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.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also **represents the IS-A relationship**.

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.

How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

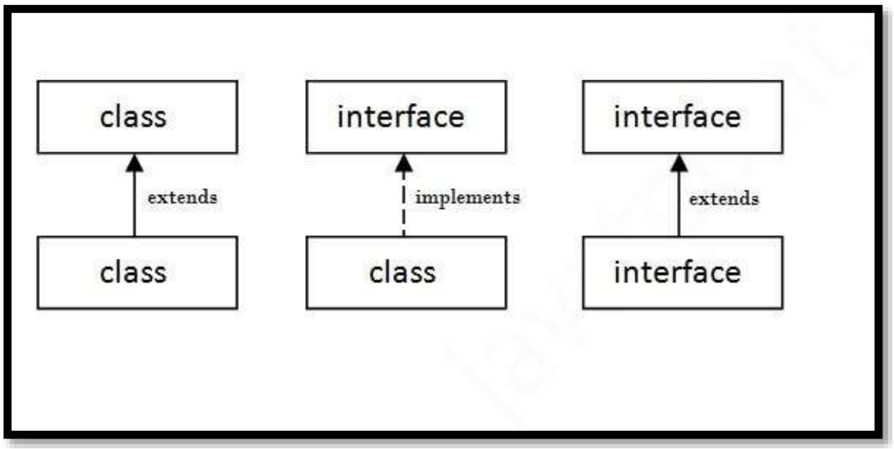
Syntax:

```
interface <interface_name>

{

   // declare constant fields

   // declare methods that abstract

   // by default.

}
```

The relationship between classes and interfaces

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

The relationship between class and interface



Java Interface Example

In this example, the Printable interface has only one method, and its implementation is provided in the A6 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();
 }
}
```

Java Interface Example: Drawable

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

Save File: TestInterface1.java

```
//Interface declaration: by first user
interface Drawable
{
void draw();
}
//Implementation: by second user
class Rectangle implements Drawable
{
public void draw()
{
System.out.println("drawing rectangle");
}
}
class Circle implements Drawable
```

```
{
public void draw()
{
System.out.println("drawing circle");
}
}
//Using interface: by third user
class TestInterface1
{
public static void main(String args[])
{
Drawable d=new Circle();//In real scenario, object is provided by method e.g.
//getDrawable()
d.draw();
}}
```

Java Interface Example: Bank

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

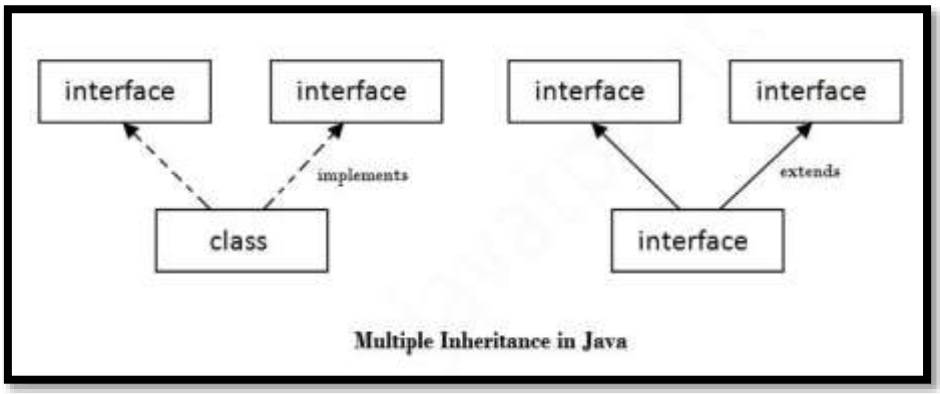
File: TestInterface2.java

```
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, it is known as multiple inheritance.



Example –

```
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();

 }

}
```