: boolean b;

#### NON-PRIMITIVE DATA TYPE:

These are derived data types constructed based on primitive data types. In java, we are having the following non-primitive data types. They are arrays, classes, interfaces.

#### Type Casting:

In java, when we assign a higher data type value to a lower data type, it loses some properties. This is called narrowing. If we assign a lower data type to a higher data type then it is called widening.

If an expression contains one integer type and one byte type and a short type and a long type, the result of that expression leads to an integer type value, unless we specify long with L.

By default the real type constant will be treated as double unless that constant succeeded by 'f'. Java performs type conversions automatically.

#### Explicit Type Casting:

Explicit type casting refers to modifying the type of the data temporarily for that expression.

Syntax: (data type) variable;

Ex: int a=6 , b=4;

float c = a / b;

Here, both a,b are of integer type. So the result will be 1. But we assign the result to a float variable. So the value of c will be 1.00000. In order to get the correct value one of either a or b must be of float type otherwise we need to cast either a or b into float temporarily.

Ex: int a=3, b=2;

float c=(float)a/b;

### **Standard Default Values:**

Every variable has a default value. If we don't initialize a variable when it is first created, Java provides default value to that variable automatically.

- |              |          |
|--------------|----------|
| 1. Byte      | (byte)0  |
| 2. Short     | (short)0 |
| 3. int       | 0        |
| 4. long      | 0L       |
| 5. float     | 0.0f     |
| 6. double    | 0.0d     |
| 7. char      | null     |
| 8. boolean   | false    |
| 9. reference | null     |

### **IMPORTANT QUESTIONS:**

1. Explain OOPs concepts.
2. Explain Features of Java
3. Explain Data Types in Java.

1. What are the applications of OOPs.
2. Explain command line arguments.
3. Explain JVM.
4. Differences between C and Java / C++ and Java.

## **UNIT – II**

### **OPERATORS AND EXPRESSIONS**

An Operator is a symbol which is used to perform arithmetic and logical manipulations on data. Generally Java has two types of operators. They are 1. Unary Operators 2. Binary Operators.

Unary Operator: If an operator has only one operand then it is called as Unary Operator.

Ex: ++a, a++, --a, a--,...

Binary Operator: If an operator has two operands then it is called as Binary Operator.

Ex: a=b, c=a+b,...

The following are the various operators available in Java.

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment and Decrement Operators
6. Conditional Operator
7. Bitwise Operators
8. Special Operators

### ARITHMETIC OPERATORS:

Arithmetic operators are used to construct mathematical expressions. These operators can operate on any built-in data types of Java. These operators cannot be operated on boolean type.

A=20, B=6

|   |                            |     |     |
|---|----------------------------|-----|-----|
| + | Addition or Unary Plus     | A+B | 26  |
| - | Subtraction or Unary Minus | A-B | 14  |
| * | Multiplication             | A*B | 120 |
| / | Division                   | A/B | 3   |
| % | Modulo Division            | A%B | 2   |

### RELATIONAL OPERATORS:

While writing programs there may be situations raised like compare two quantities and depending on their relation take certain decisions. These comparisons can be done with the help of relational operators. Java supports six relational operators. These operators gives only two results true or false.

A=20, B=6

|    |                          |          |       |
|----|--------------------------|----------|-------|
| <  | Is less than             | A<B      | FALSE |
| <= | Is less than or equal    | A<=B     | FALSE |
| >  | Is greater than          | A>B      | TRUE  |
| >= | Is greater than or equal | A>=B     | TRUE  |
| == | Is equal to              | A+B==B+A | TRUE  |
| != | Is not equal to          | A!=B     | TRUE  |

When arithmetic expressions are used on either side of a relational operator, the arithmetic expression will be evaluated first and then the results are compared.

## LOGICAL OPERATORS:

Logical operators are used when we want to form compound conditions by combining two or more relations. Java has three logical operators.

|            |             |              |              |
|------------|-------------|--------------|--------------|
| &&         | Logical AND |              |              |
|            | Logical OR  |              |              |
| !          | Logical NOT |              |              |
| Operator-1 | Operator-2  | Op-1 && Op-2 | Op-1    Op-2 |
| TRUE       | TRUE        | TRUE         | TRUE         |
| TRUE       | FALSE       | FALSE        | TRUE         |
| FALSE      | TRUE        | FALSE        | TRUE         |
| FALSE      | FALSE       | FALSE        | FALSE        |

## ASSIGNMENT OPERATORS:

Assignment operators are used to assign the value of an expression to a variable. Java supports a set of shorthand assignment operators which are used in the form

**Variable operator = expression ;**

| Statement with simple assignment operator | Statement with shorthand operator |
|-------------------------------------------|-----------------------------------|
| <b>A = A + 1</b>                          | <b>A+=1</b>                       |
| <b>A = A - 1</b>                          | <b>A-=1</b>                       |
| <b>A = A*(N+1)</b>                        | <b>A*=N+1</b>                     |
| <b>A = A/(N+1)</b>                        | <b>A/=N+1</b>                     |
| <b>A = A % B</b>                          | <b>A%=B</b>                       |

1. What appears on LHS is need not be repeated on RHS.
2. The statement is more concise and easier to read.
3. Provides more efficient code.

## INCREMENT AND DECREMENT OPERATORS:

1. These Operators are also called as Unary Operators.
2. These operators are used to increment one value of variable and decrement operator is used to decrease one value of a variable.
3. These operators are available in two forms.
  - a. Prefix Operator: Operator comes before the operand. Ex: ++a; --a;
  - b. Postfix Operator: Operator comes after the operand. Ex: a++; a--;
4. In prefix operation first increment/decrement will be done later assignment will be done.
5. In postfix operation first assignment will be done later increment/decrement will be done.

| Operator | Description |
|----------|-------------|
| ++       | Increment   |
| --       | Decrement   |

## CONDITIONAL OPERATORS:

1. Conditional operator is used to construct conditional expressions of the form **exp1?exp2:exp3**
2. The operator ?: works as follows.
3. Exp1 is evaluated first. If it is true then the exp2 will be evaluated otherwise exp3 will be evaluated.
4. Only one expression is evaluated.
5. Conditional operator is short form of If...Else statement.

Example:

A=50

B=25

X=(A>B)?A:B;

In the above example X is assigned with 50.

## BITWISE OPERATORS:

1. Bitwise operators manipulate the data at bit level. The following are the list of bitwise operators.
2. The bitwise operator 'AND' is "1" if the corresponding bits of operands both are 1.
3. The bitwise operator 'OR' is "1" if any one of the corresponding bit is 1.

|     |                            |
|-----|----------------------------|
| &   | Bitwise AND                |
|     | Bitwise OR                 |
| ^   | Bitwise EXOR               |
| >>  | Shift Right                |
| <<  | Shift Left                 |
| >>> | Shift Right with Zero Fill |
| ~   | One's compliment           |

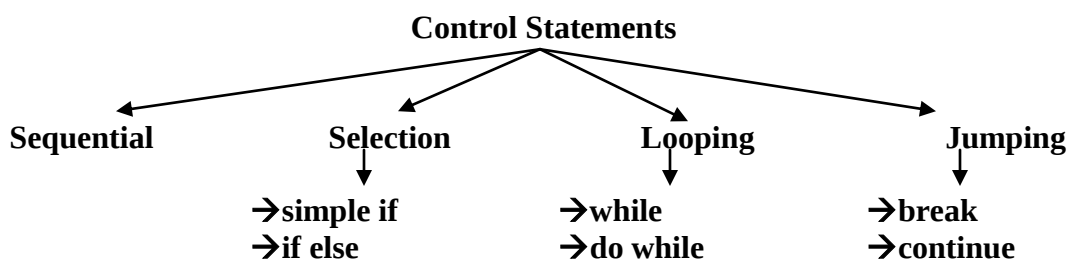
## SPECIAL OPERATORS:

|            |                                                                                                                                                               |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| .          | This operator is used to access properties and methods of an object.                                                                                          |
| Instanceof | This operator is used to check whether the object belongs to class or not. The result of this operator may be true or false and belongs to boolean data type. |

## CONTROL STATEMENTS IN JAVA:

Every Java program consists of number of statements those are executed one after other. The control statements are the statements which control the flow of execution of the statements.

Sometimes we need to choose group of statements among several alternative groups or sometimes we are interested to repeat a group of statements several number of times.



→nested if  
→switch

→for  
→for each

→labeled

**Sequential control statements:** These are the group of statements executed one after the other till end.

Syntax: statement-1;  
statement-2;  
.....  
.....  
statement-n;

Here statement-2 will be executed after completion of statement-1 and statement-3 after statement-2 and so on.  
**Selection Control Statements:** These control statements are also called as branching statements. In this type of control statements, one group of statements are executed among several alternative groups.

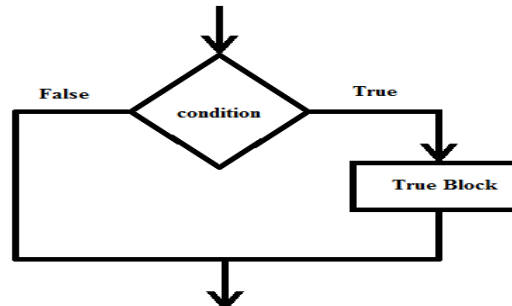
### 1) simple if

This is a primitive selection control statements. It is used in decision making on the statements required or not on the execution.

Syntax: if( condition)  
{  
Statement;  
}

The statements inside the If block will be executed only if the condition is “true”, otherwise the statements inside the if block are skipped from execution.

Flow Chart:



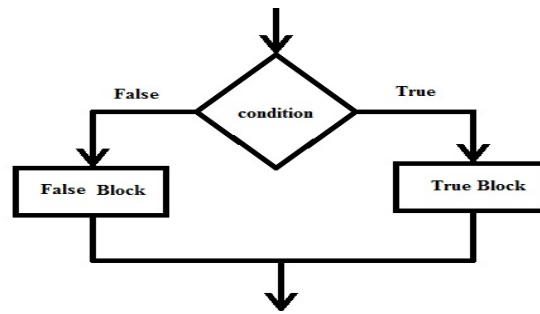
### 2) if else statement:

This control statement provide alternative statements those are executed when the condition is false.

Syntax: if( condition)  
{  
True Block Statements  
}  
else  
{  
False Block Statements  
}

In this statement first the condition will be checked. If the condition is true, then true block statements will be executed otherwise false block (else block) statements will be executed.

FLOW CHART:



### Nested if

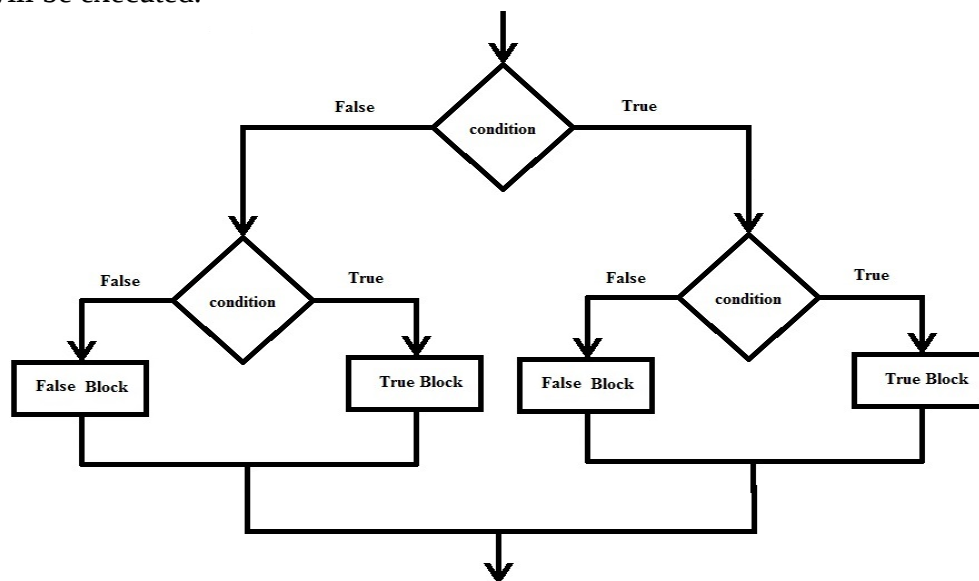
If an if control statement is placed inside another if statement, then that control statement is called Nested-if statement.

```

Syntax: if(condition1)
{
 if(condition2)
 {
 Statements-1
 }
 else
 {
 Statements-2
 }
}
else
{
 Statements-3
}

```

If outer condition of the outer if block is true, then the inner if block will be executed. If it is also true, statement-1 will be executed otherwise statement-2 will be executed. If the condition at outer if block is false then the statement-3 will be executed.



### Else-if ladder

This control statement searches for alternatives if one condition is failed. In this control statement instead of placing an if block inside another if block we place the if block inside the else block.

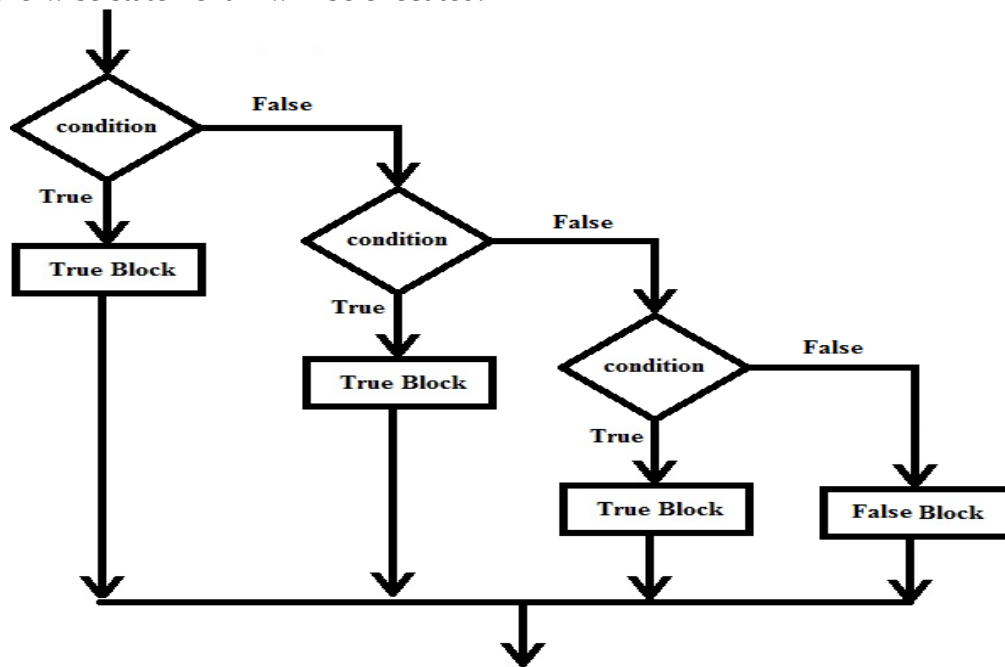
Syntax: if(condition-1)

```

{
 Statement - 1
}
else if(condition-2)
{
 Statement - 2
}
else if(condition3)
{
 Statement - 3
}
else
{
 Statement - 4
}

```

The statement-1 will be executed if the condition is true. The statement-2 will be executed if condition-1 is false and condition-2 is true. The statement-3 will be executed if both condition 1 and 2 is false and condition-3 becomes true, otherwise statement-4 will be executed.



### Switch case and Break

- ➔ This control statement is used as an alternative to else-if ladder. This statement is executed based on a value or result of an expression.
- ➔ Based on the value passed to switch, the corresponding match case will be executed. If no matching case is found the default case will be executed.

Syntax:   switch( value/expression )  
           {  
           case value-1: statements;  
                           break;  
           case value-2: statement;  
                           break;  
           case value-3: statements;  
                           break;  
                           .  
                           .  
           default:       statements;

}

Rules for using Switch case:

1. There should be only one value passed to switch conditions and ranges of values were not considered.
2. There should contain a space between case and value.
3. Each case should be terminated by a break statement, otherwise the consecutive cases will also be executed along with successful case.

### LOOPING CONTROL STATEMENTS:

Looping refers to the representation of the same statement until certain number of times. Most commonly looping control statements based on a condition to evaluate the block of statements.

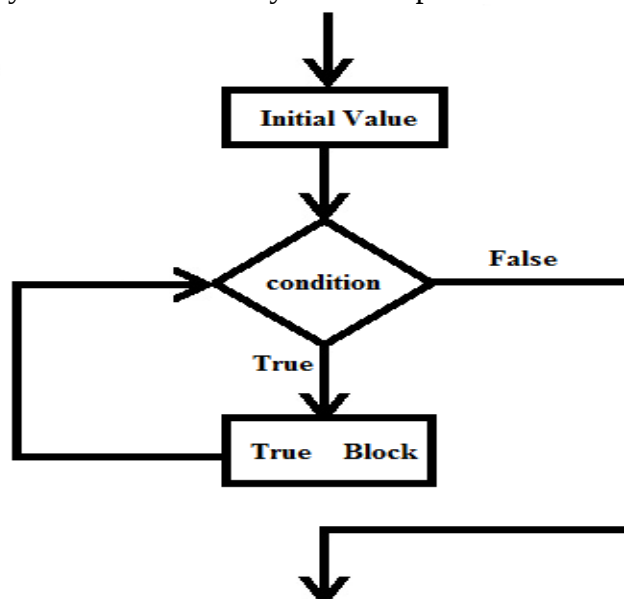
#### WHILE LOOP:

1. While statement is one of the looping control statement.
2. While statement is also known as “Entry Control Loop”.
3. In this controller first check the condition, if the condition is true then controller moves to while block and executes the statements.
4. After completing execution of last statement controller again checks the condition. This process will be repeated until the condition is false.

Syntax:        Initial value;  
              while (condition)  
              {  
                  Block of Statements  
                  ...  
                  Increment/decrement;  
              }

Here, initialization is starting value of the loop, it is done only once. Condition is an expression having relational and logical operators which results in Boolean quantity.

Increment/Decrement done every time when the body of the loop is executed.



#### DO-WHILE LOOP:

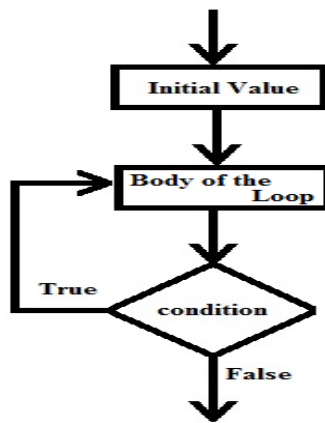
1. Do While loop is also called as “Exit Control Loop”.

2. In this first body of the loop is executed and then condition is checked.
3. If the condition is true then the body of the loop is executed.
4. When the condition becomes false then it will exit from the loop.

Syntax: initialization;

```
do
{
 Statements
 ...
 Increment/decrement

} while(condition);
```



#### **DIEEFERENCES BETWEEN WHILE AND DO-WHILE:**

##### **WHILE**

1. While loop is also called as Entry Control Loop.
2. In While Loop control first checks the condition then executes the while block statements.
3. While loop is a determinate looping control statement.
4. Statements in while loop may or may not get executed.
5. While(condition)
 

```
{
 Block of statements;
 Increment/decrement
 }
```

##### **DO-WHILE**

1. Do-While loop is also called as Exit Control Loop.
2. In Do-while loop control first executes the while block statements and then checks the condition.
3. Do while loop is an under determinate looping control statement.
4. Statements in do while loop get executed at least once.
5. Do
 

```
{
 Block of statements;
 Increment/decrement
 }while(condition);
```

##### **FOR**

This is a special kind of loop which contains all the looping characteristics at one line. For is a looping control statement in which three expressions were present.

1. Initialization
2. Condition
3. Increment/Decrement

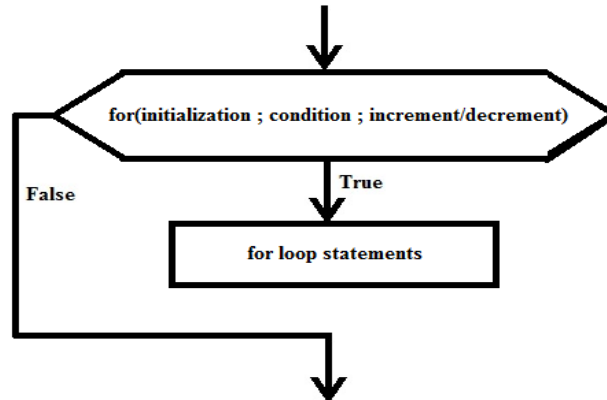
The three expressions present in the for loop are separated by using semicolon (;). Even in the absence of any expression you need to place semicolon.

**Syntax:** for( initialization ; condition ; increment/decrement)

```

{
 Statements;
}

```



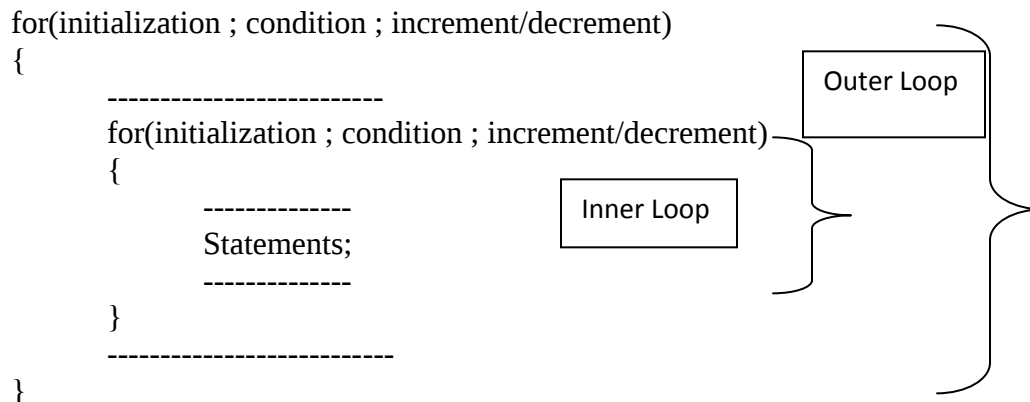
Initialization part specifies the value of the variable, condition specifies how many times the statements are to be executed and increment/decrement part increases/decreases iterative variable.

1. In first iteration for loop executes initialization part and checks the condition.
2. If the condition is true it executes the block of statements then performs increment/decrement.
3. After performing increment/decrement it checks the condition, if it returns true executes block of statements otherwise terminate the loop.

### NESTED LOOPS:

One for statement within another for statement is called as Nested for loop.

Syntax:



### FOR - EACH LOOP:

Generally this looping statement is used with arrays. We know that an array is a sequential collection of homogeneous data items and each one is represented with an index number i.e.,  $a[0]$  is the 1<sup>st</sup> element,  $a[1]$  is 2<sup>nd</sup> element, ...  $a[n-1]$  is the last element.

The arguments passed to `main()` are called command line arguments and are belongs to string data type and stored in the array named “args” to print all the command line arguments, we use the general for loop as follows.

```

for(i=0 ; i<args.length ; i++)
{
 System.out.println(args[i]);
}

```

The same statement can be represented with for each loop as follows.

```
for(String s: args)
{
 System.out.println(s);
}
```

Syntax:

```
for(datatype identifier : array name)
{
 //use identifier instead of individual elements
}
```

### **CONTINUE:**

This keyword is quite opposite to break. The occurrence of the keyword continue in the loop indicates that skip the following statements and move to next iteration, which means the lines appeared after the continue will be skipped from execution.

### **LABELLED LOOPS:**

If a break statement appears in inner loop will break the execution of only inner loop. But the outer loop remains in execution.

```
Loop1:for(i=1;i<=10;i++)
{
 Loop2:for(j=1;j<5;j++)
 {
 if(i%j==0)
 break Loop1;
 }
}
```

## **CLASSES, OBJECTS AND METHODS**

### **INTRODUCTION:**

- Java is a true object-oriented language and therefore the underlying structure of all java program is classes.
- Classes create objects and objects use methods to communicate between them.
- In java the data items are called fields and the functions are called methods.
- Encapsulation, Inheritance and Polymorphism are the basic OOP concepts.

### **DEFINING CLASSES, ADDING VARIABLES AND METHODS:**

- A class is a user defined data type.
- Once the class is defined, we can create variables of that type using declarations that are similar to the basic type declarations.
- These variables are termed as instances of classes which are actual objects.

```
class class-name [extends superclass-name]
{
 Filed declarations;
 Method declarations;
}
```

- Everything written inside the square brackets is optional.

5. Class-name and super-class name are any valid java identifiers.
6. The keyword extends indicates that the properties of the superclass-name class are extended to the class-name class.
7. Data is encapsulated in a class by placing fields inside the body of the class definition.
8. These variables are called instance variables because they are created whenever an object of the class is instantiated.

```
class student
{
 int rno;
 int marks;
 String name;
}
```

9. A class with only data fields has no life. The objects created to that class cannot respond to any messages.
10. Methods are declared and defined inside class, only after the variable declaration.
11. The general form of a method declaration is as follows:

```
Type method-name (parameter list)
{
 Method-definition;
}
```

12. Method declarations have four parts.
  - a. Name of the method
  - b. Return type of the method
  - c. Parameter list
  - d. Method Definition

```
class Student
{
 int rno;
 int marks;
 String name;
 void displayStudent()
 {
 System.out.println("Roll Number:"+rno);
 System.out.println("Name:"+name);
 System.out.println("Marks:"+marks);
 }
}
```

### **CREATING OBJECTS:**

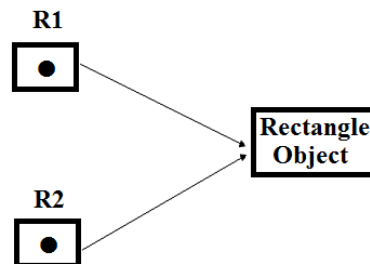
1. Creating an object is also known as instantiating object.
2. In java new operator is used to create an object to the specified class and it returns a reference to that object.
3. Creating an object in java is as follows:

```
class-name object-name; // declaring the object
object-name = new class-name(); // initiating the object
```
4. we can combine the above statements as follows.

```
class-name object-name = new class-name();
```
5. It is important to note that each and every object has a separate copy of instance variables of its class.

- Any changes made to the variables of one object have no effect on the variables of another object.
- We can also create two or more references to the same object.

```
class Rectangle
{
 int lenght;
 int width;
 int area()
 {
 return lenght*width;
 }
}
Rectangle R1 = new Rectangle();
Rectangle R2 = R1;
```



#### ACCESSING CLASS MEMBERS:

- We cannot access the instance variables and the methods directly.
- We must use the concerned object and the dot operator to access instance variables and methods as shown below.

```
Objectname.variable-name = value;
Objectname.method-name(parameter-list);
class Rectangle
{
 int lenght;
 int width;
 int area()
 {
 return lenght*width;
 }
}
class RectangleDemo
{
 public static void main(String args[])
 {
 Rectangle r1=new Rectangle();
 Rectangle r2=new Rectangle();
 r1.lenght=10;
 r1.width=20;
 r2.lenght=15;
 r2.width=25;
 System.out.println("Area of first rectangle:"+r1.area());
 System.out.println("Area of second rectangle:"+r2.area());
 }
}
```

- In the above example r1 and r2 are two objects of class Rectangle.
- Memory for Instance variables of objects r1 and r2 are created separately.

5. Changes made to the variables of object r1 will not affect variables of object r2.

## CONSTRUCTORS

- A constructor is a special type method that is used to initialize an object.
- Constructors have the same name of class.
- They do not have any return type, even void also not used as return type for constructors.
- Constructors return the instance of the class itself.
- Like general methods a constructor need not be called by using object with dot operator.
- A constructor will be invoked automatically whenever an object to a class is created.

```
class Rectangle
```

```
{
 int length;
 int width;
 Rectangle(int x,int y)
 {
 length=x;
 width=y;
 }
 int area()
 {
 return length*width;
 }
}
```

```
class RectangleDemo
```

```
{
 public static void main(String args[])
 {
 Rectangle r1=new Rectangle(10,20);
 Rectangle r2=new Rectangle(15,25);
 System.out.println("Area of Rectangle1="+r1.area());
 System.out.println("Area of Rectangle2="+r2.area());
 }
}
```

- The constructor Rectangle defined in the above program is **parameterized constructor**. We have specified input to the constructor.
- If we want the constructor to automatically initialize the object variables with default values at the time of object initialization the **default constructor** is used.

```
class Rectangle
```

```
{
 int length;
 int width;
 Rectangle()
 {
 Length=0;
 Width=0;
 }
 Rectangle(int x,int y)
 {
 length=x;
 width=y;
 }
}
```

```

 int area()
 {
 return lenght*width;
 }
 }
class RectangleDemo
{
 public static void main(String args[])
 {
 Rectangle r1=new Rectangle();
 Rectangle r2=new Rectangle(15,25);
 System.out.println("Area of Rectangle1="+r1.area());
 System.out.println("Area of Rectangle2="+r2.area());
 }
}

```

In the above example Rectangle class contains two constructors one has parameters called parameterized constructor and another doesn't have any parameters called default constructor.

### **METHOD OVERLOADING:**

1. Java uses method overloading mechanism to implement compile time polymorphism.
2. Using method overloading we can create methods that have the same name but different parameter lists and different definitions.
3. When we call a method in java, first it will look for that method then number and type of parameters to decide which method to execute.
4. The point to remember here is that method overloading will be done based on the number and type of parameters only not on the return type of the method.

```

class Rectangle
{
 double area(int r)
 {
 return 3.141*r*r;
 }
 int area(int lenght,int width)
 {
 return lenght*width;
 }
}
class RectangleDemo
{
 public static void main(String args[])
 {
 Rectangle r1=new Rectangle();
 System.out.println("Area of Cirlce="+r1.area(10));
 System.out.println("Area of Rectangle="+r1.area(10,20));
 }
}

```

### **STATIC MEMBERS:**

1. Variables and methods declared inside a class are called instance variables and instance methods because every time an object is created a new copy of variables and methods created.
2. These variables and methods are accessed by using object name and dot operator only.
3. If a situation occurs like this, a variable and method that shares all objects.
4. Such variables and methods are declared as static.

```
static int count;
static int max(int x,int y);
```

5. Static members are associated with class rather than individual objects.
6. Static variables and methods are referred as class variables and class methods.
7. Static variables and methods are called without using the objects.
8. Java class library contains a large number of class methods.
9. Static methods are called using class names.

```
class StaticDemo
{
 static float multiplication(float x,float y)
 {
 return x*y;
 }
 static float divide(float x,float y)
 {
 return x/y;
 }
}
class StaticMethod
{
 public static void main(String args[])
 {
 float a=StaticDemo.multiplication(10,14);
 float b=StaticDemo.divide(a,2.0);
 System.out.println("b="+b);
 }
}
```

The following are the restrictions of static methods:

- a. They can only call other static methods.
- b. They can only access static data.
- c. They cannot refer to this or super.

### NESTING OF METHODS:

- ☉ A method can be called by using only its name by another method of the same class.
- ☉ This is known as nesting of methods.

```
class Biggest
{
 int a,b;
 Biggest(int x,int y)
 {
 a=x;
 b=y;
 }
 int big()
 {
 if(a>b)
 return a;
 else
 return b;
 }
 void display()
 {
```

```

 int large=big();
 System.out.println("Largest is :"+large);
 }
}
class NestingDemo
{
 public static void main(String args[])
 {
 Biggest b1=new Biggest(10,20);
 b1.display();
 }
}

```

### VISIBILITY CONTROLS:

1. The variables and methods of a class are visible anywhere in the program.
2. If you want to restrict the access of certain variables and methods from outside the class, you can achieve this by applying visibility modifiers to the instance variables.
3. The visibility modifiers are also known as access modifiers.
4. Java provides three types of visibility modifiers: **public, private, protected**.

### Public Access:

- a. If a variable or method is visible to the entire class in which it is defined then it is in public access.
- b. A variable or method declared as public has the widest possible visibility and accessible everywhere.

```

public int number;
public void sum()
{

}

```

### Private Access:

- a. Private fields have the highest degree of protection.
- b. These fields can be accessed only in their own class.
- c. These variables cannot be inherited.
- d. A private method is similar to final method.

**Protected Access:**

- a. The visibility level of protected field lies between the public and friendly access.
- b. Protected fields visible not only to all classes and subclasses in the same package but also to the subclasses in other packages.
- c. Non subclasses in other packages cannot access the protected members.

## UNIT-III INHERITANCE

1. Java supports reusability with inheritance. Java classes can be reused in number of ways. The mechanism of deriving a new class from an existing class is called inheritance.
2. The old class is known as the base class or parent class or super class.
3. The new class is known as derived class or child class or sub class.
4. Inheritance allows sub class to inherit all the variables and methods of their parent class.
5. A subclass will be defined as follows.

```
class subclass-name extends superclass-name
{
 Variables
 Methods
}
```

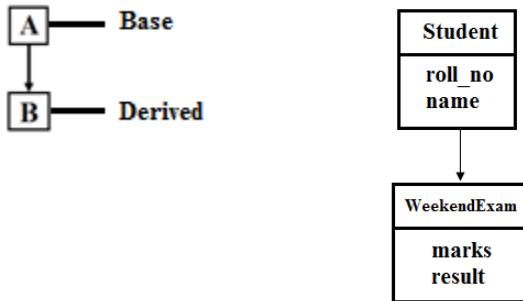
6. The keyword extends specifies that properties of super class are extended to the sub class.
7. The sub class will now contain all the properties of super class in addition to its own properties.

**TYPES OF INHERITANCES:**

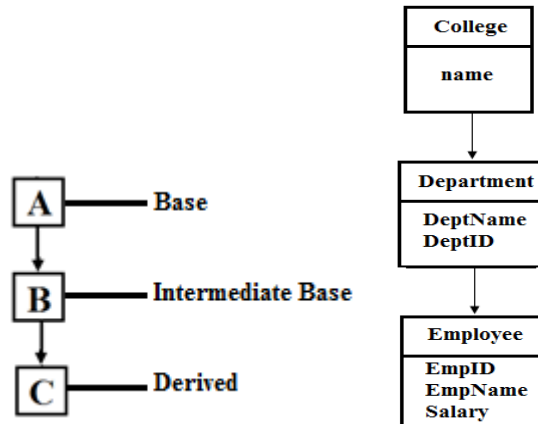
There exist parent and child relationship between classes in inheritance. The following are the various types of inheritances in Java.

- Single Inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Multilevel Inheritance
- Hybrid Inheritance

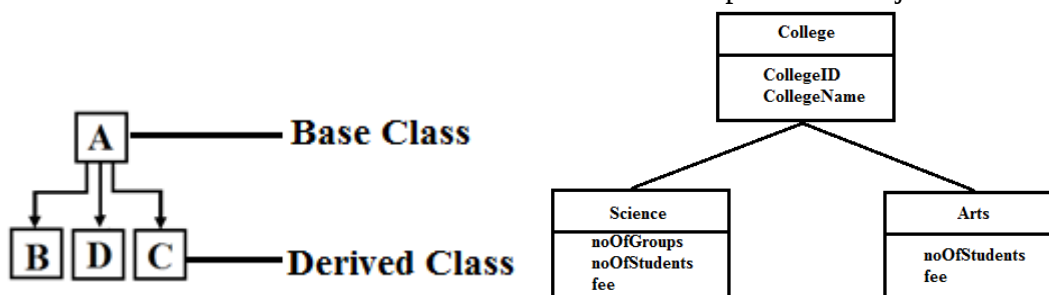
**Single Inheritance:** A Derived class with only one base class is called Single Inheritance. In this type of inheritance the child class contains only one parent class.



**Multilevel Inheritance:** The mechanism of deriving a class from another derived class is known as Multilevel Inheritance. In this type of inheritance one derived class acts as base class for another derived class. This base class is known as intermediate base class.

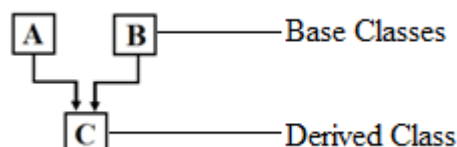


**Hierarchical Inheritance:** Multiple Derived classes with a single base class is called Hierarchical Inheritance. This type of inheritance is used to maintain a tree structured relationship between objects.

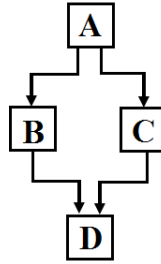


**Multiple Inheritance:** A Derived class with multiple base classes is called Multiple Inheritance. Java doesn't support extendibility for more than one class. A class cannot be extended by more than one class. So the multiple inheritance cannot be achieved directly.

We can achieve multiple inheritance through "interfaces" concept.



**Hybrid Inheritance:** The combination of any two or more types of above inheritances is known as Hybrid Inheritance.



### **OVERRIDING METHODS:**

1. In Inheritance, we may face a situation that the derived class has the same method signature as the base class method.
2. In this situation, the derived class contains two methods with same signature, one is written in derived class and another is extended from base class.
3. When that method is called the method defined in the sub class will be invoked and executed instead of the one in the super class.

```
import java.lang.*;
class Bird
{
 void fly()
 {
 System.out.println("A Bird Can Fly");
 }
}
class Crow extends Bird
{
 void fly()
 {
 System.out.println("A Crow Can Fly");
 }
}
class Override
{
 public static void main(String args[])
 {
 Bird b=new Bird();
 b.fly();
 Crow c=new Crow();
 c.fly();
 Bird b1=new Crow();
 b1.fly();
 }
}
A Bird Can Fly
A Crow Can Fly
A Crow Can Fly
```

### **DIFFERENCES BETWEEN OVERLOADING AND OVERRIDING**

| OVERLOADING                          | OVERRIDING                                 |
|--------------------------------------|--------------------------------------------|
| 1. Overloading generally implemented | 1. Overriding generally implemented within |

within a single class.

2. In overloading methods have same name with different number of arguments and type of arguments.
3. The relevant method will be executed based on number and type of arguments.
4. As constructors are also methods they can also be overloaded.

```
class Demo
{
 Demo()
 {

 }
 Demo(int x)
 {

 }
}
```

inheritance.

2. In overriding methods have same name and same number of arguments and type of arguments.
3. The relevant method will be executed based on object. If it is base class object then the method of base class will be executed and if it is derived class object then the method of derived class will be executed.
4. Constructors cannot be over-riden.

```
class Demo
{
 void display()
 {

 }
}
class Demo1 extends Demo
{
 void display()
 {

 }
}
```

### **FINAL VARIABLES AND METHODS:**

1. All variables and methods can be overridden by default in subclass.
2. If we wish to prevent sub classes from overriding the members of super class, we can declare them as final using “final” keyword.

```
final double pi=3.1412;
```

```
final void display()
{

}
```

The final variables and methods cannot be altered anyway.

### **FINAL CLASS:**

1. Sometimes we may like to prevent the class being further subclassed for security reasons. A class which cannot be subclassed is called as final class.

```
final class Demo
{

}
```

```

 }
 final class Demo1 extends Demo
 {

 }

```

Any attempt to inherit these classes will lead to an error and compiler will not allow it.

### **finalize() Method:**

1. A constructor method is used to initialize an object when an object is created and this process is called initialization.
2. Similarly, Java supports a concept called finalize.
3. We know that java at runtime has an automatic garbage collection system. It automatically frees up the memory resources used by the objects.
4. But non object resources remain in the system. In order to free these resources, we must use finalize( ) method. This process is quite opposite to initialization.
5. It is similar to destructors in C++.

### **ABSTRACT METHODS AND ABSTRACT CLASSES:**

1. Making a method final ensures that the method is not redefined in the subclass. Java also supports something that is exactly opposite to this.
2. We can indicate that a method must always be redefined in a subclass. Thus making overriding mandatory.

```

abstract class Demo
{
 abstract void display();

}

```

3. When a class contains one or more abstract methods, it should also be declared as abstract.
4. We cannot use abstract class to instantiate directly.
5. Abstract methods of an abstract class must be defined in its sub class.
6. We cannot declare abstract constructors and abstract static methods.

```

abstract class A
{
 abstract void display();
}
class B extends A
{
 void display()
 {
 System.out.println("Redefined Method");
 }
}
class AbstractDemo
{

```

```

 public static void main(String args[])
 {
 B objb=new B();
 objb.display();
 }
}

```

### **VISIBILITY CONTROLS:**

Sometimes we may want to restrict the access of variables and methods by the objects. For that purpose we may use some keywords which modifies the behavior of fields and methods of a class. These are called as visibility modifiers.

The modifier provides a mechanism to control levels of accessing the fields. So, the visibility modifiers are also called as “access specifiers”. There are three basic types of access specifiers. Those are

1. Private
2. Public
3. Protected

#### **PUBLIC:**

These fields are declared using the keyword **public**.

```

Public int x=50;
Public void display()
{

}

```

The public variables and methods can be accessed anywhere regardless of packages and class. The accessing locations of public variables and methods are

1. Same class
2. Sub class
3. Other class in the same package
4. Sub class in other package
5. Other class in the other package

#### **PRIVATE:**

These fields are declared using the keyword **private**.

```

Private int x=50;
Private void display()
{

}

```

The private variables and methods can only be accessed within the same class. The accessing locations of private variables and methods are

1. Same class

#### **PROTECTED:**

These fields are declared using keyword **protected**.

```

Protected int x=30;
Protected void display()
{

}

```

}

The protected variables and methods can be accessed only within the same class, other classes in the same package and its sub classes regarding the package. The accessing location of protected variables and methods are

1. Same class
2. Sub class
3. Other classes in the same package
4. Sub classes in other packages.

#### **PRIVATE PROTECTED:**

It is a special kind of access specifier.

Private protected int x=20;

Private protected void display( )

{

-----

-----

}

The private protected variables and methods can be accessed within the same class and in sub classes regardless of package. The accessing locations of private protected variables are

1. Same class
2. Sub class
3. Sub classes in other package

#### **ARRAY**

1. An array is a collection of homogeneous data items which can store in continuous memory location and having the same name.
2. An array is a group of contiguous or related data items that share a common name. A particular value is indicated by writing a number called index number or subscript in brackets after the array name.
3. There are three types of arrays,
  - a. One dimensional array
  - b. Two dimensional array
  - c. Multi dimensional array

There are three steps in creating an array. They are

1. Declaration of array
2. Creation of memory locations
3. Initialization of array

**Declaration of Array:** An array can be declared one of the following two ways.

Datatype array-name[ ];

Datatype[ ] array-name;

Example: int a[ ]

int[ ] a;

**Creation of Memory Locations:** Memory for an array can be created as follows.

```
Array-name=new datatype[size];
```

Example:    a=new int[10];

We can also combine the declaration and creating memory as follows.

```
Datatype array-name[]=new Datatype[size];
```

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

**Initialization of Array:** An array can be initialized as follows.

```
Datatype array-name[]={ list of values };
```

Example: int a[ ]={10,20,30,40};

Java creates arrays starting with the subscript of 0 and ends with a value one less than the size specified.

Java protects arrays from overruns and under runs. Trying to access an array bound its boundaries will generate an error message.

In java arrays will be treated as objects. So the assignment of one array to another array is legal.

```
int a[]={1,2,3,4,5};
```

```
int b[];
```

```
b=a;
```

length is a variable that can store length of array.

a.length returns the number of elements in the array a.

```
import java.lang.*;
import java.util.*;
class ArrayDemo
{
 public static void main(String args[])
 {
 Scanner s=new Scanner(System.in);
 System.out.println("How many elements you want to read:");
 int n=s.nextInt();
 int a[]=new int[n];
 System.out.println("Enter "+n+" elements:");
 for(int i=0;i<n;i++)
 a[i]=s.nextInt();
 System.out.println("Array elements are:");
 for(int i=0;i<a.length;i++)
 System.out.println(a[i]);
 }
}
```

**TWO DIMENSIONAL ARRAYS:**

1. If an array has two sub scripts, then it is called as two dimensional array. These are otherwise called as matrix arrays or table arrays.
2. These are used to organize in terms of rows and columns.
3. Each element in two dimensional array is identified by name of the array, its row index and its column index.

|   | 0       | 1       | 2       | 3       |
|---|---------|---------|---------|---------|
| 0 | a[0][0] | a[0][1] | a[0][2] | a[0][3] |
| 1 | a[1][0] | a[1][1] | a[1][2] | a[1][3] |
| 2 | a[2][0] | a[2][1] | a[2][2] | a[2][3] |
| 3 | a[3][0] | a[3][1] | a[3][2] | a[3][3] |

#### CREATION OF 2-D ARRAY:

We can create a 2D array as follows.

```
datatype array-name[][];
```

or

```
datatype[][] array-name;
```

Example:

```
int a[][];
```

```
int[][] a;
```

#### Creating Memory Locations:

```
array-name = new datatype[row-size][column-size];
```

```
a=new int[3][3];
```

In the above example, the array a has given memory for 9 elements which are organized as 3 rows and 3 columns. The first element in the array will be a[0][0] and the last element a[2][2].

```
import java.lang.*;
import java.util.*;
class MatrixMultiplication
{
 public static void main(String args[])
 {
 Scanner s=new Scanner(System.in);
 System.out.println("First matrix dimensions:");
 int r1=s.nextInt();
 int c1=s.nextInt();
 System.out.println("Second matrix dimensions:");
 int r2=s.nextInt();
 int c2=s.nextInt();
 if(c1!=r2)
 {
 System.out.println("Multiplication Not Possible");
 }
 else
 {
 int a[][]=new int[r1][c1];
 int b[][]=new int[r2][c2];
 int c[][]=new int[r1][c2];
 System.out.println("Enter first matrix elements:");
```

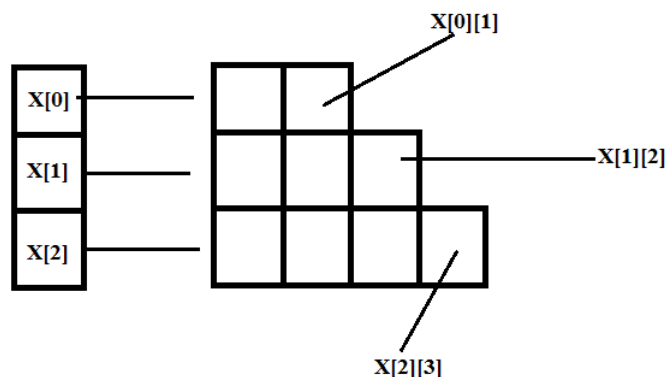
```

for(int i=0;i<r1;i++)
for(int j=0;j<c1;j++)
 a[i][j]=s.nextInt();
System.out.println("Enter second matrix elements:");
for(int i=0;i<r2;i++)
for(int j=0;j<c2;j++)
 b[i][j]=s.nextInt();
for(int i=0;i<r1;i++)
{
 for(int j=0;j<c2;j++)
 {
 c[i][j]=0;
 for(int k=0;k<c1;k++)
 {
 c[i][j]+=a[i][k]*b[k][j];
 }
 }
}
System.out.println("Resultant Matrix Is:");
for(int i=0;i<r1;i++)
{
 for(int j=0;j<c1;j++)
 {
 System.out.print(c[i][j]);
 }
 System.out.println();
}
}
}
}

```

#### VARIABLE LENGTH ARRAY or JAGGED ARRAY

1. Java treats multidimensional array as array of arrays. It is possible to declare a two-dimensional array as follows.  
 Int x[ ][ ]=new int[3][ ];  
 X[0]=new int[2];  
 X[1]=new int[3];  
 X[2]=new int[4];
2. These statements create a two-dimensional array as having different lengths for each row as shown below.



#### STRINGS:

1. Strings represent a sequence of characters.
2. You can represent a string as character array.
3. In Java, Strings are class objects and implemented using two classes, namely **String** and **StringBuffer**.
4. A Java string is an instantiated object of the string class. Java strings are more reliable and predictable.
5. A Java string is not a character array and is not NULL terminated.
6. Strings are declared and created as follows.  
`String s;  
s="Hello";  
s=new String("Hello");`
7. You can combine above statements as follows.  
`String s="Hello";  
String s=new String("Hello");`
8. Like arrays it is possible to get the length of a string using `length()` method of string class.  
`int len=s.length();`
9. Java strings are concatenated using `+` operator.  
`String s1="Hello";  
String s2="World";  
String s=s1+s2;`  
Here `s1` and `s2` are Java Strings containing string constants.
10. Java Strings are immutable. Immutable means their contents cannot be modified. `StringBuffer` class objects are mutable, so they can be modified.
11. Methods that directly manipulate data of the object are not available in `String` class. Such methods are available in `StringBuffer` class.

### STRING CLASS METHODS:

The string class defines a number of methods that allow us to accomplish a variety of string manipulation tasks.

1. `String toLowerCase()`  
Converts all characters of the string into lower case, and returns that lower-cased string.  
`String s="HELLOWORLD";  
String s1=s.toLowerCase();`
2. `String toUpperCase()`  
Converts all characters of the string into upper case, and returns that upper-cased string.  
`String s="helloworld";  
String s1=s.toUpperCase();`
3. `String replace(char c1,char c2)`  
Replaces the occurrences of character `c1` by character `c2`, and returns the modified string.  
`String s="HelloWorld";  
String s1=s.replace('l','L');`
4. `P.toString()`  
Create a string representation of object `P`.
5. `String trim()`  
Removes spaces from the beginning and ending of a string. This method does not remove the spaces in the middle of the string.

```
String s=" HelloWorld ";
```

```
String s1=s.trim();
```

6. Boolean equals(String s)

Returns true if two strings are same, otherwise false. This method is case sensitive.

```
String s1="Hello";
```

```
String s2="Hello";
```

```
boolean b=s1.equals(s2);
```

7. Boolean equalsIgnoreCase(String s)

Returns true if two strings are same, otherwise false. This method is case insensitive.

```
String s1="hello";
```

```
String s2="Hello";
```

```
boolean b=s1.equals(s2);
```

8. int length()

Returns the length or number of characters of a string.

```
String s1="Hello";
```

```
int len=s1.length();
```

9. char charAt(int i)

Returns the character at the specified location i.

```
String s1="Hello";
```

```
char ch=s1.charAt(4);
```

10. int compareTo(String s)

Used to compare two strings and to know which string is bigger or smaller. This method returns three possible values.

```
String s1="Hello";
```

```
String s2="hello";
```

```
int n=s1.compareTo(s2);
```

0      →      if s1 and s2 are same

+ve    →      if s1 greater than s2

-ve    →      if s1 less than s2

11. String concat(String s)

This method concatenates or joins two strings and returns a third string as result.

```
String s1="Hello";
```

```
String s2="World";
```

```
String s=s1.concat(s2);
```

12. String substring(int i)

Extract sub string from a main string. It returns a new string consisting of all characters starting from the position 'i' until the end of the string.

```
String s="HelloWorld";
```

```
String s1=s.substring(5);
```

13. String substring(int i1,int i2)

Returns a new string consisting of all characters starting from i1 till i2 and i2 will be excluded.

```
String s="HelloWorld";
```

```
String s1=s.substring(3,6);
```

14. int indexOf(String s)

```
String s="This is a book";
```

```
String s1="is";
```

```
int p=s.indexOf(s1);
```

Returns the position of first occurrence of String s1 in string s.

15. `int lastIndexOf(String s)`

```
String s="This is a book";
```

```
String s1="is";
```

```
int p=s.indexOf(s1);
```

Returns the position of last occurrence of String s1 in string s.

16. `String.valueOf(P)`

Creates a string object of parameter P, it is of simple type or object.

We can also create and use arrays that contain strings. We can create string arrays as follows:

```
String itemArray[]=new String[size];
```

```
String itemArray[]=new String[3];
```

Here itemArray is an array of string and can store 3 strings.

String Sorting:

```
import java.lang.*;
```

```
import java.io.*;
```

```
class StringOrdering
```

```
{
```

```
 public static void main(String args[])
```

```
 {
```

```
 String name[]={"Sachin","Dravid","Ganguly","Dhoni","Lakshman","Sehwag"};
```

```
 int size=name.length;
```

```
 String temp=null;
```

```
 for(int i=0;i<size;i++)
```

```
 {
```

```
 for(int j=i+1;j<size;j++)
```

```
 {
```

```
 if(name[j].compareTo(name[i])<0)
```

```
 {
```

```
 temp=name[i];
```

```
 name[i]=name[j];
```

```
 name[j]=temp;
```

```
 }
```

```
 }
```

```
 }
```

```
 for(int i=0;i<size;i++)
```

```
 System.out.println(name[i]);
```

```
 }
```

```
}
```

### **STRING BUFFER CLASS:**

1. StringBuffer is a peer class of String.
2. String creates strings of fixed length, StringBuffer creates strings of flexible length.
3. Strings created using StringBuffer can be modified in terms of both length and content.
4. We can insert characters and sub strings in the middle of a string, or append another string at the end.
5. The following are the various methods of StringBuffer class.

|                     |                                                             |
|---------------------|-------------------------------------------------------------|
| S1.setCharAt(n,'x') | Modifies the n th character to x                            |
| S1.append(s2)       | Appends the string s2 at the end of s1                      |
| S1.insert(n,s2)     | Inserts the string s2 at the position of n of the string s1 |
| S1.setLength(n)     | Sets the length of the string s1 to n.                      |

## **VECTORS:**

1. Variable arguments to methods will be achieved in java through the use of the Vector class that is available in java.util package.
2. Vector is used to create a generic dynamic array that can hold objects of any type and any number and these objects do not have to be homogeneous.
3. We can create vectors as follows.

```
Vector list=new Vector();
Vector list=new Vector(size);
```

4. Vector can be declared without specifying any size or with size but an array must always have its size specified.
5. We cannot store simple data types in a vector, we can store only objects. We need to convert simple type into objects by using wrapper classes.

## **USES:**

1. Vector is convenient to store objects.
2. A vector can be used to store a list of objects that may vary in size.
3. We can add and delete objects from the list as and when required.

The following are the various methods in Vector class.

|                                     |                                                    |
|-------------------------------------|----------------------------------------------------|
| List.addElement(item)               | Adds the specified item at the end of list         |
| List.elementAt(position)            | Gives the name of the object at specified position |
| List.size( )                        | Returns the number of objects present in the list  |
| List.removeElement(item)            | Removes the specified item from the list           |
| List.removeElementAt(position)      | Removes the item from specified position           |
| List.removeAllElements( )           | Removes all elements from the list                 |
| List.copyInto(array-name)           | Copies all items into array                        |
| List.insertElementAt(item,position) | Insert the specified item at specified position    |

```
import java.lang.*;
import java.io.*;
import java.util.*;
class VectorDemo
{
 public static void main(String args[])
 {
 Vector v=new Vector();
 int l=args.length;
 for(int i=0;i<l;i++)
 {
 v.addElement(args[i]);
 }
 }
}
```

```

 }
 v.insertElementAt("COBOL",2);
 int size=v.size();
 System.out.println(v);
 String arr[]=new String[size];
 v.copyInto(arr);
 for(int i=0;i<size;i++)
 System.out.println(arr[i]);
 }
}

```

## WRAPPER CLASSES:

We know that vectors cannot handle primitive data types like int, float, long, char and double. Wrapper classes are used to convert primitive types into object type and these are available in java.util package.

| PRIMITIVE TYPE | WRAPPER CLASS |
|----------------|---------------|
| boolean        | Boolean       |
| int            | Integer       |
| char           | Character     |
| float          | Float         |
| double         | Double        |
| long           | Long          |

We can convert primitive types into object types by using constructor methods as follows.

|                            |                                     |
|----------------------------|-------------------------------------|
| Primitive int to object    | Integer ival=new Integer(20);       |
| Primitive float to object  | Float fval=new Float(3.234);        |
| Primitive double to object | Double dval=new Double(3.14121734); |
| Primitive long to object   | Long lval=new Long(345322323232);   |

We can convert the object type into primitive type by using typeValue( ) methods.

|                            |                              |
|----------------------------|------------------------------|
| Object to primitive int    | int i=ival.intValue( );      |
| Object to primitive float  | float f=fval.floatValue( );  |
| Object to primitive double | double d=dval.doubleValue( ) |
| Object to primitive long   | long l=lval.longValue( )     |

## INTERFACES: MULTIPLE INHERITANCE

Java doesn't support multiple inheritance through classes that is in java a class cannot have more than one base class. For example

class A extends B extends C

```

{

}

```

This is illegal in java.

- A large number of real time applications require the use of multiple inheritance. So java provides an alternate approach known as "Interface" to support the concept of multiple inheritance.
- Even though a java class cannot be a subclass of more than one super class it can be implement more than one interface.

## DEFINING AN INTERFACE:

An interface is similar to class. Like class it contains variables and methods without body.

**Syntax:**

```
interface interface-name
{
 Variable declaration
 Method declaration
}
```

Here “interface” is a keyword and interface-name is any valid java identifier.

**VARIABLE DECLARATION AND METHOD DECLARATION:**

➔ Since all the variables are constants in interfaces these are declared as final.

**Syntax:**

```
final data-type variable-name = value;
final int i=35;
```

The variable inside an interface is treated as final even though the keyword final is absent.

➔ Since all the methods are abstract those are ended with semicolon(;).

**Syntax:**

```
Return-type method-name(parameters);
void display();
```

A class that implements interface must provide the code for methods.

All the fields and methods are public by default.

**EXTENDING INTERFACES:**

Like, classes interfaces can also be extended that is an interface can be inherited from another interface.

```
Ex: interface inter1
 {
 int x=35;
 int y=20;
 }
 interface inter2 extends inter1
 {
 int z=40;
 void display();
 }
```

Even though the interfaces are allowed to extend, the sub interfaces cannot define the methods of a super interface.

It is the responsibility of the class that implements the interface to define all methods.

**IMPLEMENTING INTERFACES:**

1. We can get the properties of super class by using **extends** keyword. In the same way we can get the properties of an interface by using **implements** keyword.
2. A class extended from another class and implemented from an interface.

```
class class-name implements interface-name
```

```
{

}
```

3. If a class that implements an interface does not provide definitions for all methods in the interface then it will become an abstract class and such class cannot be instantiated.

```
class class-name extends superclass-name implements interface-name-1,interface-name-2
{

}
```

### DIFFERENCES BETWEEN CLASS AND INTERFACE

| CLASS                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | INTERFACE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ol style="list-style-type: none"> <li>1. The class keyword is used to create a class.</li> <li>2. A class can contain variables and methods.</li> <li>3. The members of a class by default are friendly modifier.</li> <li>4. A class contains concrete methods.</li> <li>5. A class cannot be extended from more than one super class</li> </ol> <pre>Class class-name {     Variable declaration     Method declaration }</pre> <ol style="list-style-type: none"> <li>6. A class can be extended but never be implemented.</li> </ol> | <ol style="list-style-type: none"> <li>1. An interface is created by using interface keyword.</li> <li>2. An interface can contain only final variables and abstract methods.</li> <li>3. The methods of an interface by default are public modifier.</li> <li>4. An interface contains method declarations without definition.</li> <li>5. An interface can be extended from more than one super interface.</li> </ol> <pre>Interface interface-name {     Final variables;     Abstract methods; }</pre> <ol style="list-style-type: none"> <li>6. An interface can be extended and also can be implemented.</li> </ol> |

```
import java.lang.*;
import java.io.*;
class Student
{
 int rno;
 void getnum(int n)
 {
 rno=n;
 }
 void putnum()
 {
 System.out.println("ROLL NUMBER IS:"+rno);
 }
}
class Test extends Student
{
```

```

float mark1,mark2;
void getmark(int m1,int m2)
{
 mark1=m1;
 mark2=m2;
}
void putmark()
{
 System.out.println("MARKS OBTAINED");
 System.out.println("MARK1 =" +mark1);
 System.out.println("MARK2 =" +mark2);
}
}
interface Sport
{
 final static int sportwt=10;
 void putwt();
}
class Result extends Test implements Sport
{
 float total;
 public void putwt()
 {
 System.out.println("SPORT WT =" +sportwt);
 }
 void display()
 {
 total=mark1+mark2+sportwt;
 putnum();
 putmark();
 putwt();
 System.out.println("TOTAL IS:" +total);
 }
}
class MultipleDemo
{
 public static void main(String args[])
 {
 Result r=new Result();
 r.getnum(40);
 r.getmark(35,35);
 r.display();
 }
}

```

## UNIT-IV

### MULTITHREADING

→ Multithreading is a conceptual programming concept where a program(process) is divided into two or more sub programs(process), which can be implemented at the same time in parallel.

→ A Multi-threaded program contains two or more parts that can be run concurrently(parallel). Each part of such program is called a “THREAD”.

→ A process consists of a memory space allocated by the operating system that can contain one or more threads. A thread cannot exist on its own.

→ Multithreading is conceptually similar to multi-tasking. But there are few differences which distinguish threading from tasking,

#### Multi Tasking

1. It is a feature that allows our computer to run two or more programs parallel.
2. For example, we can listen to music and at the same time type a word document.
3. Processor is shared between programs.
4. A program is heavy weight process.
5. Each processor runs on its separately addressed space.
6. Inter-process communication is expensive.
7. Expensive in context switching.

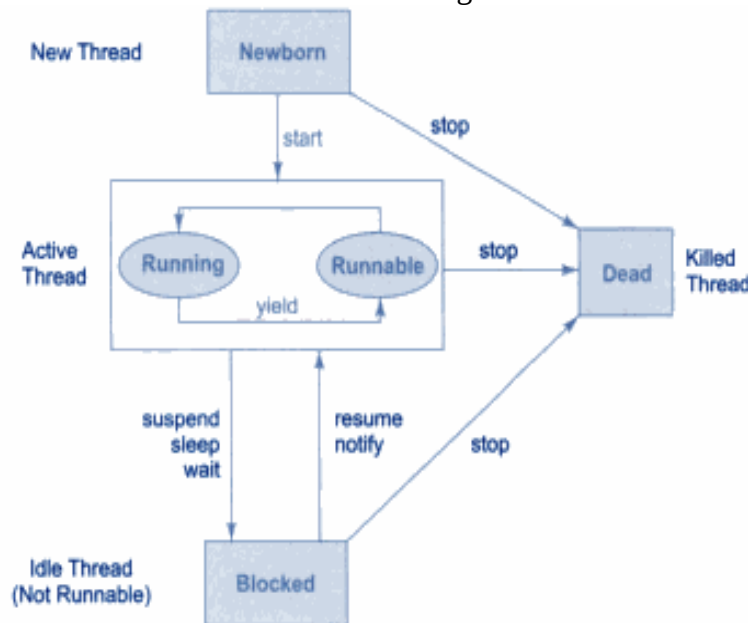
#### Multi Threading

1. Multi threading is a feature that allows single program to perform two or more tasks simultaneously.
2. For example, a text editor can print and at the same time we can edit the text.
3. Processor is shared between parts of a program.
4. A thread is called light weight process.
5. Threads share the addressed space.
6. Inter thread communication is inexpensive.
7. Inexpensive in context switching.

#### **Life Cycle of Thread:**

A thread can be in any one of the states.

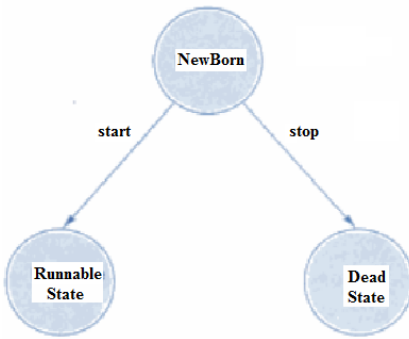
1. New Born State
2. Runnable State
3. Running State
4. Blocked State
5. Dead State



### New Born State:

When a thread object is created a thread is born and said to be in new born state. It won't get into action until we start the thread by using **start( )** method.

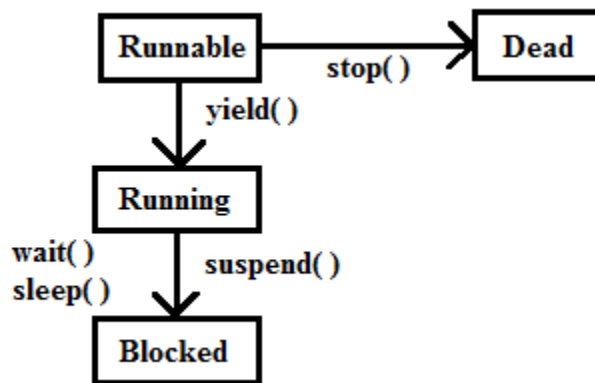
From this new born state, a thread may go to either runnable state or dead state.



### Runnable State:

A thread is said to be in runnable state, if the thread is ready for execution, and waiting for ability to process. If all threads in queue having the same priority then they are given time slots for execution in round robin fashion.

From runnable state, the thread may go to running or blocked state or dead state.



### Running State:

A thread is said to be in running state if the processor has given time to the thread for execution. A thread keeps running until the following conditions occur.

- a. Thread gives up its control on its own and it can happen in the following situations.
  1. A thread gets suspended using `suspend( )` method which can only be revived with `resume( )` method.
  2. A thread is made to sleep for a specified period of time using `sleep(time)` method, where time is given in milliseconds.
  3. A thread is made to wait for some event to occur using `wait( )` method. A thread can be scheduled to run again using `notify( )` method.
- b. If a high priority thread enters for execution then the low priority thread will be preempted(forcefully removed from execution)

### Blocked state:

If a thread is prevented from entering into runnable state and subsequently running state, then a thread is said to be in blocked state.

### Dead State:

A runnable thread enters the dead or terminated state when it completes its task or terminates.

## Creating a Thread:

Java defines two ways to create a thread. Those are

1. Extend the Thread class
2. Implements the runnable interface

Creating a Thread by extending a thread class:

The first way to create a thread is to create a new class that extends thread and then to create an instance of that class.

The extending class must override the run( ) method, which is the entry point for the new Thread. It must also called start( ) method to begin execution of the new Thread.

### Thread Class Methods:

Public void start( ) - It invokes the run( ) method to start the thread.

Public void run( ) - If the thread object was instantiated using a start( ) method then this method will be executed.

Public final void setPriority( ) - Used to set the priority of the thread, the possible values between 10 and 1.

Public int getPriority( ) - Used to get the priority of the thread.

```
import java.io.*;
```

```
import java.lang.*;
```

```
import java.util.*;
```

```
class Even extends Thread
```

```
{
 int m;
 Even(int x)
 {
 m=x;
 }
 public void run()
 {
 for(int i=2;i<=m;i+=2)
 {
 System.out.println("Even Number:"+i);
 }
 System.out.println("Even Numbers Printed");
 }
}
```

```
class Odd extends Thread
```

```
{
 int m;
 Odd(int x)
 {
 m=x;
 }
 public void run()
 {
 for(int i=1;i<=m;i+=2)
 {
 System.out.println("Odd Number:"+i);
 }
 System.out.println("Odd Numbers Printed");
 }
}
```

```

 }
}
class ThreadDemo
{
 public static void main(String args[]) throws IOException
 {
 DataInputStream d=new DataInputStream(System.in);
 System.out.println("Enter a Number:");
 int n=Integer.parseInt(d.readLine());
 Even e=new Even(n);
 Odd o=new Odd(n);
 e.start();
 o.start();
 }
}

```

### Use of suspend( ) method and resume( ) method:

A suspended thread can be revived by using the resume( ) method. This approach is useful when we want to suspend a thread for some time but not kill the thread.

```

class ThreadDemo
{
 public static void main(String args[]) throws IOException
 {
 DataInputStream d=new DataInputStream(System.in);
 System.out.println("Enter a Number:");
 int n=Integer.parseInt(d.readLine());
 Even e=new Even(n);
 Odd o=new Odd(n);
 e.start();
 o.start();
 e.suspend();
 e.resume();
 }
}

```

### Thread Priority:

- ➔ Every java thread has a priority that helps the operating system to determine the order in which the threads are scheduled.
- ➔ Thread priorities are given in the range between MIN\_PRIORITY that is 1 and MAX\_PRIORITY 10.
- ➔ By default, every thread is given priority NORM\_PRIORITY.

```

class ThreadDemo
{
 public static void main(String args[])
 {
 Even e=new Even(10);
 Odd o=new Odd(10);
 e.setPriority(5);
 o.setPriority(10);
 e.start();
 o.start();
 }
}

```

### Synchronization:

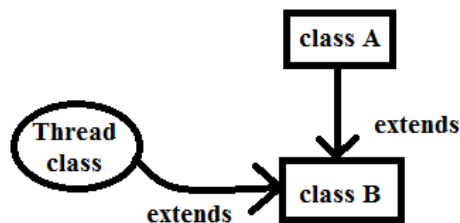
- ➔ When two or more threads need access to a shared resource, they need some mechanism to assure that the resource will be used by only one thread at a time.
- ➔ The process by which the synchronization is achieved is called “Thread Synchronization”.
- ➔ The “synchronized” keyword in java creates a block of code referred to as a critical section.
- ➔ Every java object with critical section of code gets a lock associated with the object.
- ➔ To enter into a critical section, a thread needs to obtain the corresponding object's lock.

Syntax:

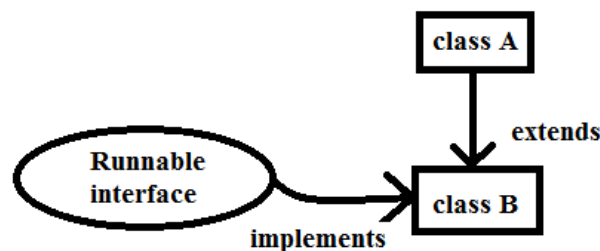
```
synchronized(lock)
{
 // statements
}
```

### Creating Thread Using Runnable Interface:

- ➔ Consider a situation where we need to create a thread class by extending another class. In those situations we face a problem that java doesn't support multiple inheritance.



- ➔ This operation is not possible in java. So there is another technique implemented to achieve this.



- ➔ The following are the various steps to create a thread by implementing runnable interface.
1. Create a class by implementing interface “Runnable”.
  2. Define the code for the abstract method public void run( ).
  3. Create an object for the implemented class.
  4. Create an object for Thread class and instantiated it with the object of implemented class.
  5. Invoke the start( ) method by using Thread object.

```
import java.lang.*;
class Even implements Runnable
{
 int n;
 Even(int a)
 {
 n=a;
 }
 public void run()
 {
 for(int i=2;i<=n;i=i+2)

```

```

 System.out.println(i);
 }
}
class Odd implements Runnable
{
 int n;
 Odd(int a)
 {
 n=a;
 }
 public void run()
 {
 for(int i=1;i<=n;i=i+2)
 System.out.println(i);
 }
}
class EvenOdd
{
 public static void main(String... aargs)
 {
 Even e=new Even(50);
 Odd o=new Odd(50);
 Thread t1=new Thread(e);
 Thread t2=new Thread(o);
 t1.start();
 t2.start();
 }
}

```

### Using of sleep( ) method:

The sleep( ) method is used to stop the running thread from execution for certain milli seconds. On execution of sleep( ) method, the thread immediately goes to blocked state and after the time elapsed, it again comes back to runnable state.

We need to handle the exception while using the sleep method.

```

import java.lang.*;
class TS extends Thread
{
 public void run()
 {
 for(int i=1;i<=5;i++)
 {
 try
 {
 if(i==2)
 sleep(1000);
 }
 catch(Exception e)
 {
 }
 }
 }
}

```

```

 System.out.println(i);
 }
 System.out.println("Numbers printing completed");
}
}
class TSDemo
{
 public static void main(String args[])
 {
 TS obj=new TS();
 obj.start();
 }
}

```

### THREAD EXCEPTIONS:

- ➔ Whenever we call a thread method that is likely to throw an exception, we have to supply an appropriate exception handler to catch it.
- ➔ Whenever we attempt to invoke a method that thread cannot handle it throw an exception.
- ➔ For example, a sleeping method cannot deal with the resume( ) method because a sleeping thread cannot receive any instructions.
- ➔ The catch statement may take one of the following forms:
  1. ThreadDeath → Killed thread
  2. InterruptedException → Cannot handle it in the current state
  3. IllegalArgumentException → Illegal method argument
  4. Exception → Any other Exception

### EXCEPTION HANDLING

Exception: An Exception is a problem that arises during the execution of the program (Run Time Error). An Exception can occur for many reasons including the following.

1. The user had entered invalid data.
2. A file needed to be opened cannot be found
3. A network connection has been lost in the middle of the communication
4. The JVM has run out of memory.

Exception handling is a task of maintaining normal flow of the program. For this, we should try to catch the exception object thrown by the error condition and then display appropriate message for taking correct action.

### Types of Errors:

In general, there are 2 types of errors in a programming language. Those are

1. Compile Time Errors
2. Run Time Errors

**Compile Time Errors:** The following are the most commonly recognized compile time errors.

1. Misplaced else
2. Missing semicolon ;

### 3. Undefined Symbol

All compile time errors occur because of syntactical mistakes of the programmer. Those can be recognized at the design stage and corrected.

**Run Time Errors:** These are happened due to exceptional cases such as

1. Divided by 0
2. Array Index Out of Bound
3. Invalid number format etc.

These errors will be identified at the run time of the program. So, we must use a mechanism to handle these types of errors called “**Exception Handling**”.

Built In Java Exceptions:

- |                                   |                                                 |
|-----------------------------------|-------------------------------------------------|
| 1. ArithmeticException            | - Arithmetic Exception such as divided by zero  |
| 2. ArrayIndexOutOfBoundsException | - Insufficient array size                       |
| 3. ArrayStoreException            | - incompatible assignment of an array element   |
| 4. IllegalArgumentException       | - Incompatible argument used to invoke a method |
| 5. NumberFormatException          | - Invalid Number Format                         |
| 6. NullPointerException           | - Invalid use of Null pointer                   |
| 7. ClassNotFoundException         | - If a class is not found                       |
| 8. NoSuchMethodException          | - A requested method does not exist             |

### Handling Exceptions in Java:

The following keywords are used to handle the exceptions in Java.

1. Try
2. Catch
3. Throw
4. Throws
5. Finally

Try – Catch Block:

1. A method catches an exception with the combination of the try and catch keywords.
2. A try block is placed around the code that might generate an exception.
3. The code within a try-catch block is referred to as “protected code”. The syntax for using try-catch is as follows.

```
try
{
 Protected Code
}
catch(Exception-Name Object-Name)
{
```

### Executable Code

```
}
```

→ A catch statement involves declaring the type of exception we are trying to catch.

→ Any statements placed between try and catch blocks lead to compile time error.

class Demo

```
{
```

```
 public static void main(String... args) throws IOException
```

```
 {
```

```
 DataInputStream d=new DataInputStream(System.in);
```

```
 int a,b,c;
```

```
 System.out.print("Enter a Number:");
```

```
 a=Integer.parseInt(d.readLine());
```

```
 System.out.print("Enter b number:");
```

```
 b=Integer.parseInt(d.readLine());
```

```
 try
```

```
 {
```

```
 c=a/b;
```

```
 }
```

```
 catch(ArithmeticException ae)
```

```
 {
```

```
 System.out.println("Divided by Zero error");
```

```
 }
```

```
 System.out.println("No Error");
```

```
 }
```

```
}
```

### **MULTIPLE CATCH BLOCKS:**

A Try block can be followed by multiple catch blocks. When we use multiple catch statements it is important that exception sub classes come before any of the super class.

#### **Syntax:**

```
try
```

```
{
```

```
 //Protected Code
```

```
}
```

```
catch(Exception-1 Object_name)
```

```
{
```

```
 //statements
```

```
}
```

```
catch(Exception-2 Object_name)
```

```
{
```

```
 //statements
```

```
}
```

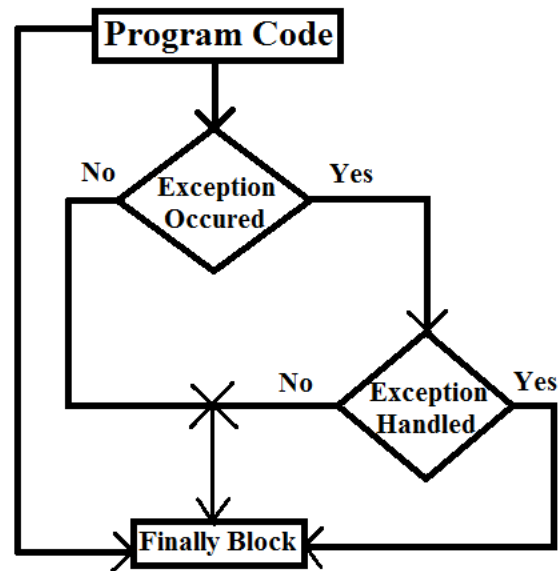
```

import java.io.*;
import java.lang.*;
class Demo
{
 public static void main(String... args)
 {
 int marks[]=new int[3];
 int tot=0,avg;
 try
 {
 DataInputStream d=new DataInputStream(System.in);
 System.out.print("Enter Number of Subjects:");
 int n=Integer.parseInt(d.readLine());
 System.out.print("Enter Marks of "+n+" Subjects");
 for(int i=0;i<n;i++)
 {
 marks[i]=Integer.parseInt(d.readLine());
 tot=tot+marks[i];
 }
 System.out.print("Enter denominator to calculate Average:");
 int dt=Integer.parseInt(d.readLine());
 avg=tot/dt;
 System.out.println("Average :"+avg);
 }
 catch(NumberFormatException e1)
 {
 System.out.println("Invalid Number Format");
 }
 catch(ArithmeticException e2)
 {
 System.out.println("Divided by Zero error");
 }
 catch(ArrayIndexOutOfBoundsException e3)
 {
 System.out.println("Array out of Range");
 }
 catch(ArrayStoreException e4)
 {
 System.out.println("Invalid Array Data");
 }
 catch(IOException e5)
 {
 System.out.println("IO Exception");
 }
 finally
 {
 System.out.println("All Most Done");
 }
 }
}

```

## **FINALLY**

- The finally keyword is used to create a block of code that follows a try-block.
- A final block of code always executes whether or not an exception has occurred.
- Finally block allows us to execute some statements even though we don't know what happens in the protected code.



## **THROWS Keyword:**

- The throws keyword is used to throw the exceptions to the operating system.
- Throws keyword is placed immediately after the name of the method.
- The throws keyword doesn't follow any try-catch statements.

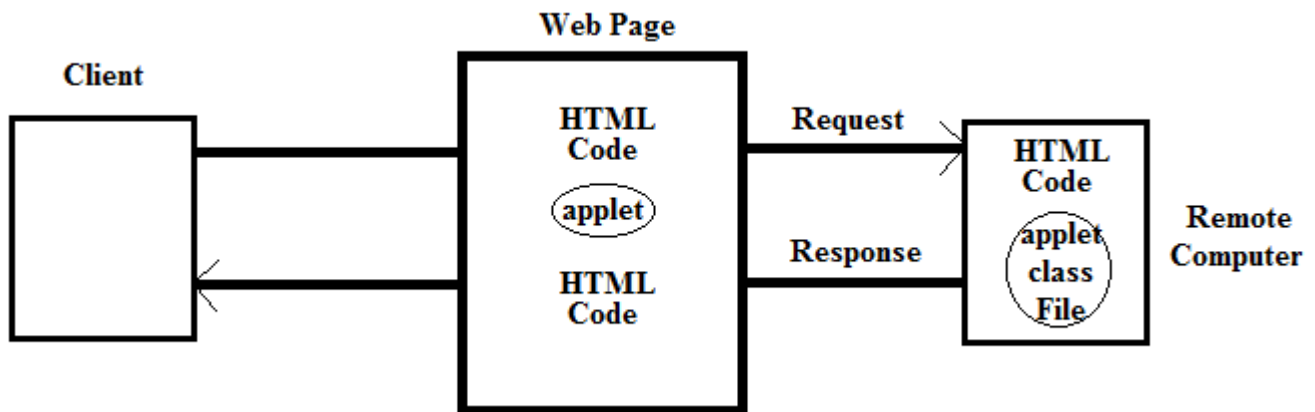
## UNIT-V

### APPLETS

An applet is a program that runs in a web browser. An applet can be a java application. There are certain differences between applet and standard alone java application. These are described below.

- ❖ An applet is a java class that extends Applet class from java applet package.
- ❖ The main( ) method is not involved on an applet. So, an applet class will not be defined main( ) method.
- ❖ Applets are designed to embedded within a HTML page.
- ❖ When a user uses an HTML page that contains an APPLET, the code for that applet is downloaded to the users machine from the server.
- ❖ A JVM is required to view an applet.
- ❖ Applets have strict security rules that are enforced by the web browser.

The following diagram represents the concept of Applet mechanism.



#### Applet Tag: <applet>

The tag is used to attach a java class file to the HTML document. This tag takes several attributes to manipulate the behavior of applet on the web page.

Syntax:

```
<applet code="class File" [codebase = URL] width="value" height="value" />
```

1. The code attribute takes the java class file name as its value. This class file should be available in the server.
2. The codebase attribute is used to specify the complete URL of class file.
3. The height and width attributes are used to specify the screen area occupied by the applet.

#### Creating Applets:

The applets are created by extending Applet class from java applet package. The following steps are followed for designing applets for web application.

1. We need to import applet class from java.applet package.
2. Applets are interactive applications. So we need to import graphical API by java.awt package.
3. Create a class by extending applet class.
4. Override the method

```
public void paint(Graphics g)
```

by placing all the applet code inside that method.

5. Save and compile the code to create class file.
6. Embed the class file into a web page by using <applet> tag

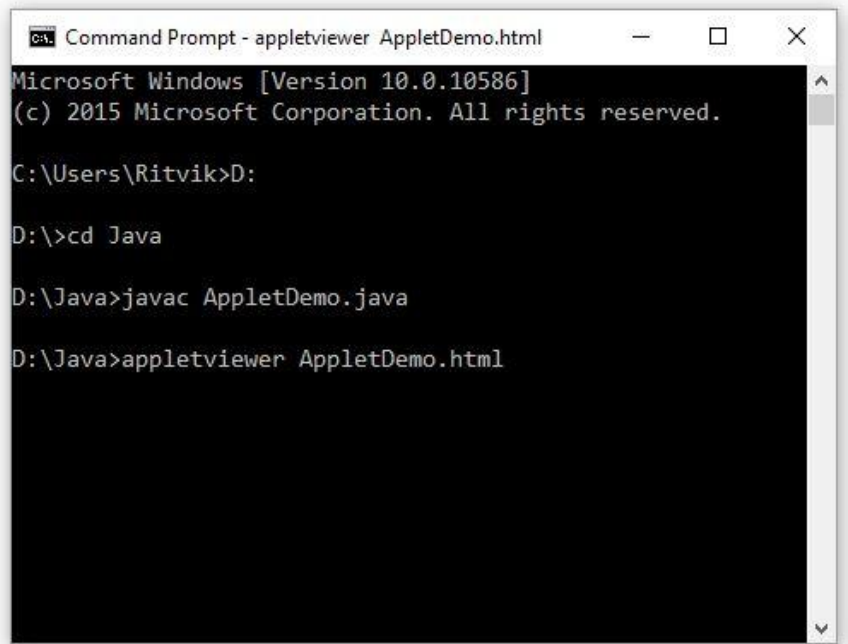
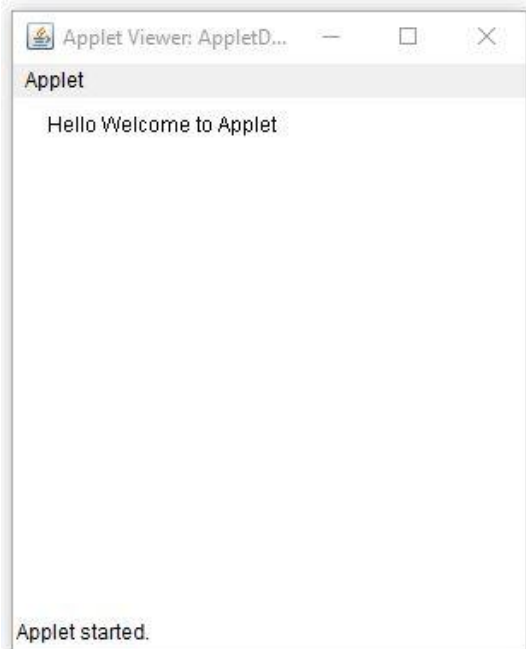
## Program to demonstrate applets:

```
import java.applet.*;
import java.awt.*;
public class AppletDemo extends Applet
{
 public void paint(Graphics g)
 {
 g.drawString("Hello Welcome to Applet",20,20);
 }
}
```

Save this file as AppletDemo.java and compile it.

```
<html>
<applet code="AppletDemo.class" height=300 width=300>
</applet>
</html>
```

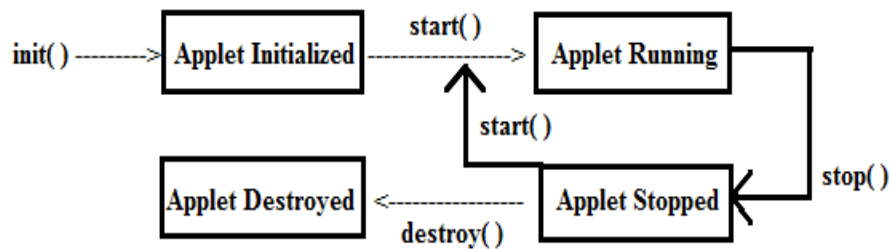
Save this file as AppletDemo.html and run as appletviewer AppletDemo.html



## APPLET LIFE CYCLE:

There are different methods that are used in applet life cycle. An applet is an embedded java class file that runs on a browser by the request of a HTML document.

In general, applet class files are stored in remote computer and downloaded and executed on the client computer. So, the applet passes through number of stages on its journey. The following is the complete life cycle of an Applet class.



➔ **init( ) method:-** This method is intended for whatever initialization has to be done in applet. This method is used to set some default values.

**Syntax:**

```

public void init()
{
 /* Code */
}

```

➔ **start( ) method:-** This method is automatically called after the browser calls the init( ) method. It is also called whenever the user returns to the page containing the applets after having gone off to the other page.

**Syntax:**

```

public void start()
{
 // Code
}

```

➔ **stop( ) method:-** This method is automatically called when the user moves off the page on which the applet is running. Therefore, it can be called repeatedly in the same applet.

**Syntax:**

```

public void stop()
{
 // Code
}

```

➔ **destroy( ) method:-** This method is only called when the browser shuts down. Applets are meant to live on a HTML page. We should not normally leave resources behind after a user leaves the page that contains applet.

**Syntax:**

```

public void destroy()
{
 // Code
}

```

➔ **paint( ) method:-** This method is invoked immediately after the start( ) method and also invoked anytime the applet needs to repaint( ) itself in the browser. The paint( ) method is actually inherited from the java awt package.

**Syntax:**

```

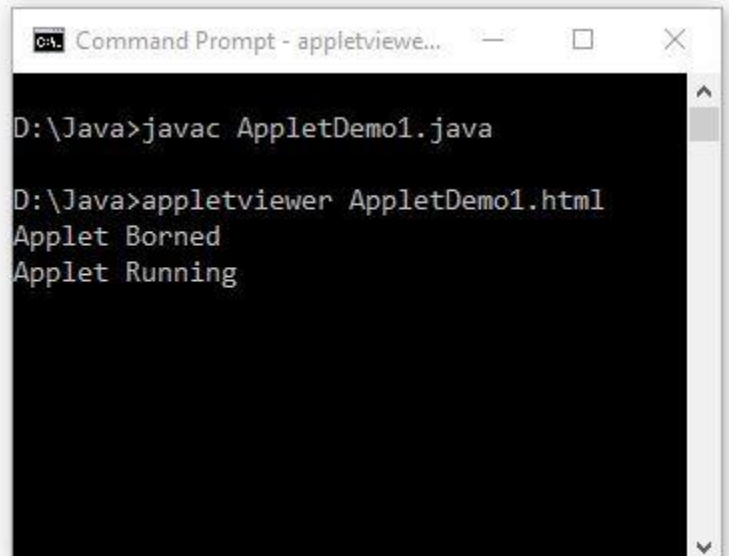
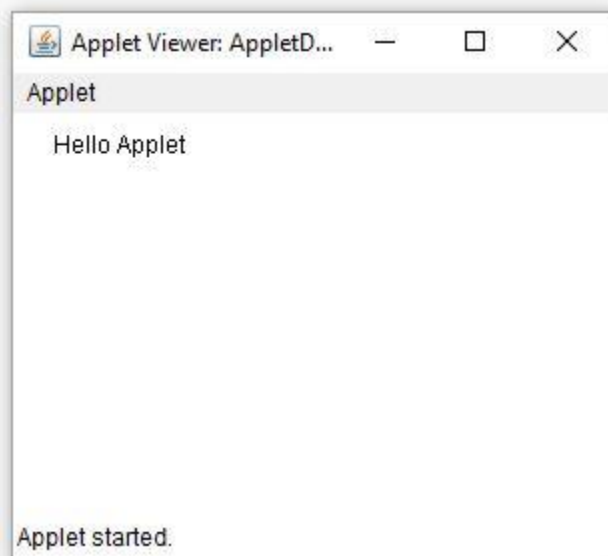
public void paint(Graphics g)
{
 // Code
}

```

```

import java.awt.*;
import java.applet.*;
public class AppletDemo1 extends Applet
{
 public void init()
 {
 System.out.println("Applet Borneed");
 }
 public void start()
 {
 System.out.println("Applet Running");
 }
 public void paint(Graphics g)
 {
 g.drawString("Hello Applet",20,20);
 }
 public void stop()
 {
 System.out.println("Applet Idle");
 }
 public void destroy()
 {
 System.out.println("Applet Dead");
 }
}
<html>
<applet code="AppletDemo1.class" height=200 width=300>
</applet>
</html>

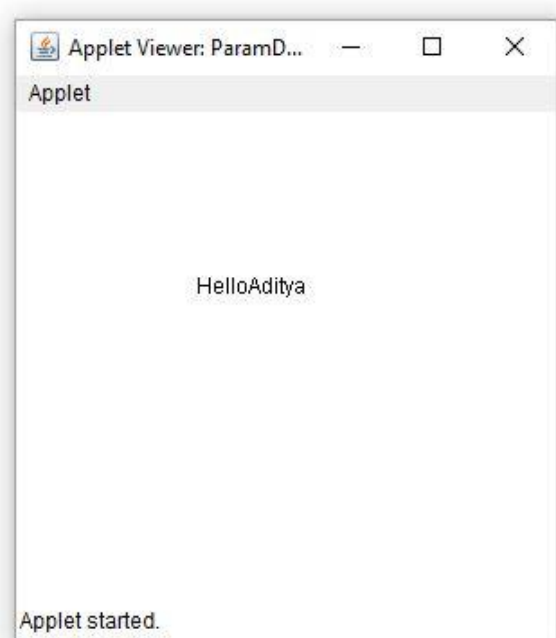
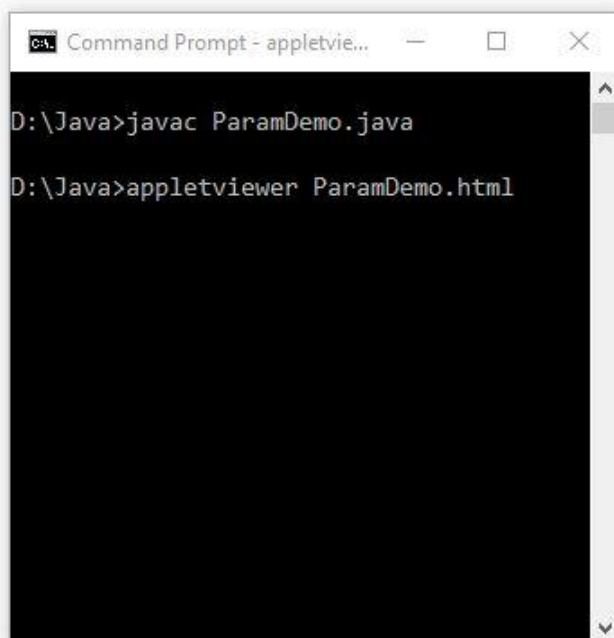
```



## PASSING VALUES TO AN APPLETS:

- ➔ <PARAM> tag is used to pass parameter value from HTML file to applet code. The syntax of PARAM is as follows.
- ➔ <PARAM name="name" value="value"/>
- ➔ The name attribute is used as recognizer of the value passed from HTML file to applet code.
- ➔ The PARAM should be nested inside <APPLET>. The method **getParameter("name")** is used inside the applet code to read the value passed from HTML file using <PARAM>.

```
import java.applet.*;
import java.awt.*;
public class ParamDemo extends Applet
{
 public void paint(Graphics g)
 {
 String s=getParameter("a");
 s="Hello"+s;
 g.drawString(s,100,100);
 }
}
<html>
<head>
<title>PARAMETER PASSING</title>
</head>
<body>
<h1 align=center>Demo of Passing Parameter</h1>
<applet code="ParamDemo.class" height=300 width=300>
<param name="a" value="Aditya"/>
</applet>
</body>
</html>
```



# PACKAGES

1. A package is similar to class libraries in 'C'. Packages provide a new way of reusability.
2. Package is a collection of related classes and interfaces. By organizing the classes into packages we have the following benefits.
  - a. The classes contained in the packages of the other programs can be easily reused.
  - b. In packages, classes can be unique compared to classes in other packages, i.e., two classes in two different packages can have the same name.
  - c. Packages provide a way to hide classes, thus preventing other programs or packages to access.
  - d. Packages also provide a way to separate design from coding.
3. There are two different kinds of packages in Java.
  - a. Java API packages (System packages)
  - b. User Defined Packages

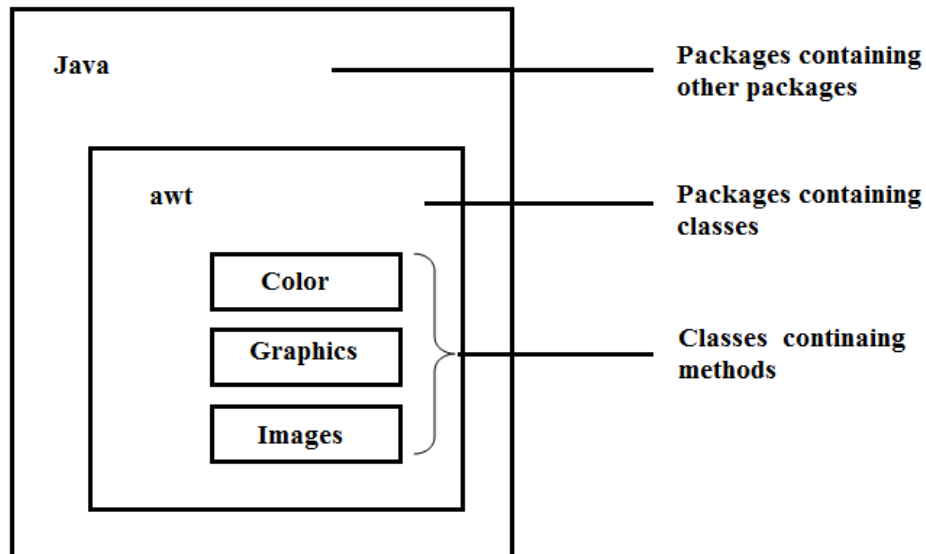
## JAVA API PACKAGES:

### Java API:

- ↪ Language package (collection of classes and methods used to write programs)
- ↪ I/O package (collection of classes and methods used to perform I/O operations)
- ↪ Utility package (collection of classes and methods used to perform Date and time functions)
- ↪ Network package (collection of classes and methods used to connect one computer to another)
- ↪ Abstract window Toolkit package (collection of classes and methods used to develop GUI)
- ↪ Applet package (collection of classes and methods that are used for applet applications)

## USING SYSTEM PACKAGES:

Packages are organized in a hierarchical structure. For example



In the above example, “Java” is a package which contains different sub packages. One among them is “awt” which contains one or more classes.

To use a specific class, we need to address its “fully qualified name” from top package to class. For example, to use the class “Images” in the above example we need to write **java.awt.Image;**

Another way to import system packages is import statement **import java.awt.\*;**

Which imports all the classes of java.awt to our program. So, the methods of such classes can be accessed by their partial names.

## CREATING PACKAGES:

Creating user defined packages involves the following steps:

1. Declare the package at the beginning of the source file using the form `package package-name;`  
Here, package is the keyword and package-name is any valid java identifier.
2. Define the class that is to be put in the package as public.  
A java package file can have more than one class definition. In such cases, one of the class may be declared public and save with that class name.
3. Create a sub-directory under the directory where the main source files are stored. The name of the sub-directory should be exactly same as package name as we declared in package file.
4. Store the source file in the created sub-directory.
5. Compile the file. This will create dot class file in the created sub-directory.

## ACCESSING PACKAGES:

1. Java system packages can be accessed either by fully qualified class name or using a shortcut approach.
2. We use import statement to a particular package to access classes in that package.  
**`import package1[ . package2][ . package3] . classname;`**
3. Here package1 is the name of the top level package, package2 is the name of the package that is inside the package1, and so on.
4. We can have any number of packages in a hierarchy, at last you need to specify the class name.
5. The point to remember is that the statement must be ends with semi colon (;).  
**`import packageName.*;`**
6. Here \* indicates that the compiler should search the entire package for a class.

## USING A PACKAGE:

```
package Mypack;
import java.lang.*;
public class PackageDemo
{
 void display()
 {
 System.out.println("Welcome to package");
 }
}
```

1. Save the above file as “PackageDemo.java”.
2. Create a sub-directory with name “Mypack”.
3. Copy the “PackageDemo.java” into “Mypack” directory.
4. Compile the “PackageDemo.java” to create “PackageDemo.class”.

```
import Mypack.PackageDemo;
import java.lang.*;
class MainProgram
{
 public static void main(String args[])
 {
 PackageDemo obj=new PackageDemo();
 p.display();
 }
}
```

```
}
```

## ADDING A CLASS TO A PACKAGE:

We can add a class to an existing package as follows.

```
package P1
public class classA
{
 Body
}
```

The package P1 contains one public class classA. If we want to add new classB to this package, this can be done as follows.

1. Define the class and make it as public.
2. Place package statement package P1; before the class definition as follows.

```
package P1;
public class classB
{
 Body
}
```

3. Store this file as classB.java under P1 sub-directory.
4. Compile classB.java file to get classB.class file.

➔ A java source file can have only one class declared as public, we cannot place two or more classes as public in a single file.

## HIDING CLASSES:

1. Whenever we import a package using asterisk(\*), all public classes in that package are imported.
2. If we want some classes not to import then you can make them as not public.

```
package P1
public class A
{
 Body - A
}
class B
{
 Body - B
}
```

3. The not public classes can be accessed by the other classes in the same package only.
4. Java source file can have only one public class and any number of non-public classes.
5. If a class import the above package P1 and try to create an object for the class B then compiler may generate an error message, because class B is not public and therefore it is not imported.

## **STATIC IMPORT:**

1. By importing a class, we can have the rights to access all the members and methods of that class. Along with instance members, java classes contain static variables and static methods.
2. We can use the import statement to import classes from packages and use them without qualifying the package.
3. Similarly, we can use the static import statement to import static members from the classes and use them without qualifying the class name.

**import static package-name . sub-package-name . class-name . static-member-name;**  
**(or)**

**import static package-name . sub-package-name . class-name.\*;**