Object and Class Definition: A class is a non-primitive or user-defined data type in Java, while an object is an instance of a class. A class is a basis upon which the entire Java is built because class defines the nature of an object.

Object: An object is an identifiable entity with some characteristics, state and behavior. Understanding the concept of objects is much easier when we consider real-life examples around us because an object is simply a real-world entity. You will find yourself surrounded by the number of objects which have certain characteristics and behaviors.

For example: we can say 'Orange' is an object. Its characteristics are: spherical in shape and color is orange. Its behavior is: juicy and tastes sweet-sour.

Class: A class is a group of objects that share common properties and behavior.

For example: we can consider a car as a class that has characteristics like steering wheels, seats, brakes, etc. And its behavior is mobility. But we can say Honda City having a reg.number 4654 is an 'object' that belongs to the class 'car'.

For Example:

```
public class Person
{
 //class body starts here
  //creating the data members of the class
static String name = "John";
static int age = 25;
  //creating the methods of the class
void eat()
  {
    //methodBody
    System.out.println("he is like to eat ice-cream");
  }
void study()
  {
    //methodBody
    System.out.println("he is like to read history");
  }
void play()
  {
    //methodBody
```

```
    System.out.println("he is like to playing cricket");
  }
public static void main(String args[])
  {
System.out.println("Name of the person: " +name);
System.out.println("Age of the person: " +age);
Person p1=new Person();
p1.eat();
p1.study();
p1.play();
  }
}
```

Using Multiple Classes: Using multiple classes means that we can create an object of a class and use it in another class. Also, we can access this object in multiple classes.

For Example:

```
public class Add {
int a;
int b;
int c;
void add(int x , int y )
{
a = x;
y = b;
c = a + b;
System.out.println("C = " + c);
}
}
```

Program: 2 using subtraction.

```
public class Sub
{
int m;
int n;
int p;
void subt(int r, int s)
{
```

```
    m = r;
    n = s;
    p = m - n;
System.out.println("P =" + p);
    }
    }
```

Program: 3 final show result in this program.

```
public class Test
{
public static void main(String[] args)
    {
System.out.println("hello");
  Add obj = new Add();
    Sub obj2 = new Sub();
obj.add(10,30);
obj2.subt(30,20);
    }
    }
```

Object: As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class −

Declaration: A variable declaration with a variable name with an object type.

Instantiation: The 'new' keyword is used to create the object.

Initialization: The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

For Example:

```
public class obj {
public obj(String name) {
       // This constructor has one parameter, name.
System.out.println("My first Name is :" + name );
    }

public static void main(String [] args) {
       // Following statement would create an object myPuppy
```

```
obj myname = new obj( "rajiv" );
    }
 }
```

Encapsulation: Encapsulation is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction.

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.

To achieve encapsulation in Java –

- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

For Example: Program First in declare the information in program.

```
public class A {
private String name;
private String idNum;
private int age;
public int getAge()
{
return age;
   }
public String getName()
{
return name;
   }
public String getIdNum()
{
return idNum;
   }
public void setAge( int newAge)
{
age = newAge;
   }
public void setName(String newName)
```

```
{
name = newName;
   }
public void setIdNum( String newId)
{
idNum = newId;
   }
}
```

For Example: Second program is show the result display in output.

```
public class B {
public static void main(String args[]) {
      A encap = new A();
encap.setName("CCIT");
encap.setAge(1994);
encap.setIdNum("core java");

System.out.print("Name : " + encap.getName() + " Age : " + encap.getAge() +" "+
"ID : "+encap.getIdNum());
   }
}
```

Inheritance: Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

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

extends Keyword: extends is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

For Example:-

```
public class Calculation {
int z;

public void addition(int x, int y) {
      z = x + y;
System.out.println("The sum of the given numbers:"+z);
```

```
   }
public void Subtraction(int x, int y) {
      z = x - y;
System.out.println("The difference between the given numbers:"+z);
   }
}
class Calculation2 extends Calculation {
public void multiplication(int x, int y) {
      z = x * y;
System.out.println("The product of the given numbers:"+z);
   }
public static void main(String args[]) {
int a = 20, b = 10;
      Calculation2 demo = new Calculation2();
demo.addition(a, b);
demo.Subtraction(a, b);
demo.multiplication(a, b);
   }
}
```

For Example: In which program use the different variable in use same value.

```
public class ths
{
    int i;
    void setvalue(int a)
    {
    i = a;
}
void show()
{
    System.out.println("this is value of i=" +i);
}
}
class test
{
    public static void main(String[] args)
    {
```

```
        ths t=new ths();
        t.setvalue(15);
        t.show();
    }
}
```

This keyword: The this keyword refers to the current object in a method or constructor. The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method.

For Example:

```
class ths
{
    int i;
    void setvalue(int i)
    {
    this.i = i;
}

void show()
{
    System.out.println("this is value of i=" +i);
}
}
class test
{
    public static void main(String[] args)
    {
        ths t=new ths();
        t.setvalue(15);
        t.show();
    }
}
```

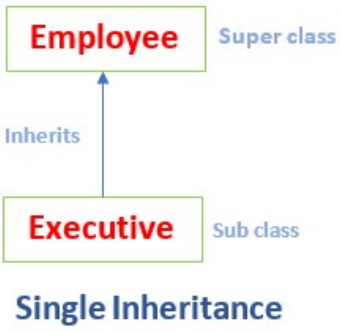
Super Keyword: The super keyword refers to superclass (parent) objects. It is used to call superclass methods, and to access the superclass constructor. The most common use of the super keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name.

For Example:

```
class A
{
    int a=21;
    void display()
    {
        System.out.println("Hello World");
        System.out.println("Value of a="+a);
    }
    /*void getvalue()
    {
        System.out.println("Hello");
    }  */
}
class B extends A
{
int a=34;
 void show()
 {
     System.out.println("Value of a="+super.a);
 }   public static void main(String[] args)
    {
        B a1=new B();
        //a1.show();
       a1.show();
        //a1.getvalue();
    }
}
```

Types of Inheritance: Java supports the following four types of 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.



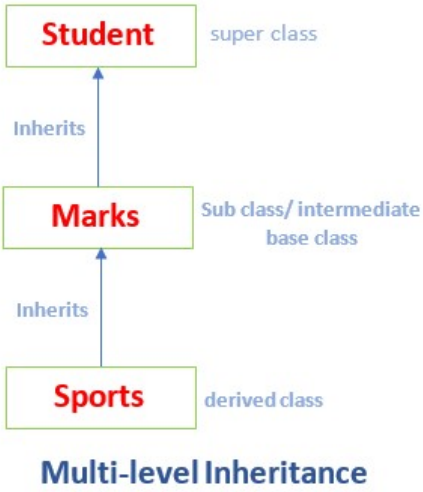
Single Inheritance

For Example:

```
class one
{
```

```
  int salary=45000,bonus=3000;
void fun()
{
  System.out.println("hello this is single inheritance");
}
}
public class arr extends one
{

public static void main(String args[])
{
arr obj=new arr();
System.out.println("Total salary credited: "+obj.salary);
System.out.println("Bonus of six months: "+obj.bonus);
}
}
```

Multi-level Inheritance: In multi-level inheritance, a class is derived from a class which is also derived from another class is called multi-level inheritance. In simple words, we can say that a class that has more than one parent class is called multi-level inheritance. Note that the classes must be at different levels. Hence, there exists a single base class and single derived class but multiple intermediate base classes.

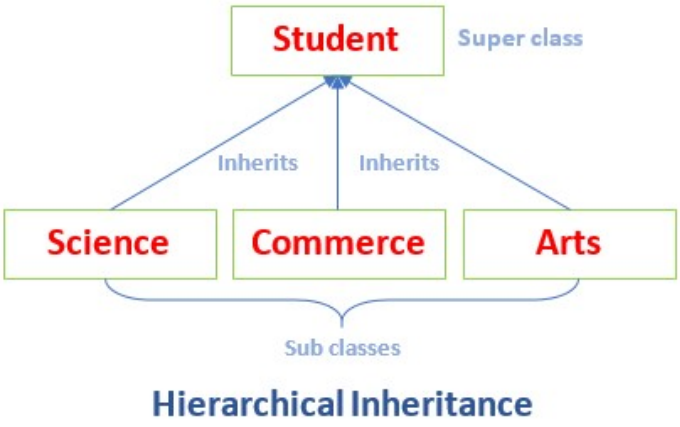


Multi-level Inheritance

For Example:

```
class one
{

public void first()
{
  System.out.println("hello this is first multilevel inheritance");
}
}
class two extends one
{
```

```
public void second()
{
  System.out.println("hello this is second multilevel inheritance");
}
}
public class arr extends two
{

public static void main(String args[])
{
arr obj=new arr();
obj.first();
obj.second();
}
}
```



Hierarchical Inheritance

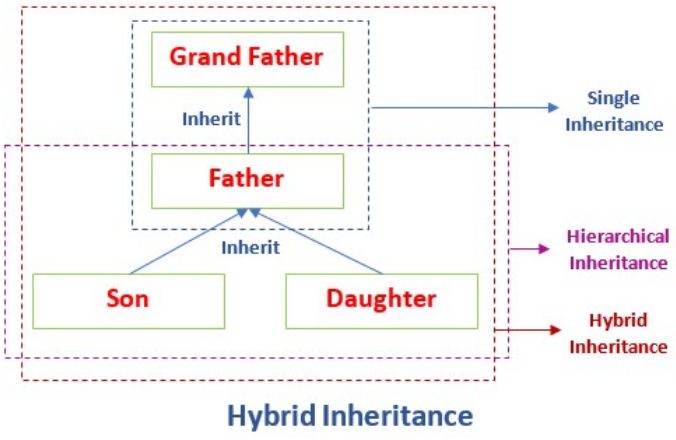
Hierarchical Inheritance: If a number of classes are derived from a single base class, it is called hierarchical inheritance.

For Example:

```
class one
{

public void first()
{
  System.out.println("hello this is first Hierarchical inheritance");
}
}
class two extends one
{

public void second()
{
  System.out.println("hello this is second Hierarchical Inheritance");
}
}
```

```
class three extends one
{

public void third()
{
  System.out.println("hello this is third Hierarchical Inheritance");
}
}
public class arr extends one
{
 public void fourth()
 {
  System.out.println("this is final funtion");
 }
public static void main(String args[])
{
arr obj1=new arr();
obj1.fourth();
obj1.first();
System.out.println();
one obj2=new arr();
obj2.first();
System.out.println();
two obj3=new two();
obj3.first();
obj3.second();
System.out.println();
three obj4=new three();
obj4.first();
obj4.third();
}
}
```

Hybrid Inheritance: Hybrid means consist of more than one. Hybrid inheritance is the combination of two or more types of inheritance.



For Example:

```
//parent class
class GrandFather
{
public void show()
{
System.out.println("I am grandfather.");
}
}
//inherits GrandFather properties
class Father extends GrandFather
{
public void show()
{
System.out.println("I am father.");
}
}
//inherits Father properties
class Son extends Father
{
public void show()
{
System.out.println("I am son.");
}
}
//inherits Father properties
public class Daughter extends Father
{
public void show()
{
System.out.println("I am a daughter.");
}
public static void main(String args[])
{
Daughter obj = new Daughter();
obj.show();
```

```
}
}
```

Note: Multiple inheritances is not supported in Java.

Polymorphism: Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

For Example:

```
class animal
{
    void display()
    {
        System.out.println("there are disfferent animals in forest");
        }
}
class dog extends animal
{
    void display()
    {
        System.out.println("there are dog house");
        }

}
class cat extends animal
 {

void display()
{
    System.out.println("there are cats house");
}
}
```

```
class forest
{
    public static void main(String[] args)
    {
        animal a1=new animal();
        animal d1=new dog();
        animal c1=new cat();

        a1.display();
        d1.display();
        c1.display();
    }
}
```

Difference between in Method Overloading and Method Overriding

Method Overloading	Method Overriding
In which same name in methods.	In which same name in methods.
In which use same class.	In which use different class.
Different argument.	Same Argument.
No. of argument is different Sequence of argument is used. Type of argument is different used.	No. of argument is same. Sequence of argument is also same. Type of Argument is same.

Method Overloading (Compile time polymorphism): When there are multiple functions with the same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by change in the number of arguments or/and a change in the type of arguments.

For Example:

```
class over
{
    void show(int c)
    {
    System.out.println("Hello World");
    System.out.println("value of c="+c);
}
void show(int a, int b)
```

```
{
    System.out.println("CCIT in core java");
    System.out.println("value of a="+a);
    System.out.println("value of b="+b);
}
}
class over2
{
    public static void main(String[] args)
    {
        over o1=new over();
        o1.show(21);
        o1.show(21,312);
    }
}
```

Method overriding (Run time polymorphism): It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding. Method overriding, on the other hand, occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.

For Exmaple:

```
class animal
{
    void display()
    {
        System.out.println("there are disfferent animals in forest");
        }
}
class dog extends animal
{
    void display()
    {
        System.out.println("there are dog house");
        }
}
class cat extends animal
```

```
 {

void display()
{
    System.out.println("there are cats house");
}
}
class forest
{
    public static void main(String[] args)
    {
        animal a1=new animal();
        animal d1=new dog();
        animal c1=new cat();
        a1.display();
        d1.display();
        c1.display();
    }
}
```

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

Abstract classes may or may not contain abstract methods, i.e., methods without body (public void get();)

But, if a class has at least one abstract method, then the class must be declared abstract.

If a class is declared abstract, it cannot be instantiated.

To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it. If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

For Example:

```
abstract class vehicle
{
int a;
abstract void start();
}
class car extends vehicle
{
void start()
```

```
{
System.out.println("start with key");
}

}
class scooter extends vehicle{
void start()
    {
System.out.println("start with kick");
    }
}
class test
{
public static void main(String[] args)
    {
vehicle c1=new car();
vehicle s1=new scooter();
c1.start();
s1.start();
    }
}
```

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.
- we can have default and static methods in an interface.
- we can have private methods in an interface.

Syntax:

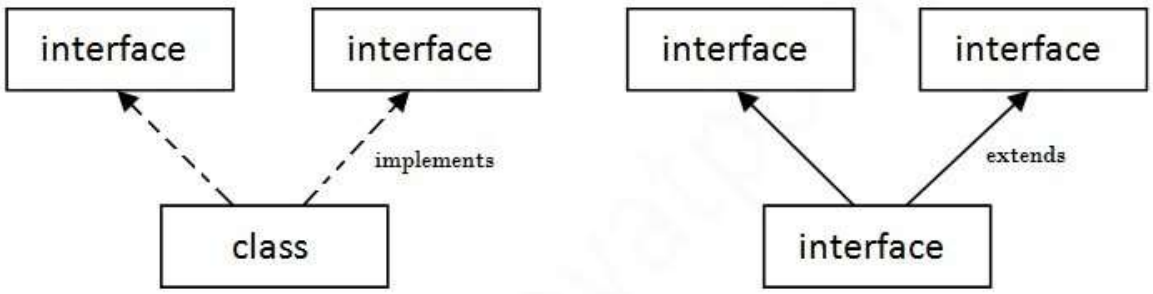
```
interface <interface_name>{

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

For Example:

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Multiple inheritance: In Java by interface: If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.

For Example:



Multiple Inheritance in Java

```
interface Printable{
  void print();
  }
  interface Showable{
  void show();
  }
  class arr implements Printable,Showable{
  public void print(){System.out.println("Hello");}
  public void show(){System.out.println("Welcome");}
```

```
  public static void main(String args[]){
  arr obj = new arr();
  obj.print();
  obj.show();
   }
  }
```

Interface inheritance: A class implements an interface, but one interface extends another interface.

For Example:

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class TestInterface4 implements Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
TestInterface4 obj = new TestInterface4();
obj.print();
obj.show();
 }
}
```

Default Method in Interface: Since Java 8, we can have method body in interface. But we need to make it default method. Let's see an example:

For Example:

```
interface Drawable{
void draw();
default void msg(){System.out.println("default method");}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
```

```
class TestInterfaceDefault{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
d.msg();
}}
```

Abstract class	Interface
Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
An abstract class can be extended using keyword "extends".	An interface can be implemented using keyword "implements".
A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.

Example:
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

For Example:

```
interface A{
void a();//bydefault, public and abstract
void b();
void c();
```

```
void d();
}

//Creating abstract class that provides the implementation of one method of A interface
abstract class B implements A{
public void c(){System.out.println("I am C");}
}

//Creating subclass of abstract class, now we need to provide the implementation of rest of the methods
class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}

//Creating a test class that calls the methods of A interface
class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

For Example: In this program use the interface in default function and abstract function.

```
interface ad{
 abstract void show();
 default void start()
 {
  System.out.println("hello how are you");
 }
}
class arr implements ad
```

```
{
public void show()
{
  System.out.println("hello 123");
}
public static void main(String[] args) {
  arr s=new arr();
  s.show();
  s.start();
}
}
```