INHERITANCE

- Inheritance is the mechanism in java by which **one class is allow to inherit the features of another class.**
- It is process of deriving a new class from an existing class.
- A class that is inherited is called **a *superclass*** and the class that does the inheriting is called a ***subclass.***
- Inheritance represents the IS-A relationship, also known as ***parent child relationship.*** The keyword used for inheritance is **extends.**

Syntax:

```
class Subclass-name extends Superclass-name
{
 //methods and fields
}
```

Here, the extends keyword indicates that we are creating a new class that derives from an existing class.

Note: The constructors of the superclass are never inherited by the subclass

Advantages of Inheritance:

- **Code reusability** - public methods of base class can be reused in derived classes
- **Data hiding** – private data of base class cannot be altered by derived class
- **Overriding**--With inheritance, we will be able to override the methods of the base class in the derived class

Example:

```
// Create a superclass.
class BaseClass
{
   int a=10,b=20;
   public void add()
   {
      System.out.println("Sum:"+(a+b));
    }
   }

   public class Main extends BaseClass
   {
      public void sub()
      {
```

```
System.out.println("Difference:"+(a-b));
    public static void main(String[] args)
    {
    Main obj=new Main();
    obj.add();
      obj.sub();
    }
}
```

Sample Output:

Sum:30

Difference:-10

Types of inheritance

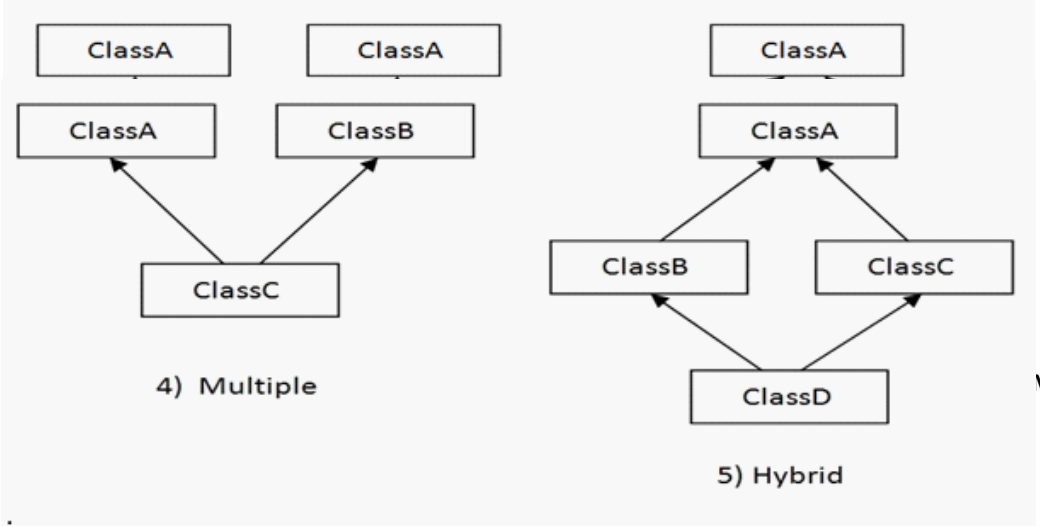
Single Inheritance :

In single inheritance**, a subclass inherit the features of one superclass**.

Example:

```
class Shape{
  int a=10,b=20;
}
class Rectangle extends Shape{
  public void rectArea(){
    System.out.println("Rectangle Area:"+(a*b));

public class Main
{
    public static void main(String[] args) {
      Rectangle obj=new Rectangle();
          obj.rectArea();
    }}
```



Multilevel Inheritance:

In Multilevel Inheritance, **a derived class will be inheriting a base class** and as well as the derived class also act as the base class to other class i.e. a derived class in turn acts as a base class for another class.

Example:

```
class Numbers{
   int a=10,b=20;
}
class Add2 extends Numbers{
   int c=30;
   public void sum2(){
      System.out.println("Sum of 2 nos.:"+(a+b));
   }
}
class Add3 extends Add2{
   public void sum3(){
      System.out.println("Sum of 3 nos.:"+(a+b+c));
   }
}
public class Main
{
     public static void main(String[] args) {
        Add3 obj=new Add3();
              obj.sum2();
              obj.sum3();
     }
}
```

Sample Output: Sum of 2 nos.:30Sum of 3 nos.:60

Hierarchical Inheritance:

In Hierarchical Inheritance, **one class serves as a superclass** (base class) for more than one sub class.

Example:

```
class Shape{
   int a=10,b=20;
}
class Rectangle extends Shape{
   public void rectArea(){
      System.out.println("Rectangle Area:"+(a*b));
   }
}
class Triangle extends Shape{
   public void triArea(){
      System.out.println("Triangle Area:"+(0.5*a*b));
   }
}
public class Main
{
    public static void main(String[] args) {
        Rectangle obj=new Rectangle();
            obj.rectArea();
        Triangle obj1=new Triangle();
            obj1.triArea();
    }
}
```

Sample Output: Rectangle

Area:200Triangle

Area:100.0

Multiple inheritance

Java does not allow multiple inheritance:

- **To reduce the complexity and simplify the language**
- ***To avoid the ambiguity caused by multiple inheritance***

For example, Consider a class C derived from two base classes A and B. Class C inheritsA and B features. If A and B have a method with same signature, there will be ambiguity to call method of A or B class. It will result in compile time error.

```
class A{
void msg(){System.out.println("Class A");}
}
class B{
```

```
void msg(){System.out.println("Class B ");}
}
class C extends A,B{//suppose if it were
  Public Static void main(String args[]){C
  obj=new C();
  obj.msg();//Now which msg() method would be invoked?
}
}
```

Sample Output:

Compile time error

Direct implementation of multiple inheritance is not allowed in Java. But it is achievable using Interfaces. The concept about interface is discussed in chapter.2.7.

Access Control in Inheritance

The following rules for inherited methods are enforced −

- Variables declared public or protected in a superclass are inheritable in subclasses.
- Variables or Methods declared private in a superclass are not inherited at all.
- Methods declared public in a superclass also must be public in all subclasses.
- Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.

Example:

```
// Create a superclass
class A{
   int x;              // default specifier
   private int y;      // private to A
   public void set_xy(int a,int b){
      x=a;
      y=b;
   }
}
// A's y is not accessible here.
class B extends A{
   public void add(){
      System.out.println("Sum:"+(x+y)); //Error: y has private access in A – not inheritable
   }
}
class Main{
```

```
    public static void main(String args[]){B
        obj=new B();
        obj.set_xy(10,20);
        obj.add();
    }
}
```

In this example since y is declared as private, it is only accessible by its own class members. Subclasses have no access to it.