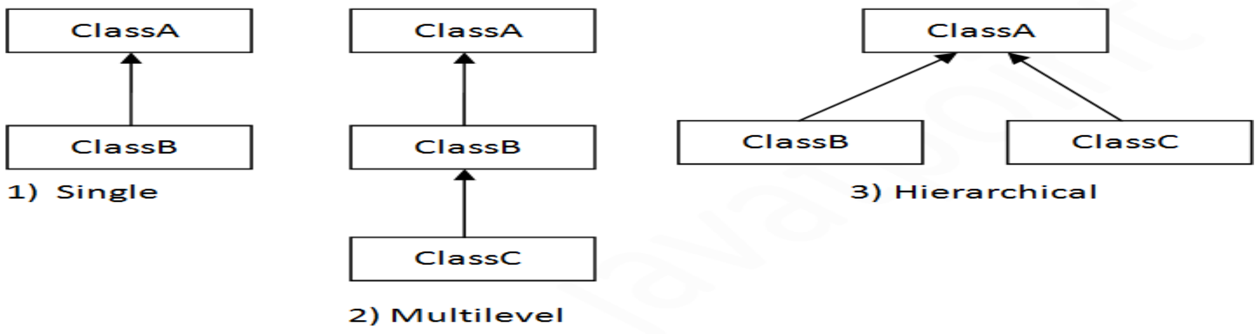
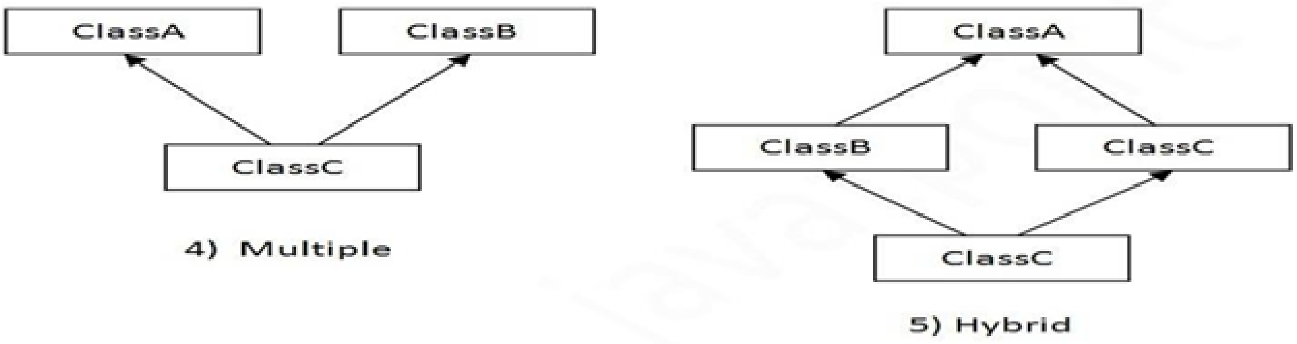
TYPES OF INHERITACE:

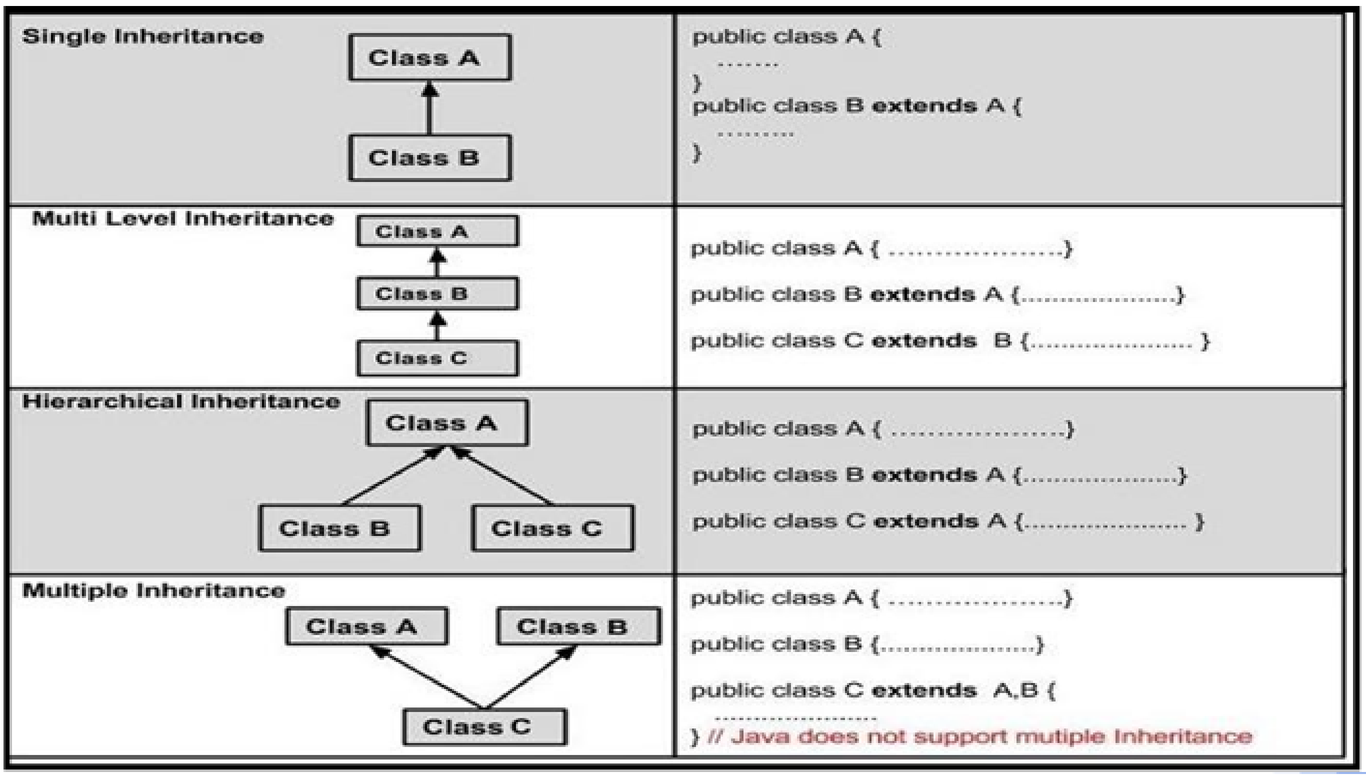
1. Single Inheritance
2. Multilevel Inheritance
3. Multiple Inheritance

Note: The following inheritance types are not directly supported in Java.

4. Hierarchical Inheritance
5. Hybrid Inheritance







Single Inheritance Class A ← Class B	public class A {
	....
	}
	public class B **extends** A {
	....
	}

Multi Level Inheritance Class A ← Class B ← Class C	public class A { ....}
	public class B **extends** A {....................}
	public class C **extends** B {..................... }

Hierarchical Inheritance Class A ← Class B, Class C	public class A { ....}
	public class B **extends** A {....................}
	public class C **extends** A {..................... }

Multiple Inheritance Class A, Class B ← Class C	public class A { ....}
	public class B {....................}
	public class C **extends** A,B {
	....
	} // Java does not support mutiple Inheritance

1. SINGLE INHERITANCE

The process of creating only one subclass from only one super class is known as **Single Inheritance.**

- ✓ Only two classes are involved in this inheritance.
- ✓ The subclass can access all the members of super class.

Example: Animal → Dog

```
class Animal
{
   void eat()
   {
       System.out.println("eating...");
   }
}
class Dog extends Animal
{
void bark()
{
     System.out.println("barking...");
  }
}
class TestInheritance
{
   public static void main(String args[])
   {
       Dog d=new Dog();
       d.bark();
       d.eat();
   }
}
```

Output:

```
$java TestInheritance
barking...
eating...
```

2. MULTILEVEL INHERITANCE:

- ✓ The process of creating a new sub class from an already inherited sub class is known as **Multilevel Inheritance**.
- ✓ Multiple classes are involved in inheritance, but one class extends only one.
- ✓ The lowermost subclass can make use of all its super classes' members.
- ✓ Multilevel inheritance is an indirect way of implementing multiple inheritance.
- ✓ **Example: Animal → Dog → BabyDog**

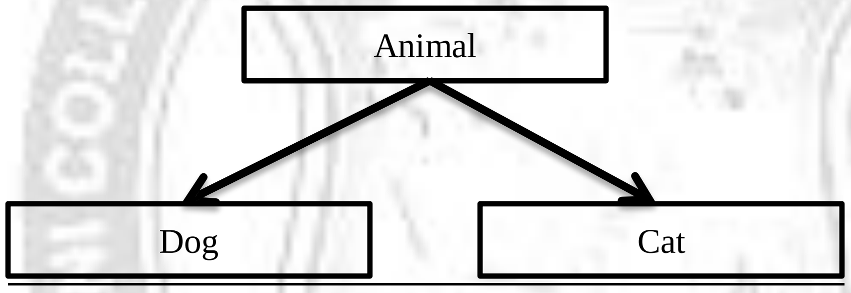
```
class Animal
{
  void eat()
 {
     System.out.println("eating...");
 }
}
class Dog extends Animal
{
   void bark()
  {
      System.out.println("barking...");
  }
 }
class BabyDog extends Dog
{
   void weep()
  {
     System.out.println("weeping...");
  }
}
class TestInheritance2
{
  public static void main(String args[])  {
       BabyDog d=new BabyDog();
       d.weep();
       d.bark();
       d.eat();
  }
}
```

Output:

```
$java TestInheritance2
weeping...
barking...
eating..
```

3. HIERARCHICAL INHERITANCE

✓ The process of creating more than one sub classes from one super class is called **Hierarchical Inheritance.**



✓ **Example:**

```
class Animal
{
void eat()
{
System.out.println("eating...");
}
}
class Dog extends Animal
{
void bark()
{
System.out.println("barking...");
}
}
class Cat extends Animal
{
void meow()
{
System.out.println("meowing...");
}
}
class TestInheritance3
{
public static void main(String args[])
{
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}
}
```

Output:

```
meowing...
eating...
```

Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritances is not supported in java.

Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error now.

```
class A  {
    void msg()
   {
       System.out.println("Hello");
   }
}
class B  {
   void msg()
   {
        System.out.println("Welcome");
   }
}
class C extends A,B  // this is multiple inheritance which is ERROR
  {
         Public Static void main(String args[])
         {
            C obj=new C();
            obj.msg();//Now which msg() method would be i
nvoked?
         }
}
```

Output

```
Compile Time Error
```

Multiple Inheritance using Interface

```
interface Printable  {
     void print();
}

interface Showable  {
    void show();
}
class A implements Printable, Showable  {
         public void print() {
              System.out.println("Hello");
           }
        public void show() {
            System.out.println("Welcome");
        }
       public static void main(String args[]) {
            A obj = new A();
            obj.print();
            obj.show();
       }
}
```

Output:

```
Hello
Welcome
```

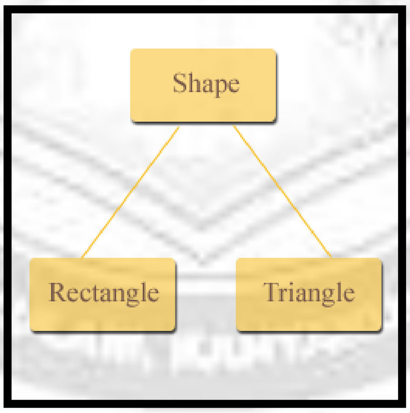
2.4.1: PROTECTED MEMBER

The private members of a class cannot be directly accessed outside the class. Only methods of that class can access the private members directly. However, sometimes it may be necessary for a subclass to access a private member of a superclass. If you make a private member public, then anyone can access that member. So, if a member of a superclass needs to be (directly) accessed in a subclass then you must declare that member **protected**.

Following table describes the difference

Modifier	Class	Package	subclass	World
public	Yes	Yes	Yes	Yes
private	Yes	No	No	No
protected	Yes	Yes	Yes	No

Following program illustrates how the methods of a subclass can directly access a protected member of the superclass.



Consider two kinds of shapes: **rectangles and triangles**. These two shapes have certain common properties height and a width (or base).

This could be represented in the world of classes with a **class Shapes** from which we would derive the two other ones : Rectangle and Triangle

Program : (Shape.java)

```
public class Shape
{
    protected double height; // To hold height.
    protected double width;  //To hold width or base
```

```
     public void setValues(double height, double width)
    {
           this.height = height;
           this.width = width;
   }
}
```

Program : (Rectangle.java)

```
public class Rectangle extends Shape
{
      public double getArea()
     {
        return height * width; //accessing protected members
     }
}
```

Program : (Triangle.java)

```
public class Triangle extends Shape
{
               public double getArea()
             {
                   return height * width / 2; //accessing protected members
             }
}
```

Program : (TestProgram.java)

```
public class TestProgram
{
        public static void main(String[] args)
       {
             //Create object of Rectangle.
              Rectangle rectangle = new Rectangle();

            //Create object of Triangle.
            Triangle triangle = new Triangle();

            //Set values in rectangle object
           rectangle.setValues(5,4);
         //Set values in trianlge object
         triangle.setValues(5,10);

        // Display the area of rectangle.
         System.out.println("Area of rectangle : " +
         rectangle.getArea());

       // Display the area of triangle.
        System.out.println("Area of triangle : " +
        triangle.getArea());
     }
}
```

Output :

```
Area of rectangle : 20.0
Area of triangle : 25.0
```

2.4.2: CONSTRUCTORS IN SUB – CLASSES

In Java, constructor of base class with no argument gets automatically called in derived class constructor.

When Constructors are Called?

Constructors are called in order of derivation, from superclass to subclass. Because a superclass has no knowledge of any subclass, any initialization it needs to perform is separate from and possibly prerequisite to any initialization performed by the subclass. Therefore, it must be executed first.

Example:

```
class A
{
   A()
   {      System.out.println(" Inside A's Constructor");    }
}

class B extends  A
{
   B()
   {    System.out.println(" Inside B's Constructor");    }
}
class C  extends  B
{
   C()
   {    System.out.println(" Inside C's Constructor");    }
}
class CallingCons
{
   public static void main(String args[])
  {
      C objC=new C();
   }
}
```

Output:

```
    Inside          A's
    Constructor
    Inside          B's
    Constructor
    Inside          C's
    Constructor
```

Program Explanation:

In the above program, we have created three classes A, B and C using multilevel inheritance concept. Here, constructors of the three classes are called in the order of derivation. Since **super()** must be the first statement executed in subclass's constructor, this order is the same whether or not **super()** is used. If **super()** is not used, then the default or parameterless constructor of each superclass will be executed. When inheriting from another class, super() has to be called first in the constructor. If not, the compiler will insert that call. This is why super constructor is also invoked whena Sub object is created.

After compiler inserts the super constructor, the sub class constructor looks like thefollowing:

```
B()
{
        super();
        System.out.println("Inside B’s Constructor");
}
C()
{
        super();

        System.out.println("Inside C’s Constructor");
}
```