What is Inheritance in Java Programming?

Inheritance is one of the features of Object-Oriented Programming (OOPs). Inheritance allows a class to use the properties and methods of another class. In other words, the derived class inherits the states and behaviors from the base class. The derived class is also called subclass and the base class is also known as super-class. The derived class can add its own additional variables and methods. These additional variable and methods differentiates the derived class from the base class.

Inheritance is a compile-time mechanism. A super-class can have any number of subclasses. But a subclass can have only one superclass. This is because Java does not support multiple inheritance.

The superclass and subclass have **"is-a"** relationship between them. Let's have a look at the example below.

Inheritance Example

Let's consider a superclass `Vehicle`. Different vehicles have different features and properties however there few of them are common to all. Speed, color, fuel used, size are few which are common to all. Hence we can create a class 'Vehicle' with states and actions that are common to all vehicles. The subclass of this superclass can be any type of vehicle. Example: Class Car A has all the features of a vehicle. But it has its own attributes which makes it different from other subclasses. By using inheritance we need not rewrite the code that we've already used with the `Vehicle`. The subclass can also be extended. We can make a class 'Sports Car' which extends 'Car'. It inherits the features of both 'Vehicle' and 'Car'.

The keyword used for inheritance is extends. Syntax:

```
public class ChildClass extends BaseClass  {
   // derived class methods extend and possibly override
}
```

Here is the complete example:

```
// A class to display the attributes of the vehicle
class Vehicle {
   String color;
   int speed;
   int size;
   void attributes() {
      System.out.println("Color : " + color);
      System.out.println("Speed : " + speed);
      System.out.println("Size : " + size);
   }
}

// A subclass which extends for vehicle
class Car extends Vehicle {
   int CC;
   int gears;
   void attributescar() {
      // The subclass refers to the members of the superclass
      System.out.println("Color of Car : " + color);
      System.out.println("Speed of Car : " + speed);
      System.out.println("Size of Car : " + size);
      System.out.println("CC of Car : " + CC);
      System.out.println("No of gears of Car : " + gears);
   }
}
public class Test {
   public static void main(String args[]) {
      Car b1 = new Car();
      b1.color = "Blue";
      b1.speed = 200 ;
      b1.size = 22;
      b1.CC = 1000;
      b1.gears = 5;
      b1.attributescar();
   }
}
```

The output is

```
Color of Car : Blue
Speed of Car : 200
Size of Car : 22
CC of Car : 1000
No of gears of Car : 5
```

Note:
The derived class inherits all the members and methods that are declared as public or protected. If declared as private it cannot be inherited by the derived classes. The private members can be accessed only in its own class. The private members can be accessed through assessor methods as shown in the example below. The derived class cannot inherit a member of the base class if the derived class declares another member with the same name.

```
// A class to display the attributes of the vehicle
class Vehicle {
   String color;
   private int speed;
   private int size;
   public int getSize() {
      return size;
   }
   public int getSpeed() {
      return speed;
   }
   public void setSize(int i) {
      size = i;
   }
   public void setSpeed(int i) {
      speed = i;
   }
}

// A subclass which extends for vehicle
class Car extends Vehicle {
   int CC;
   int gears;
   int color;
   void attributescar() {
       // Error due to access violation
      // System.out.println("Speed of Car : " + speed);
      // Error due to access violation
      //System.out.println("Size of Car : " + size);
   }
}
public class Test {
   public static void main(String args[]) {
      Car b1 = new Car();
      // the subclass can inherit 'color' member of the superclass
      b1.color = 500;
      b1.setSpeed(200) ;
      b1.setSize(22);
      b1.CC = 1000;
      b1.gears = 5;
      // The subclass refers to the members of the superclass
      System.out.println("Color of Car : " + b1.color);
      System.out.println("Speed of Car : " + b1.getSpeed());
      System.out.println("Size of Car : " + b1.getSize());
      System.out.println("CC of Car : " + b1.CC);
      System.out.println("No of gears of Car : " + b1.gears);
   }
}
```

The output is:

```
Color of Car : 500
Speed of Car : 200
Size of Car : 22
CC of Car : 1000
No of gears of Car : 5
```

Constructors and Inheritance

The constructor in the superclass is responsible for building the object of the superclass and the constructor of the subclass builds the object of subclass. When the subclass constructor is called during object creation, it by default invokes the default constructor of super-class. Hence, in inheritance the objects are constructed top-down. The superclass constructor can be called explicitly using the keyword super, but it should be first statement in a constructor. The keyword super always refers to the superclass immediately above of the calling class in the hierarchy. The use of multiple super keywords to access an ancestor class other than the direct parent is illegal.

```
class Shape {
   private int length;
   private int breadth;
   public int getBreadth() {
      return breadth;
   }
   public int getLength() {
      return length;
   }
   public void setBreadth(int i) {
      breadth = i;
   }
   public void setLength(int i) {
      length = i;
   }
   // default Constructor
   Shape() {
      length = 0;
      breadth = 0;
      System.out.println("Inside default constructor of Shape ");
   }

   // Parameterized Constructor
   Shape(int len, int bdth) {
       length = len;
```

```
        breadth = bdth;
        System.out.println("Inside constructor of Shape ");
        System.out.println("length : " + length);
        System.out.println("breadth : " + breadth);
    }
}

// A subclass which extends for shape
class Rectangle extends Shape {
   private String type;

   // default Constructor
   Rectangle() {
      super();
      type = null;
      System.out.println("Inside default constructor of rectangle ");
   }

    // Parameterized Constructor
    Rectangle(String ty, int len, int bdth) {
       super (len, bdth);
       System.out.println("Inside constructor of rectangle ");
       System.out.println("length : " + len);
       System.out.println("breadth : " + bdth);
       System.out.println("type : " + type);
    }

    public String getType() {
       return type;
    }

    public void setType(String string) {
       type = string;
    }
}

// A subclass which extends for rectangle
class ColoredRectangle extends Rectangle {
    private String color;
    /* default Constructor*/
    ColoredRectangle() {
       super();
       color = null;
       System.out.println("Inside default constructor of coloredRectangle");
    }

     // Parameterized Constructor
     ColoredRectangle(String c, String ty, int len, int bdth) {
       super (ty, len, bdth);
       System.out.println("Inside constructor of coloredRectangle ");
       System.out.println("length : " + len);
       System.out.println("breadth : " + bdth);
       System.out.println("type : " + ty);
     }
     public String getColor() {
         return color;
     }
     public void setColor(String string) {
        color = string;
     }
}

public class Test {
   public static void main(String args[]) {
       ColoredRectangle CR = new ColoredRectangle();
       ColoredRectangle CR2 = new ColoredRectangle("Red","Big", 5, 2 );
   }
}
```

The output is:

```
Inside default constructor of Shape
Inside default constructor of rectangle
Inside default constructor of coloredRectangle
Inside constructor of Shape
length : 5
breadth : 2
Inside constructor of rectangle
length : 5
breadth : 2
type : null
Inside constructor of coloredRectangle
length : 5
breadth : 2
type : Big
```

Inheritance and Method Overriding

By using super we can access the overridden method in the super class.

```
class Shape {
   private int length;
   private int breadth;
   // default Constructor
   Shape() {
      length = 0;
      breadth = 0;
   }
   // Parameterized Constructor
   Shape(int len, int bdth) {
      length = len;
      breadth = bdth;
   }
   void showattributes() {
      System.out.println("length : " + length);
      System.out.println("breadth : " + breadth);
   }
}

// A subclass which extends for shape
class Rectangle extends Shape {
   private String type;
   /* default Constructor
   */
   Rectangle() {
      type = null;
   }
   // Parameterized Constructor
   Rectangle(String ty, int len, int bdth) {
     super(len,bdth);
     type = ty;
   }
   void showattributes() {
      // showattributes() of class Shape is called
      super.showattributes();
      System.out.println("type : " + type);
   }
}

public class Test {
   public static void main(String args[]) {
       Rectangle rect = new Rectangle("Blue",5,7);
       // showattributes() in rectangle is called
       rect.showattributes();
   }
}
```

The output is :

```
length : 5
breadth : 7
type : Blue
```

Inheritance & Abstract Classes

The superclasses are more general than their subclasses. Usually, the superclasses are made abstract so that the objects of its prototype cannot be made. So the objects of only the subclasses can be used. To make a class abstract, the keyword abstract is used in the class definition.

Abstract methods are methods which do not have method statements. The subclasse provides the method statements. The methods provided by the superclass needs to be overridden by the subclass. The class that has at least one abstract method should be made abstract. The abstract class cannot be instantiated because it does not define a complete implementation.

```
public abstract class {
   ….
}
```

Using Final with methods:

We can prevent a method from being overridden by using the keyword final at the start of its declaration. Final methods cannot be overridden.

```
public abstract void methodname();
class Shape {
   final void showattributes() {
      System.out.println("Inside class shape ");
   }
}

// A subclass which extends for shape
class Rectangle extends Shape {
    void showattributes() { // Cannot override the final method
        System.out.println("Inside class rectangle");
    }
}
```

The method `showattributes()` cannot be overridden in the class rectangle because it is declared as final in class shape. It shows an error when we try to override it.

Using Final with class:

We can also prevent inheritance by making a class final. When a class is declared as final, its methods also become final. An abstract class cannot be declared as final because an abstract class is incomplete and its subclasses need to provide the implementation.

```
final class shape {
   void showattributes() {
      System.out.println("Inside class shape ");
   }
}

/* A subclass which extends for shape
*/
class rectangle extends shape {
   // The type rectangle cannot subclass the final class shape
   void showattributes() {
     System.out.println("Inside class rectangle");
   }
}
```

The class shape cannot be inherited because it is declared as final. It will show an error when we try to inherit it.

Reference:

Java 2: the complete reference: fifth Edition
http://java.sun.com