Interfaces in Java

An Interface is defined as an abstract type used to specify the behavior of a class. An interface in Java is a blueprint of a class. A Java interface contains **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 the 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. Java Interface also represents the **IS-A relationship.**
- Like a class, an interface can have methods and variables, but the methods declared in an interface are **by default abstract**.

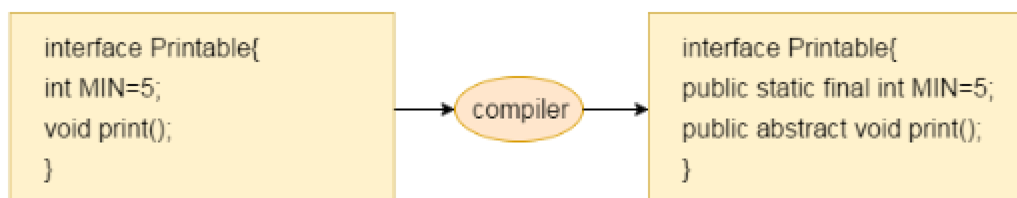
Notes:

- It cannot be **instantiated** just like the abstract class.
- Since Java 8, we can have **default** and **static** methods in an interface.
- Since Java 9, we can have **private methods** in an interface.

Why do we use an Interface?

- It is used to achieve total **abstraction**.
- Since java does not support **multiple inheritance** in the case of class, by using an interface it can achieve multiple inheritance.
- Interfaces are used to implement abstraction. So the question arises why use interfaces when we have abstract classes?

A// the reason is, abstract classes may contain non-final variables, whereas variables in the interface are **final, public and static**.



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**.

```
// A simple interface
interface Player
{
   final int id = 10;
   int move(); //abstract method
}
```

Difference between Class and Interface

The major differences between a class and an interface are:

Class	Interface
In class, you can instantiate variables and create an object.	In an interface, you can't instantiate variables and create an object.
Class can contain concrete(with implementation) methods	The interface cannot contain concrete(with implementation) methods
The access specifiers used with classes are private, protected, and public.	In Interface only one specifier is used- Public.

البرمجة الشبابية

Implementation: To implement an interface we use the keyword **implements**

Example:

```
interface In1 {

  final int a = 10;       // public, static and final
  void display();    // public and abstract

}

class TestClass implements In1 { // A class that implements the interface.
  public void display(){
   System.out.println("Test");
  }

  public static void main(String[] args)
  {
    TestClass t = new TestClass();
    t.display();
    System.out.println(a);
  }
}
```

Output
Test
10

Real-World Example: Let's consider the example of vehicles like bicycle, car, and bike………, they have **common functionalities**. So we make an interface and put all these common functionalities. And lets Bicycle, Bike, car ….etc. implement all these functionalities in their own class in their own way.

```
interface Vehicle {

  void changeGear(int a);
  void speedUp(int a);        // all are the abstract methods.
  void applyBrakes(int a);
}
```

```
class Bicycle implements Vehicle{
    int speed;
    int gear;
    @Override
    public void changeGear(int newGear){  // to change gear
                gear = newGear;
    }
   @Override
    public void speedUp(int increment){
                speed = speed + increment;
    }
    @Override
    public void applyBrakes(int decrement){
                speed = speed - decrement;
    }
    public void printStates() {
        System.out.println("speed: " + speed
            + " gear: " + gear);
    }
}

class Bike implements Vehicle {
    int speed;
    int gear;
    @Override
    public void changeGear(int newGear){
        gear = newGear;
    }
    @Override
    public void speedUp(int increment){
        speed = speed + increment;
    }
    @Override
    public void applyBrakes(int decrement){
        speed = speed - decrement;
```

البرمجة الكيانية

```
    }

    public void printStates() {
        System.out.println("speed: " + speed
            + " gear: " + gear);
    }

}
Class Test {

    public static void main (String[] args) {

        Bicycle bicycle = new Bicycle();
        bicycle.changeGear(2);
        bicycle.speedUp(3);
        bicycle.applyBrakes(1);
        System.out.println("Bicycle present state :");
        bicycle.printStates();

        Bike bike = new Bike();
        bike.changeGear(1);
        bike.speedUp(4);
        bike.applyBrakes(3);

        System.out.println("Bike present state :");
        bike.printStates();
    }
}
```

Output

```
Bicycle present state :
speed: 2 gear: 2
Bike present state :
speed: 1 gear: 1
```

Advantages of Interfaces in Java

The advantages of using interfaces in Java are as follows:

- Without bothering about the implementation part, we can achieve the **security of the implementation**.
- In Java, multiple inheritance is not allowed, however, you can use an interface to make use of it as you can implement more than one interface.

New Features Added in Interfaces in JDK 8

1. Prior to JDK 8, the interface could not define the implementation. We can now add **default implementation for interface methods**. **This default implementation has a special use and does not affect the intention behind interfaces.**

Suppose we need to add a new function in an existing interface. **Obviously**, the old code will not work as the classes have not implemented those new functions. So with the help of default implementation, we will give a default body for the newly added functions. Then the old codes will still work.

```
interface In1
{
   final int a = 10;
   default void display()
   {
      System.out.println("hello");
   }
}

class TestClass implements In1// A class that implements the interface.
{
   // Driver Code
   public static void main (String[] args)
   {
      TestClass t = new TestClass();
      t.display();
   }
}
```

Output
hello

2. Another feature that was added in **JDK 8** is that we can now define static methods in interfaces that can be called independently without an object.

Note: these methods are not inherited.

```
interface In1
{
   final int a = 10;
```

```
    static void display()
    {
        System.out.println("hello");
    }
}

class TestClass implements In1// A class that implements the interface.
{
    public static void main (String[] args)
    {
        In1.display();
    }
}
```

Output
hello

Important Points about Interface or Summary of the Article:

- We **can't** create an instance (interface can't be instantiated) of the interface but we can make the **reference** of it that refers to the Object of its implementing class.
- A class can implement **more than one** interface.
- An interface can extend to another interface or interfaces (more than one interface).
- A class that implements the interface must implement all the methods in the interface.
- All the methods are public and abstract. And all the fields are public, static, and final.
- It is used to achieve multiple inheritances.

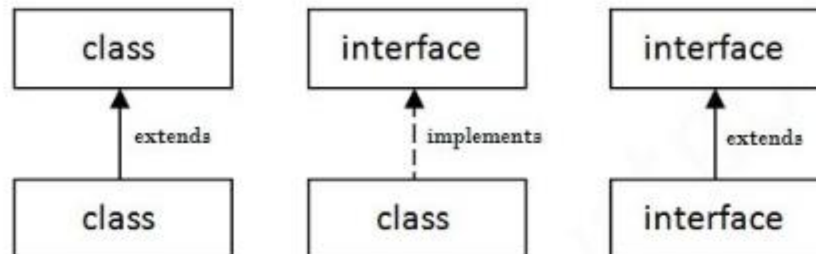
New Features Added in Interfaces in JDK 9

From Java 9 onwards, interfaces can contain the following also:

- Static methods
- Private methods
- Private Static methods

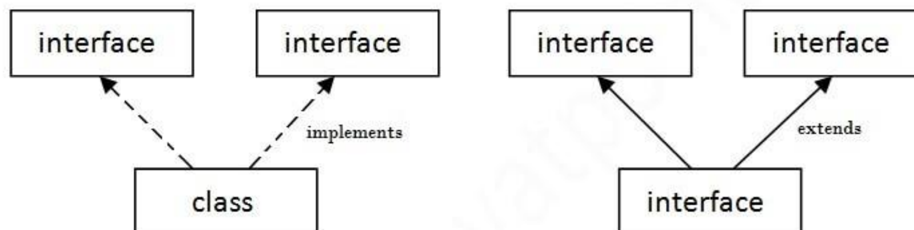
The relationship between classes and interfaces

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



Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

```
interface Printable{
void print();
}
interface Showable{
void show();
}
class Test implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}
```

```
public static void main(String args[]){
Test obj = new Test ();
obj.print();
obj.show();
 }
}
```

```
interface Printable{

void print();

}

interface Showable{
void print();
}
```

هنا نلاحظ حل لمشكلة الوراثة المتعدد المتعلقة بوجود نفس الدالة في اكثر من كلاس.

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

Interface inheritance

A class implements an interface, but one interface extends another interface.

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class Test implements Showable{
public void print(){System.out.println("Hello");}
```

```
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
Test obj = new Test ();
obj.print();
obj.show();
 }
}
```

البرمجة الكيانية