

Introduction to Java:

What is Java?

* Java is a popular platform independent and Object Oriented programming language, developed by James Gosling in 1995.

* The main purpose of OOP is to deal with real world entities using Programming Language.

Basic concepts of Object-Oriented Programming Language:

1. Object

2. Class

3. Inheritance

4. Polymorphism

5. Abstraction

6. Encapsulation

1. Object –

Objects are basic run time entities in OOP. An object is an instance of a class which executes the class.

They may represent cars, dogs, humans, a place, a bank account or any item that the program has to handle.

All these objects have a state and a behavior.

Example: If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging the tail, running.

Software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods.

So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods.

once an object is created, it takes up space in memory like other variables.

Object Creation:

Syntax: `class_name object_name = new class_name();`

eg. `Pen p = new Pen();`

2. Class –

A class is a collection of objects and it doesn't occupy any space on memory.

A class can be defined as a template/blueprint from which individual objects are created.

In JAVA, we have two types of classes.

i. pre-defined class: eg. System, String etc.

ii. User-defined class:

Following is a sample of a User-defined class.

```
class Dog
{
    String breed; // describes state
    int age;      // describes state
    String color; // describes state
    void barking() // describes behaviour
    {
        System.out.println("Dog is Barking");
    }
    void hungry()
    {
        System.out.println("Dog is Hungry");
    }
}
```

```

    }

    void sleeping()
    {
        System.out.println("Dog is Sleeping");
    }
}

```

A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.

Difference between object and class

No.	Object	Class
1)	Object is an instance of a class.	Class is a blueprint or template from which objects are created.
2)	Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a group of similar objects .
3)	Object is a physical entity.	Class is a logical entity.
4)	Object is created through new keyword mainly e.g. Student s1=new Student();	Class is declared using class keyword e.g. class Student{ }
5)	Object is created many times as per requirement.	Class is declared once .
6)	Object allocates memory when it is created .	Class doesn't allocated memory when it is created .
7)	There are many ways to create object in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only one way to define class in java using class keyword.

3. Inheritance

Inheritance can be defined as the process where one class acquires the properties (methods and data members) of another class with the help of 'extends' keyword.

The class which inherits the properties of other class is known as **subclass (derived class, child class)** and the class whose properties are inherited is known as **superclass (base class, parent class)**.

Inheritance can take place between two or more than two classes.

Inheritance defines **is-a** relationship between a Super class and its Sub class.

extends is the keyword used to inherit the properties of a class.

Syntax:

```
class Super
{
    .....
    .....
}
class Sub extends Super
{
    .....
    .....
}
```

Example:

```
class Vehicle
{
    .....
}
class Car extends Vehicle
{
    ..... //extends the property of vehicle class.
}
```

Types of Inheritance:

There are various types of inheritance as demonstrated below.

1. Single Inheritance:

Derived class with only one base class.

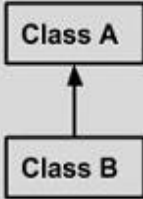
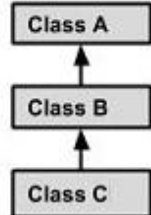
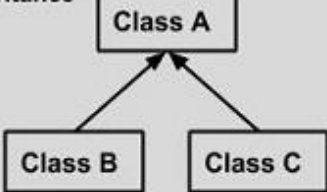
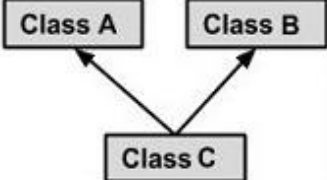
2. Multilevel Inheritance:

Mechanism of deriving a class from another derived class.

3. Hierarchical Inheritance:

Features of one class may be inherited by more than one class.

NOTE : Multiple inheritance is not supported in java.

Single Inheritance  <pre>graph BT; B[Class B] --> A[Class A]</pre>	<pre>public class A { } public class B extends A { }</pre>
Multi Level Inheritance  <pre>graph BT; C[Class C] --> B[Class B]; B --> A[Class A]</pre>	<pre>public class A {} public class B extends A {.....} public class C extends B {.....}</pre>
Hierarchical Inheritance  <pre>graph BT; B[Class B] --> A[Class A]; C[Class C] --> A</pre>	<pre>public class A {} public class B extends A {.....} public class C extends A {.....}</pre>
Multiple Inheritance  <pre>graph BT; C[Class C] --> A[Class A]; C --> B[Class B]</pre>	<pre>public class A {} public class B {.....} public class C extends A,B { } // Java does not support mutiple Inheritance</pre>

Single Inheritance:

Class A

```
{  
    public void methodA()  
    {  
        System.out.println("Base class method");  
    }  
}
```

Class B extends A

```
{  
    public void methodB()  
    {  
        System.out.println("Child class method");  
    }  
    public static void main(String args[])  
    {  
        B obj = new B();  
        obj.methodA(); //calling super class method  
        obj.methodB(); //calling local method  
    }  
}
```

Multilevel Inheritance:

class A

```
{  
  
    int i=100, j=200;  
  
    void xyz()  
    {  
        System.out.println(i+j);  
    }  
  
    void lmn()
```

```

        {
            System.out.println("!!!\tGood Afternoon!!!");
        }
    }

class B extends A
{
    int k=120, l=45;
    void abc()
    {
        System.out.print("Hi  Roll Number:");
        System.out.print(i+j+k);
        lmn();
    }
}

class C extends B
{
}

class MultiLevel
{
    public static void main(String args[])
    {
        C p= new C();
        p.abc();
    }
}

```

```
}
```

Hierarchical Inheritance:

Class A

```
{  
    public void methodA()  
    {  
        System.out.println("method of Class A");  
    }  
}
```

Class B extends A

```
{  
    public void methodB()  
    {  
        System.out.println("method of Class B");  
    }  
}
```

Class C extends A

```
{  
    public void methodC()  
    {  
        System.out.println("method of Class C");  
    }  
}
```

Class D extends A

```
{  
    public void methodD()  
    {  
        System.out.println("method of Class D");  
    }  
}
```

Class MyClass

```
{  
    public void methodB()  
    {  
        System.out.println("method of Class B");  
    }  
    public static void main(String args[])  
    {
```

```
        B obj1 = new B();
```

```
        C obj2 = new C();
```

```
        D obj3 = new D();  
    }
```



```
obj1.methodA();  
obj2.methodA();  
obj3.methodA();  
}  
}
```

Why multiple Inheritance is not supported by Java?

When one class extends more than one classes then this is called **multiple inheritance**.

For **example**: Class C extends class A and B then this type of **inheritance** is known as **multiple inheritance**.

Java does not support multiple inheritance due to **diamond-ambiguity** problem. Consider a case where class C extends class A and Class B and both class A and B have the same method display().

Now **java** compiler cannot decide, which display method it should **inherit**. To prevent such situation, **multiple inheritance** is **not allowed in java**.

4. Polymorphism:

It is one of the OOPS principle where object shows different behavior at different stages of life cycle. i. e. The ability to define a function in multiple forms is called Polymorphism.

Polymorphism is an latin word where poly stands for “many” and morphism stands for “forms”.

In Java Polymorphism is classified into two types:

- i. Compile time Polymorphism
- ii. Run time polymorphism.

A. Compile time polymorphism:

In compile time polymorphism, method declaration is going to get binded to its definition at compilation time based on arguments is known as compile time polymorphism.

As binding takes during compilation time only, so it is also known as **early binding**.

Once binding is done, again rebinding can't be done so it is called as **static binding**.

It is achieved using **method overloading**.

```
public class Overloading {  
  
    public void addition()  
    {  
        int a = 50;  
        int b = 40;  
        System.out.println("Addition of two nos:" + (a + b));  
    }  
    public void addition(int a, int b)  
    {  
        System.out.println("Addition:" + (a + b));  
    }  
    public void addition(int a, int b, int c) {  
        System.out.println("Addition of three nos:" + (a + b + c));  
    }  
    public void addition(int a, int b, int c, int d)  
    {
```

```

        System.out.println("Addition of four nos:" + (a + b + c + d));
    }
}

public class TestOverloading
{
    public static void main(String[] args)
    {
        Overloading o = new Overloading();
        o.addition();
        o.addition(50, 80);
        o.addition(10, 20, 30);
        o.addition(30, 40, 50, 60);

    }
}

```

B. Runtime polymorphism:

In runtime polymorphism, method declaration is going to get binded to its definition at runtime, based on object creation is known as **runtime polymorphism**.

As binding takes place at runtime i.e. after compilation it is called as **late binding**.

Once binding is done, again rebinding can be done, so it is called as **dynamic binding**.

Run time Polymorphism can be achieved using **method overriding**.

```
public class Overriding1
{
    public void m1()
    {
        System.out.println("running method in super class:m1");
    }
}

public class Overriding2 extends Overriding1
{
    public void m1()
    {
        System.out.println("running method in sub class:m1");
    }
}

public class TestOverriding
{
    public static void main(String[] args)
    {
        Overriding2 o2 = new Overriding2();
        o2.m1();
    }
}
```

Difference between method overloading and method overriding:

5. Abstraction:

Abstraction is one of the most important oops principle in java.

Hiding the implementation code and providing only functionality to the end user is called abstraction.

Ex. ATM machine where we can withdraw or transfer money easily but how it happens. We don't know. We don't know internal details or implementation details.

We can achieve abstraction in java by using two way:

- By using abstract class (0 to 100%).
- By using interface (100%).

i. Abstract Class:

A class which contains the abstract keyword in its declaration is known as abstract class.

Abstract class contains both abstract and non-abstract methods.

It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method. Eg.

```
abstract class Bike
```

```
{  
    abstract void run(); //no method body and abstract  
}
```

```
class Honda4 extends Bike
```

```
{  
    void run()  
    {
```

```

        System.out.println("running safely..");
    }

    public static void main(String args[])
    {
        Bike obj = new Honda4();
        obj.run();
    }
}

```

OUTPUT: running safely..

6. Encapsulation-

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.

In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.

```

public class Student
{
    private String name;           //private data member
    public String getName()        //getter method for name
    {
        return name;
    }
    public void setName(String name)
    {
        this.name=name
    }
}

class Test
{
    public static void main(String[] args)
    {
        Student s=new Student(); //creating instance of the encapsulated class
        s.setName("vijay"); //setting value in the name member
        System.out.println(s.getName()); //getting value of the name member
    }
}

```

```
}  
}
```

What is the difference between Polymorphism and Inheritance?

Sr. No.	Inheritance	Polymorphism
1.	Inheritance is one in which a new class is created (derived class) that inherits the features from the already existing class(Base class).	Whereas polymorphism is that which can be defined in multiple forms.
2.	It is basically applied to classes.	Whereas it is basically applied to functions or methods.
3.	Inheritance supports the concept of reusability and reduces code length in object-oriented programming.	Polymorphism allows the object to decide which form of the function to implement at compile-time (overloading) as well as run-time (overriding).
4.	Inheritance can be single, multiple, hierarchical and multilevel inheritance.	Whereas it can be compile-time polymorphism (overload) as well as run-time polymorphism (overriding).