## 1. What is inheritance and explain types of inheritance with examples

The process by which one class acquires the properties(data members) and functionalities(methods) of another class is called **inheritance**. The main aim of inheritance is code reusability.

To inherit a class we use extends keyword. Here class XYZ is child class and class ABC is parent class. The class XYZ is inheriting the properties and methods of ABC class.

**Syntax:-**

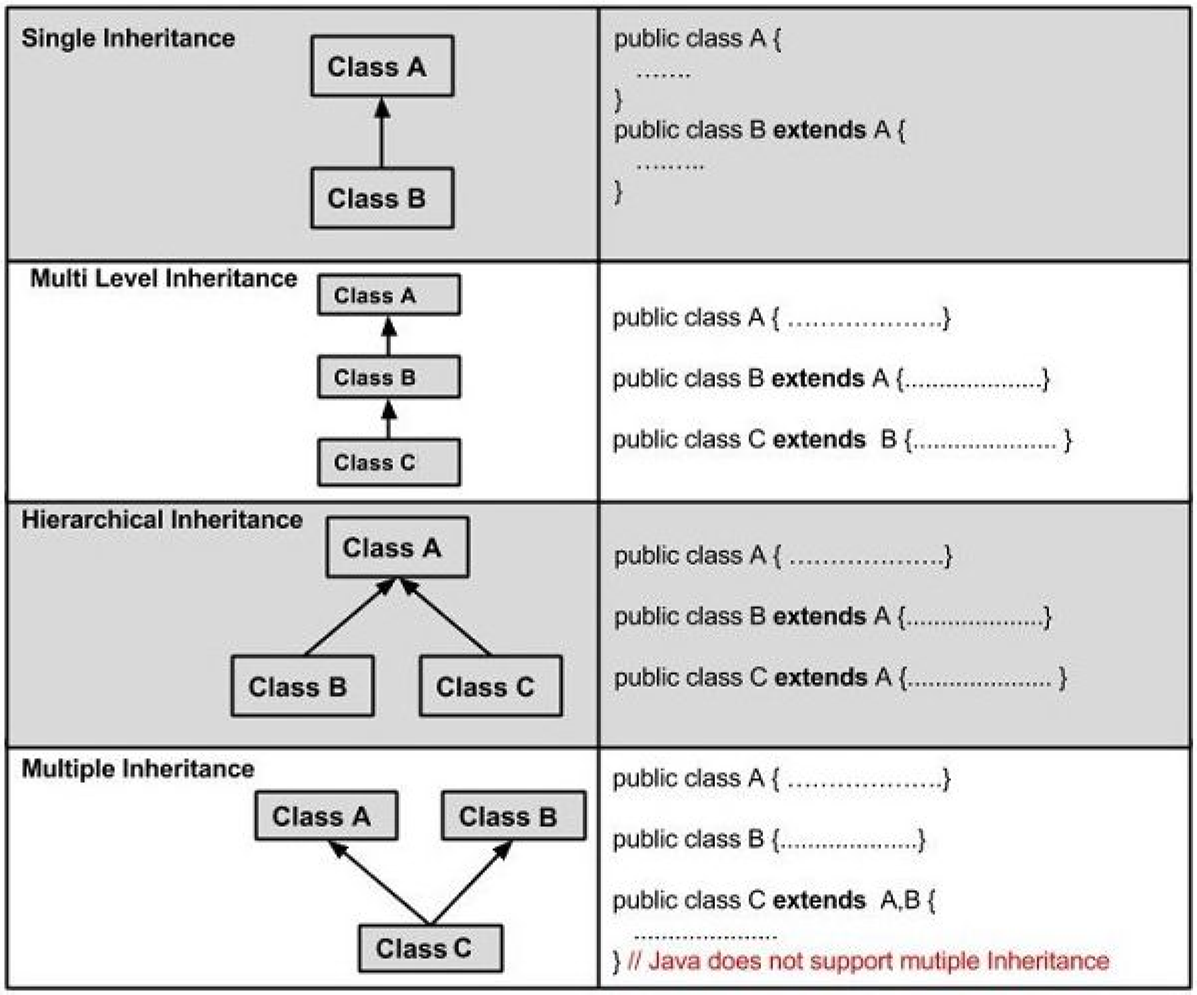
```
class ABC
{
}

class XYZ extends ABC
{
}
```

In the above syntax, ABC is called as parent class or base class or superclass, XYZ is called as child class or derived class or subclass.

**Types of Inheritance:**

- Single inheritance
- Multilevel inheritance
- Hierarchical inheritance
- Multiple inheritance
- Hybrid inheritance

Java doesn't support multiple inheritance directly, it is achieved by the interface concept.



| | |
|---|---|
| Single Inheritance | public class A {<br>……. <br>}<br>public class B **extends** A {<br>………<br>} |
| Multi Level Inheritance | public class A { ……………….}<br>public class B **extends** A {……………….}<br>public class C **extends** B {………………… } |
| Hierarchical Inheritance | public class A { ……………….}<br>public class B **extends** A {……………….}<br>public class C **extends** A {………………… } |
| Multiple Inheritance | public class A { ……………….}<br>public class B {……………….}<br>public class C **extends** A,B {<br>…………………<br>} // Java does not support mutiple Inheritance |

**Example of  inheritance:-**

```
class Teacher {
  String designation = "Teacher";
  String collegeName = "Beginnersbook";
  void does(){
        System.out.println("Teaching");
  }
}

public class PhysicsTeacher extends Teacher{
  String mainSubject = "Physics";
  public static void main(String args[]){
        PhysicsTeacher obj = new PhysicsTeacher();
        System.out.println(obj.collegeName);
        System.out.println(obj.designation);
        System.out.println(obj.mainSubject);
```

```
          obj.does();
   }
}
```

**2. Let’s consider the example of vehicles like car, bike, and bus they have common functionalities. So we make an interface and put all these common functionalities. And lets  Bike, car and bus implement all these functionalities in their own class in their own way.**

```
package sai;
interface Fuctionatlities  //interface for all funcionalities
{
          void vahiclebreak();
          void gare();
          void accelarator();
}
class Car implements Fuctionatlities  //car class
{
          public void vahiclebreak()
          {
                    System.out.println("Apply car break");
          }
          public void gare()
          {
                    System.out.println("change the car gare");
          }
          public void accelarator()
          {
                    System.out.println("increse the car accelarator");
          }
}
class Bike implements Fuctionatlities  //bike class
{
          public void vahiclebreak()
          {
                    System.out.println("Apply bike break");
          }
          public void gare()
```

```
        {
                System.out.println("change the bike gare");
        }
        public void accelarator()
        {
                System.out.println("increse the bike accelarator");
        }
}
class Bus implements Fuctionatlities  //bus
{
        public void vahiclebreak()
        {
                System.out.println("Apply bus break");
        }
        public void gare()
        {
                System.out.println("change the bus gare");
        }
        public void accelarator()
        {
                System.out.println("increse the bus accelarator");
        }
}
public class Vahicles{   //main class
        public static void main(String args[])
        {
                System.out.println("functionalities of Car:");
                Car car=new Car();
                car.accelarator();
                car.gare();
                car.vahiclebreak();

                System.out.println("\nfunctionalities of Bike:");
                Bike bike=new Bike();
                bike.accelarator();
                bike.gare();
                bike.vahiclebreak();
                System.out.println("\nfunctionalities of Bus:");
                Bus bus=new Bus();
                bus.accelarator();
                bus.gare();
                bus.vahiclebreak();

        }
}
```

**3. Consider a base class is “shape” and each shape has a color, size, and areas. From this, specific types of shapes are derived(inherited)-circle, square, triangle, each of which may have additional characteristics and behaviors. For example, certain shapes can be flipped. Some behaviors may be different, such as when you want to calculate the area of a shape. The type hierarchy embodies both the similarities and differences between the shapes.**

```
package training;
class Shapes {
String color;
float area;
float size;
}
class Circle extends Shapes
{
        void circolor()
        {
                color="red";
                System.out.println("circle color is:" +color);
        }
        void cirarea(float r)
        {
                area=(float)((3.14)*(r*r));
                System.out.println("area of circle is:" + area);
        }
        void cirsize(float r)
        {
                size=(float)(2*(3.14)*r);
                System.out.println("size of circle is:" + size);
        }
}
class Square extends Shapes
{
        void squcolor()
        {
                color="green";
                System.out.println("square color is:" +color);
        }
        void squarea(float r)
        {
                area=(float)(r*r);
                System.out.println("area of circle is:" + area);
        }
        void squsize(float r)
        {
                size=(float)(4*r);
                System.out.println("size of circle is:" + size);
        }
}
class Triangle extends Shapes
{
        void tricolor()
        {
                color="blue";
                System.out.println("triangle color is:" +color);
        }
        void triarea(float b,float h)
        {
                area=(float)(b*h)/2;
                System.out.println("area of triangle is:" + area);
        }
        void trisize(float a,float b,float c)
        {
                size=(float)(a+b+c);
                System.out.println("size of triangle is:" + size);
        }
```

```
}
class Shape
{
          public static void main(String[] args) {
                    System.out.println("Square color, area and perimeter:");
                    Square square =new Square();
                    square.squcolor();
                    square.squarea(5);
                    square.squsize(5);
                    System.out.println("\nTriangle color, area and perimeter:");
                    Triangle triangle =new Triangle();
                    triangle.tricolor( );
                    triangle.triarea(2, 4);
                    triangle.trisize(2, 4, 8);
                    System.out.println("\nCircle color, area and perimeter:");
                    Circle circle =new Circle();
                    circle.circolor();
                    circle.cirarea(7);
                    circle.cirsize(7);

          }

}
```

## 4. What is encapsulation, mention its advantages? explain with an example

**Ans:-** Encapsulation simply means binding object state(fields) and behavior (methods) together. If you are creating the class, you are doing encapsulation.The whole idea behind encapsulation is to hide the implementation details from users. If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class.

## Advantages of encapsulation

1. It improves maintainability and flexibility and re-usability: for e.g. In the above code the implementation code of void setEmpName(String name) and String getEmpName() can be changed at any point of time. Since the implementation is purely hidden for outside classes they would still be accessing the private field empName using the same methods setEmpName(String name) and getEmpName()). Hence the code can be maintained at any point of time without breaking the classes that uses the code. This improves the re-usability of the underlying class.
2. The fields can be made read-only (If we don't define setter methods in the class) or write-only (If we don't define the getter methods in the class). For e.g. If we have a field(or variable) that we don't want to be changed so we simply define the variable as private and instead of set and get both we just need to define the get method for that variable. Since the set method is not present there is no way an outside class can modify the value of that field.
3. User would not be knowing what is going on behind the scene. They would only be knowing that to update a field call set method and to read a field call get method but what these set and get methods are doing is purely hidden from them.

implement encapsulation in java:

1) Make the instance variables private so that they cannot be accessed directly from outside the class. You can only set and get values of these variables through the methods of the class.

2) Have getter and setter methods in the class to set and get the values of the fields.

```
class EncapsulationDemo{
   private int ssn;
}

   private String empName;
   private int empAge;

   //Getter and Setter methods
   public int getEmpSSN(){
       return ssn;
   }

   public String getEmpName(){
       return empName;
   }

   public int getEmpAge(){
       return empAge;
   }

   public void setEmpAge(int newValue){
       empAge = newValue;
   }

   public void setEmpName(String newValue){
       empName = newValue;
   }

   public void setEmpSSN(int newValue){
       ssn = newValue;
   }
}
public class EncapsTest{
   public static void main(String args[]){
        EncapsulationDemo obj = new EncapsulationDemo();
        obj.setEmpName("Mario");
        obj.setEmpAge(32);
        obj.setEmpSSN(112233);
        System.out.println("Employee Name: " + obj.getEmpName());
        System.out.println("Employee SSN: " + obj.getEmpSSN());
        System.out.println("Employee Age: " + obj.getEmpAge());
   }
}
```

## 5. What is method overloading and method overriding? explain with an example

## Ans:-

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different.

**Three ways to overload a method**

In order to overload a method, the argument lists of the methods must differ in either of these:

**1. Number of parameters.**

For example: This is a valid case of overloading

add(int, int)

add(int, int, int)

**2. Data type of parameters.**

For example:

add(int, int)

add(int, float)

**3. Sequence of Data type of parameters.**

For example:

add(int, float)

add(float, int)

**Example:-**

```
class DisplayOverloading
{
    public void disp(char c)
    {
        System.out.println(c);
    }
    public void disp(char c, int num)
    {
        System.out.println(c + " "+num);
    }
}
class Sample
{
   public static void main(String args[])
```

```
  {
     DisplayOverloading obj = new DisplayOverloading();
     obj.disp('a');
     obj.disp('a',10);
  }
}
```

## Method Overriding:-

Declaring a method in **sub class** which is already present in **parent class** is known as method overriding. Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class. In this case the method in parent class is called overridden method and the method in child class is called overriding method.

**Example:-**

```
class Human{
  //Overridden method
  public void eat()
  {
    System.out.println("Human is eating");
  }
}
class Boy extends Human{
  //Overriding method
  public void eat(){
    System.out.println("Boy is eating");
  }
  public static void main( String args[]) {
    Boy obj = new Boy();
    //This will call the child class version of eat()
    obj.eat();
  }
}
```

## 6. Consider a class Rectangle, it should contain 5 printArea methods accepting different parameter

```
public class Rectangle {

   public void printArea(int length, int breadth)
        {
        System.out.println(2*(length+breadth));
   }
        public void printArea(float length, int breadth)
        {
        System.out.println(2*(length+breadth));
        }
        public void printArea(int length, float breadth)
        {   System.out.println(2*(length+breadth));
        }
        public void printArea(byte length, byte breadth)
        {
        System.out.println(2*(length+breadth));
        }
        public void printArea(int length, byte breadth)
        {
        System.out.println(2*(length+breadth));
        }
        public static void main(String args [])
        {
        Rectangle rectangle=new Rectangle();
        rectangle.printArea(10,15);
        rectangle.printArea(10.0f,15 );
        rectangle.printArea(11,15.0f );
        rectangle.printArea((byte)11,(byte)15);
        rectangle.printArea(10,(byte)15);

   }

}
```

**7. Consider a class Plants, it should contain the methods releases oxygen and accepts carbon dioxide and this class should be extends by different plants(for example SunflowerPlant extends Plants) and**

**those classes should contain same methods (extends by at least 3 classes).**

```
package sai;
class Plants
{
    public void releasesOxygen()
         {
         System.out.println("Plants releases oxygen :");
         }
         public void acceptsCarbondioxide()
         {
         System.out.println("Plants accepts oxygen :");
          }
}
class SunFlowerPlant extends Plants
{
    public void releasesOxygen()
         {
         System.out.println("SunFlower plant releases oxygen :");
         }
         public void acceptsCarbondioxide()
         {
         System.out.println("SunFlower plant accepts oxygen :");
         }
}

class Lilly extends Plants
{
    public void releasesOxygen()
         {
         System.out.println("Lilly plant releases oxygen :");
         }
         public void acceptsCarbondioxide()
         {
         System.out.println("Lilly plant accepts oxygen :");
         }
}
```

```
class Rose extends Plants
{
     public void releasesOxygen()
            {
            System.out.println("Rose plants releases oxygen :");
            }
            public void acceptsCarbondioxide()
            {
            System.out.println("Rose plants accepts oxygen :");
            }
 }
class Apple extends Plants
{
    public void releasesOxygen()
           {
           System.out.println("Apple pants releases oxygen :");
            }
           public void acceptsCarbondioxide()
           {
           System.out.println("Apple plants accepts oxygen :");
           }
}
public class Tree
{
     public static void main(String args [])
            {
            SunFlowerPlant sunFlowerPlant =new SunFlowerPlant();
            sunFlowerPlant.releasesOxygen();
            sunFlowerPlant.acceptsCarbondioxide();
            Lilly Lilly=new Lilly();
            Lilly.releasesOxygen();
            Lilly.acceptsCarbondioxide();
            Rose RosePlant=new Rose();
            RosePlant.releasesOxygen();
            RosePlant.acceptsCarbondioxide();
            Apple ApplePlant = new Apple();
            ApplePlant.releasesOxygen();
            ApplePlant.acceptsCarbondioxide();
```

```
        }
}
```