JAVA BASICS & OOPS CONCEPTS

What is Java?

Java is a high-level, general-purpose, object-oriented, and secure programming language developed by James Gosling at Sun Microsystems, Inc. in 1991. It is formally known as OAK. In 1995, Sun Microsystem changed the name to Java. In 2009, Sun Microsystem takeover by Oracle Corporation.

Editions of Java

Each edition of Java has different capabilities. There are three editions of Java:

- **Java Standard Editions (JSE) :** It is used to create programs for a desktop computer.
- **Java Enterprise Edition (JEE) :** It is used to create large programs that run on the server and manages heavy traffic and complex transactions.
- **Java Micro Edition (JME) :** It is used to develop applications for small devices such as set- top boxes, phone, and appliances.

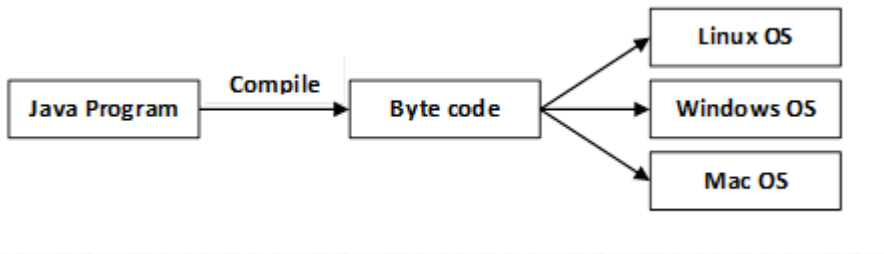
Types of Java Applications

There are four types of Java applications that can be created using Java programming:

- **Standalone Applications:** Java standalone applications uses GUI components such as AWT, Swing, and JavaFX. These components contain buttons, list, menu, scroll panel, etc. It is also known as desktop alienations.
- **Enterprise Applications:** An application which is distributed in nature is called enterprise applications.
- **Web Applications:** An applications that run on the server is called web applications. We use JSP, Servlet, Spring, and Hibernate technologies for creating web applications.
- **Mobile Applications:** Java ME is a cross-platform to develop mobile applications which run across smartphones. Java is a platform for App Development in Android.

Java Platform

Java Platform is a collection of programs. It helps to develop and run a program written in the Java programming language. Java Platform includes an execution engine, a compiler and set of libraries. Java is a platform-independent language.



Features of Java

•**Simple:** Java is a simple language because its syntax is simple, clean, and easy to understand. Complex and ambiguous concepts of C++ are either eliminated or re-implemented in Java. For example, pointer and operator overloading are not used in Java.

•**Object-Oriented:** In Java, everything is in the form of the object. It means it has some data and behavior. A program must have at least one class and object.

•**Robust:** Java makes an effort to check error at run time and compile time. It uses a strong memory management system called garbage collector. Exception handling and garbage collection features make it strong.

•**Secure:** Java is a secure programming language because it has no explicit pointer and programs runs in the virtual machine. Java contains a security manager that defines the access of Java classes.

•**Platform-Independent:** Java provides a guarantee that code writes once and run anywhere. This byte code is platform-independent and can be run on any machine.

•**Portable:** Java Byte code can be carried to any platform. No implementation-dependent features. Everything related to storage is predefined, for example, the size of primitive data types.

•**High Performance:** Java is an interpreted language. Java enables high performance with the use of the Just-In-Time compiler.

•**Distributed:** Java also has networking facilities. It is designed for the distributed environment of the internet because it supports TCP/IP protocol. It can run over the internet. EJB and RMI are used to create a distributed system.

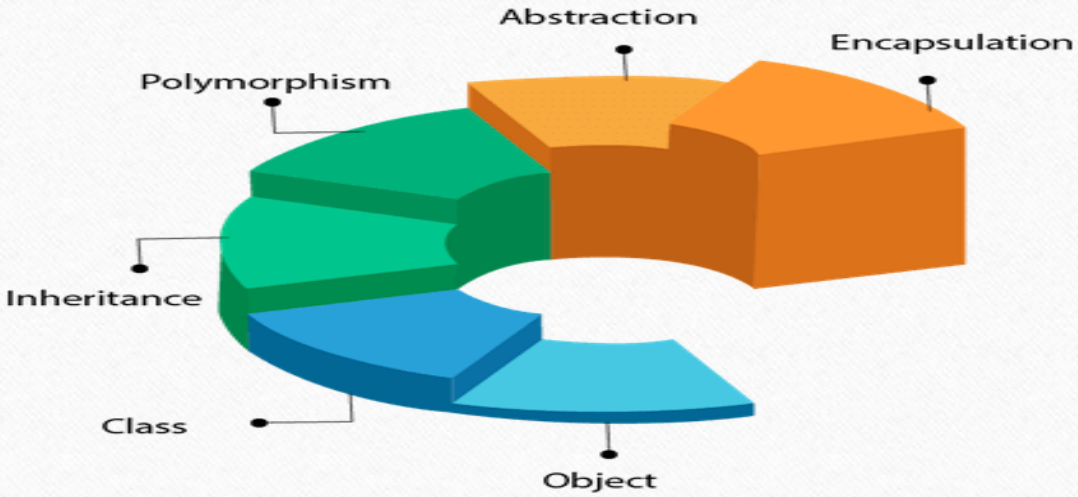
•**Multi-threaded:** Java also supports multi-threading. It means to handle more than one job a time.

OOPS (Object Oriented Programming System)

Object-oriented programming is a way of solving a complex problem by breaking them into a small sub-problem. An object is a real-world entity. It is easier to develop a program by using an object. In OOPs, we create programs using class and object in a structured manner.

What is OOPs in java?

Oops in java is to improve code readability and reusability by defining a Java program efficiently. The main principles of object-oriented programming are abstraction, encapsulation, inheritance, and polymorphism. These concepts aim to implement real-world entities in programs.



Objects:-

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Example1: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Example2:

```
Public class Mybook {
Int x=10;
Public static void main (String args []){
Mybook Myobj= new Mybook ();
System.out.println(MyObj.x);
}
}
```

In the above example, a new object is created, and it returns the value of x which may be the number of books.

Mybook Myobj= new Mybook ();

This is the statement used for creating objects.

System.out.println(Myobj.x);

This statement is used to return the value of x of an object.

Class:-

Collection of objects is called class. A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

A class declaration consists of:

1.Modifiers: These can be public or default access.
2.Class name: Initial letter.
3.Superclass: A class can only extend (subclass) one parent.
4.Interfaces: A class can implement more than one interface.
5.Body: Body surrounded by braces, { }.

Example:-Create a class named "Main" with a variable x:

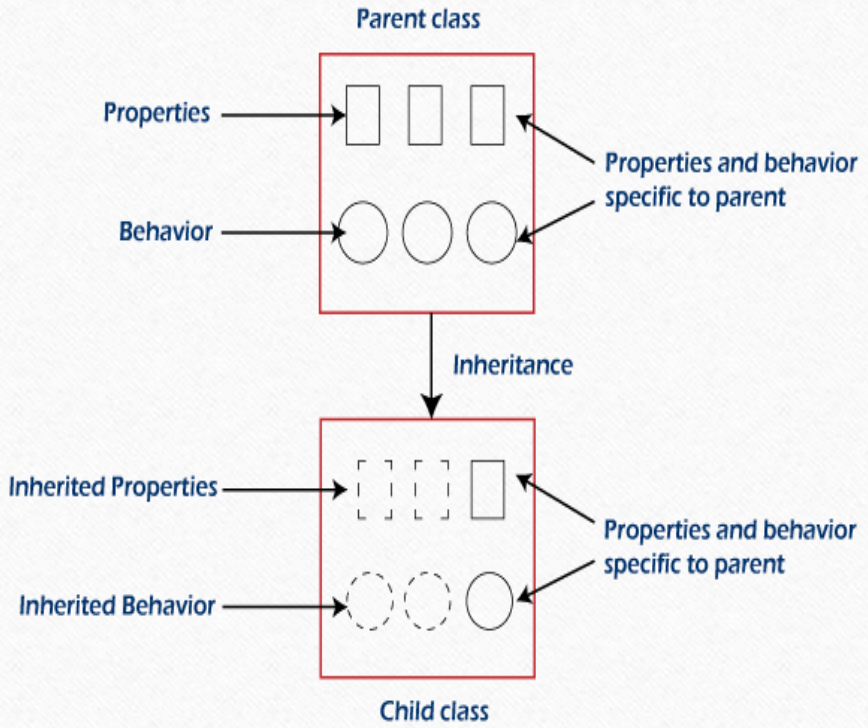
```
public class Main {
  int x = 5;
}
```

Inheritance:-

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Example

```
//base class or parent class or super class
class A{
//parent class methods
void method1(){}
void method2(){}
}
//derived class or child class or base class
class B extends A{ //Inherits parent class methods
//child class methods
void method3(){}
void method4(){}
}
```



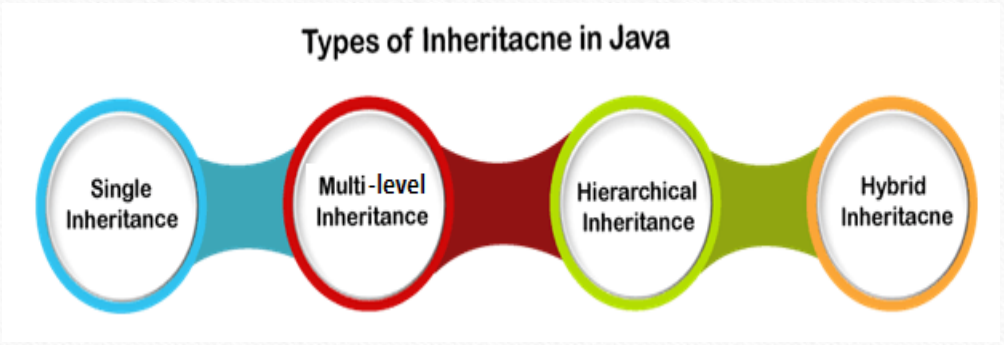
Types of Inheritance:-

Java supports the following four types of inheritance:

- Single Inheritance
- Multi-level Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Java doesn't support the following type of inheritance:

- Multiple Inheritance



Single Inheritance:-

- In single inheritance, a sub-class is derived from only one super class. It inherits the properties and behavior of a single-parent class. Sometimes it is also known as simple inheritance.
- Class A is the base or parental class and class b is the derived class.

Syntax:-

```
Class a {
}
Class b extends class a {
}
```

Multi-level Inheritance:-

- This one class is derived from another class which is also derived from another class i.e., this class has more than one parental class, hence it is called multilevel inheritance.

Syntax:-

```
Class a {
}
Class b extends class a {
}
Class c extends class b {
}
```

Hierarchical Inheritance:-

- In this one parental class has two or more derived classes or we can say that two or more child classes have one parental class.
  Syntax:-
  Class a {
  }
  Class b extends class a {
  }
  Class c extends class a {
  }

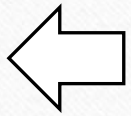
Hybrid Inheritance:-

- Hybrid means consist of more than one. Hybrid inheritance is the combination of two or more types of inheritance.
- In the below figure GrandFather is a super class. The Father class inherits the properties of the GrandFather class. Since Father and GrandFather represents single inheritance. Further, the Father class is inherited by the Son and Daughter class. Thus, the Father becomes the parent class for Son and Daughter. These classes represent the hierarchical inheritance. Combinedly, it denotes the hybrid inheritance.

Multiple Inheritance (not supported):-

Java does not support multiple inheritances due to ambiguity. For example, consider the following Java program.

```
class Wishes
{
void message()
{
System.out.println("Best of Luck!!");
}
}
class Birthday
{
void message()
{
System.out.println("Happy Birthday!!");
}
}
public class Demo extends Wishes, Birthday  //considering a scenario
{
public static void main(String args[])
{
Demo obj=new Demo();
//can't decide which classes' message() method will be invoked
obj.message();
}
}
```



The code gives error because the compiler cannot decide which message() method is to be invoked. Due to this reason, Java does not support multiple inheritances at the class level but can be achieved through an interface.

Polymorphism:-

- If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.
- In Java, we use method overloading and method overriding to achieve polymorphism.

Example:

```
public class Bird {
Public void sound ( ) {
System.out.println ( " birds sounds " );
}
}
public class pigeon extends Bird {
@override
public void sound ( ) {
System.out.println( " cooing " ) ;
}
}
public class sparrow extends Bird ( ) {
@override
Public void sound ( ){
System.out.println( " chip " ) ;
}
}
```

In the example, we can see common action sound () but there are different ways to do the same action. This is one of the examples which shows polymorphism.

Polymorphism in java can be classified into two types:
1.Static / Compile-Time Polymorphism
2.Dynamic / Runtime Polymorphism

What is Compile-Time Polymorphism in Java?

Compile-Time polymorphism in java is also known as Static Polymorphism. to resolved at compile-time which is achieved through the Method Overloading.

What is Runtime Polymorphism in Java?

Runtime polymorphism in java is also known as Dynamic Binding which is used to call an overridden method that is resolved dynamically at runtime rather than at compile time.

What is method overloading and method overriding in Java?

When two or more methods in the same class have the same method name but different parameters, this is called overloading. In contrast, overriding occurs when two methods have the same name and parameters.

Encapsulation:-

- Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.
- A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Capsule

```
//Encapsulation using private modifier
//Employee class contains private data called employee id and employee name
class Employee {
        private int empid;
        private String ename;
}
```

Abstraction:-

Abstraction is the process of hiding certain details and showing only essential information to the user.
Abstraction can be achieved with either abstract classes or interfaces (which you will learn more about in the next chapter).
The abstract keyword is a non-access modifier, used for classes and methods:
Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

Interface in java:-

An interface in Java is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

It is used to achieve abstraction.

By interface, we can support the functionality of multiple inheritance.

It can be used to achieve loose coupling.

How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax:

```
interface <interface_name>{

    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

Constructors in Java:-

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Rules for creating Java constructor

There are two rules defined for the constructor.

1.Constructor name must be the same as its class name

2.A Constructor must have no explicit return type

3.A Java constructor cannot be abstract, static, final, and synchronized.

Types of Java constructors

There are two types of constructors in Java:

Default constructor (no-arg constructor):-A constructor is called "Default Constructor" when it doesn't have any parameter.The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Parameterized constructor:-A constructor which has a specific number of parameters is called a parameterized constructor.

THANK YOU