Object Oriented Programming

Abstract Classes, Inheritance via abstract classes

Compiled By: Umair Yaqub

Abstract Classes in Java: A class that is declared with abstract keyword is known as abstract class in java. It can have abstract and non-abstract methods (method with body). Before learning java abstract class, let's understand the abstraction in java first.

Abstraction in Java: Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

Ways to achieve Abstraction: There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

- **Abstract class:** A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

  e.g:- **abstract** class A{}

- **Abstract method:** A method that is declared as abstract and does not have implementation is known as abstract method.

  e.g:- **abstract** void printStatus(); //no body and abstract

Example 1: In this example, Bike the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

```
abstract class Bike{
        abstract void run();
    }
```

```
class Honda4 extends Bike{
                    void run()
                    {
                    System.out.println("running safely..");
                    }
public static void main(String args[]){
        Bike obj = new Honda4();
        obj.run();
        }
}
```

Example 2: In this example, Shape is the abstract class, its implementation is provided by the Rectangle and Circle classes. Mostly, we don't know about the implementation class (i.e. hidden to the end user) and object of the implementation class is provided by the **factory method**.

```
abstract class Shape{
                    abstract void draw();
                    }
class Rectangle extends Shape{
                    void draw()
                    {
                    System.out.println("drawing rectangle");
                    }
}
class Circle1 extends Shape{
                    void draw()
                    {
                    System.out.println("drawing circle");
                    }
}
class TestAbstraction1{
                    public static void main(String args[]){
                    Shape s=new Circle1();
                    s.draw();
        }
}
```

Example 3:

```
abstract class Bank{
                    abstract int getRateOfInterest();
                }
class SBI extends Bank{
                    int getRateOfInterest()
                    {
                    return 7;
                    }
}
class PNB extends Bank{
                    int getRateOfInterest()
                    {
                    return 8;
                    }
}

class TestBank{
                public static void main(String args[]){
                Bank b;
                b=new SBI();
                System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
                b=new PNB();
                System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
        }

}
```

Interface in Java: An interface in java is a blueprint of a class. It has static constants and abstract methods. The interface in java is **a mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java. **Java Interface also** represents IS-A relationship**.**

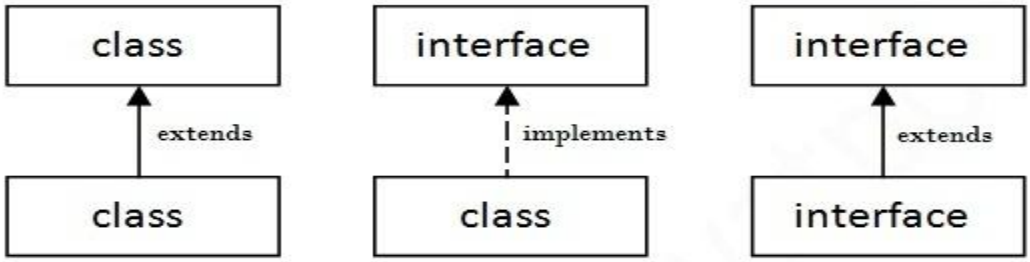
It cannot be instantiated just like abstract class.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

As shown in the figure given below, a class extends another class, an interface extends another interface but a class implements an interface.



Example: In this example, Printable interface has only one method, its implementation is provided in the A class.

```
interface printable{
                    void print();
                    }
class A6 implements printable{
                    public void print()
                    {
                    System.out.println("Hello");
                    }
        public static void main(String args[]){
        A6 obj = new A6();
        obj.print();
        }
}
```

Example: Drawable

In this example, Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In real scenario, interface is defined by someone but implementation is provided by different implementation providers. And, it is used by someone else. The implementation part is hidden by the user which uses the interface.

```
interface Drawable{
                    void draw();
                    }
class Rectangle implements Drawable{
                    public void draw()
                    {
                    System.out.println("drawing rectangle");
                    }
}
class Circle implements Drawable{
                    public void draw()
                    {
                    System.out.println("drawing circle");
                    }
}
class TestInterface1{
                    public static void main(String args[]){
                    Drawable d=new Circle();
                    d.draw();
                }
}
```

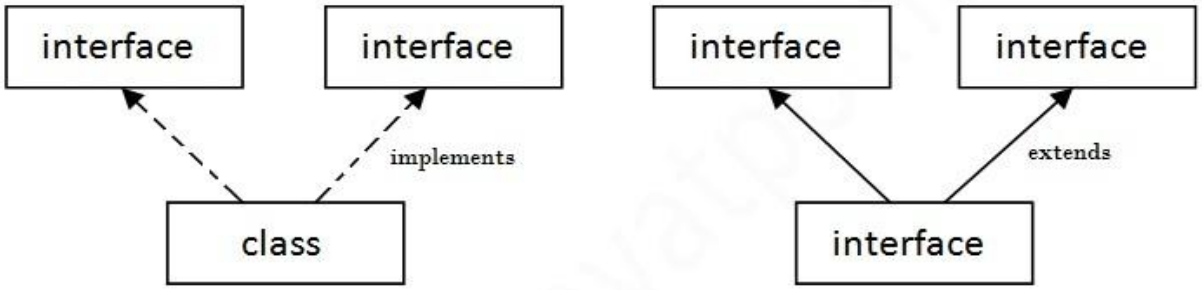
Example: Bank

Let's see another example of java interface which provides the implementation of Bank interface.

```
interface Bank{
                float rateOfInterest();
                }
class SBI implements Bank{
                public float rateOfInterest()
                {
                return 9.15f;
                }
}
```

```
class PNB implements Bank{
                        public float rateOfInterest()
                        {
                        return 9.7f;
                        }
}
class TestInterface2{
                    public static void main(String[] args){
                    Bank b=new SBI();
                    System.out.println("ROI: "+b.rateOfInterest());
        }
}
```

Multiple inheritance in Java by interface: If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.



Multiple Inheritance in Java

```
interface Printable{
                    void print();
}
interface Showable{
                void show();
}
class A7 implements Printable,Showable{
                public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}
                public static void main(String args[]){
                A7 obj = new A7();
                obj.print();
                obj.show();
        }
}
```