Relationship between the classes :

- In java, In between the classes we have two types of relation :

  1) IS-A Relation (We can achieve by using Inheritance Concept)
  2) HAS-A Relation (We can achieve by using Association concept)

Example of IS-A Relation :

```
class Bird
{

}
class Parrot extends Bird    //[IS-A Relation, Parrot IS-A Bird]
{

}
```

Example of HAS-A Relation :

```
public class Engine
{

}

public class Car
{
   private Engine engine;   //[Car HAS-An Engine]

}
```

Inheritance [IS-A Relation]

```
public class A
{
   public void sum(int x, int y)
   {
   }

   public void sub(int x, int y)
   {
   }
}
```

```
public class B
{
   public void sum(int x, int y)
   {
   }

   public void sub(int x, int y)
   {
   }

   public void mul(int x, int y)
   {
   }

   public void div(int x, int y)
   {
   }
}
```

Note : In the B class we have duplicate code.

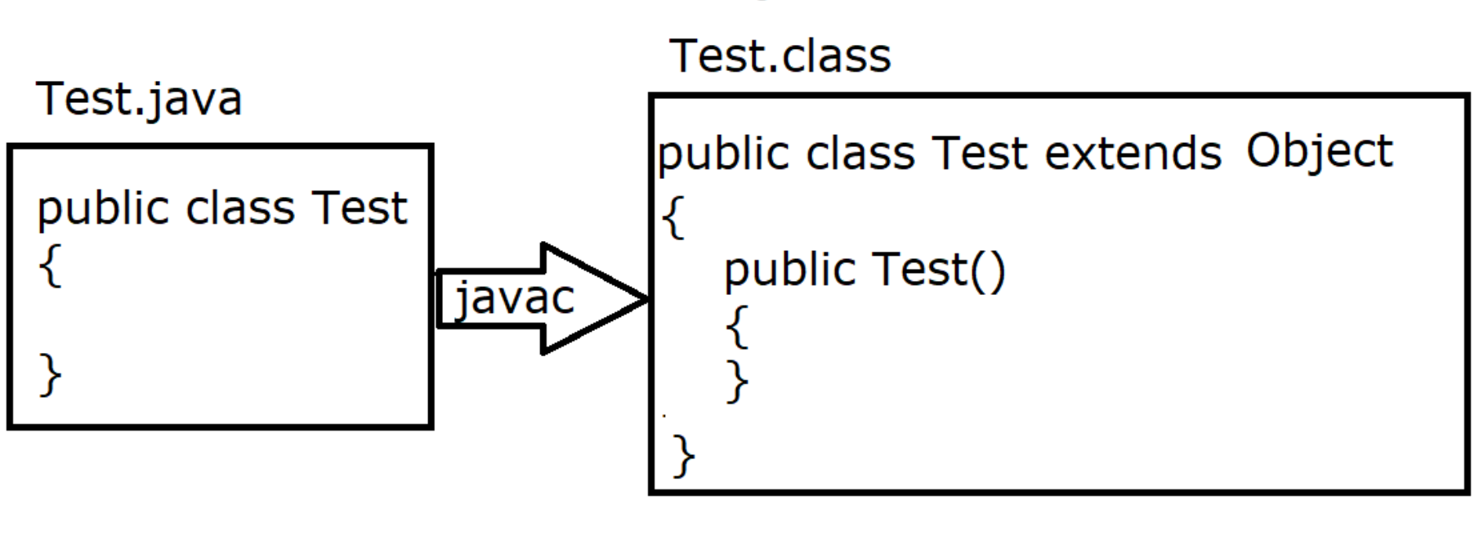
OOP says we should always **reuse our code rather then re-creating it.**

In order to reusability of the code we should use Inheritance Concept as shown below :

```
public class A
{
   public void sum(int x, int y)
   {
   }

   public void sub(int x, int y)
   {
   }
}

public class B extends A
{
   public void mul(int x, int y)
   {
   }

   public void div(int x, int y)
   {
   }
}
```

Some points :

- We can achieve inheritance by using extends keyword
- In this context A is called Parent class OR Super class
- In this context B is called Child class OR Sub class
- By default java.lang.Object class is the super class of all the classes we have in java.



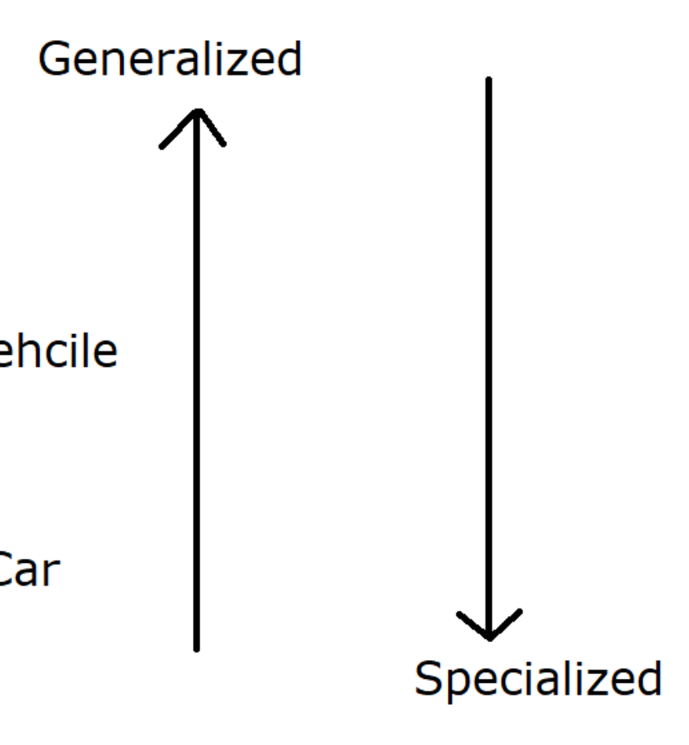
```
public class Test
{

}
```

```
public class Test extends  Object
{
   public Test()
   {
   }
}
```

- **Deriving** a new class (class B) from existing class (class A) in a such a way that the new class will **acquire** all the features and properties (except private) is called Inheritance.
- It is one most important feature of OOP, which provides **"Code Reusability"**.
- Using inheritance, all the properties of parent OR super class will be available to the sub classes so the sub class need not to start the process from begning onwards.
- While using Inheritance, We should always create the object for sub class OR more specialized class.
- In inheritance tree, When we move towards upward direction then we will get more **generalized** properties, on the other hand if we move towards downward direction then we will get more **specialized** properties.

Example :

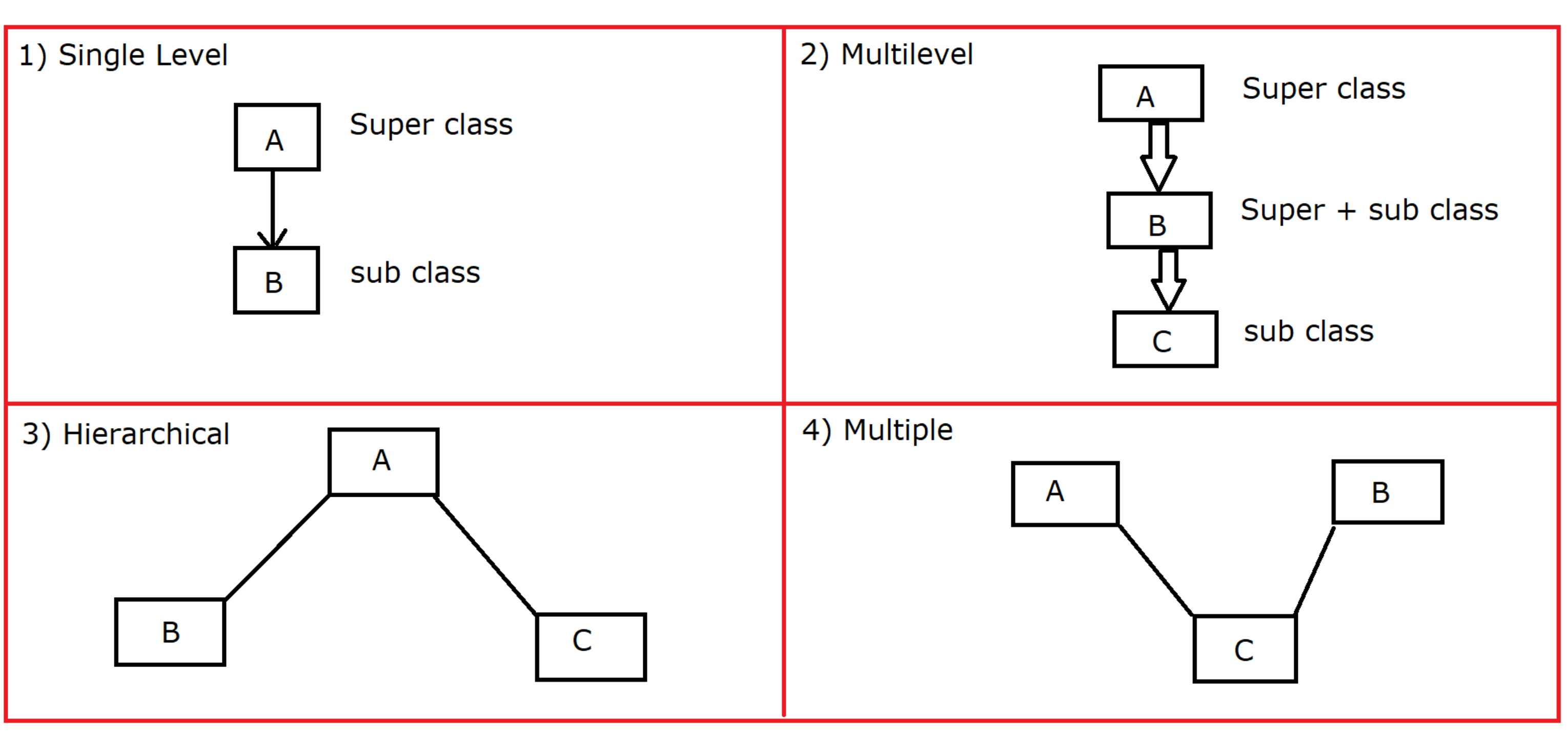
```
class Vehicle
{
}

class Car extends Vehcile
{
}

class Ford extends Car
{
}
```



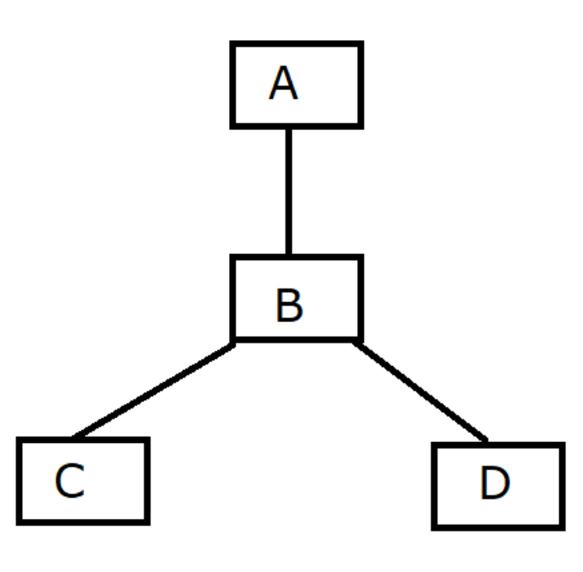
Types of Inheritance :

- We have five types of Inheritance in java

  1) Single Level Inehritance
  2) Multilevel Inehritance
  3) Hierarchical Inehritance
  4) Multiple Inehritance (Not supported by using class)
  5) Hybrid Inehritance (combination of any two)



Hybrid Inheritance



//Programs :

Single Level Inheritance Program :

```
package com.ravi.inheritane;

class Parent
{
    public void house()
    {
        System.out.println("3 BHK House");
    }
}
class Child extends Parent
{
    public void car()
    {
        System.out.println("Audi Car");
    }
}
public class InheritanceDemo
{
    public static void main(String[] args)
    {
        Child child = new Child();
        child.house();
        child.car();
    }
}
```

Another Program on Single Level Inheritance :

```
package com.ravi.inheritane;

class Super
{
    private int x,y;

    public void setData(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public int getX()
    {
        return x;
    }

    public int getY()
    {
        return y;
    }
}
class Sub extends Super
{
    public void showData()
    {
        System.out.println("x value is :"+getX());
        System.out.println("y value is :"+getY());
    }
}
public class SingleLevelDemo
{
    public static void main(String[] args)
    {
        Sub s1 = new Sub();
        s1.setData(100, 200);
        s1.showData();
    }
}
```

- Constructors are not inherited in java because sub class name and constructor name both will not be same

Example :

```
class Alpha
{
  public Alpha()
  {
  }
}
class Beta extends Alpha
{
  public Beta()
  {
  }
}
```

main method :

```
Beta b1 = new Beta();
b1.Alpha(); //Invalid
Alpha a1 = new Alpha(); // If we create two object then
                           there is no use of Inheritance
```