Interfaces in Java

Difficulty Level : Easy Last Updated : 17 Feb, 2021

Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

- Interfaces specify what a class must do and not how. It is the blueprint of the class.
- An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
- If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.
- A Java library example is, Comparator Interface. If a class implements this interface, then it can be used to sort a collection.

Syntax :

```
interface <interface_name> {

    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

To declare an interface, use **interface** keyword. It is used to provide total abstraction. That means all the methods in an interface are declared with an empty body and are public and all fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface. To implement interface use **implements** keyword.

Why do we use interface ?

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

The reason is, abstract classes may contain non-final variables, whereas variables in interface are final, public and static.

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

To implement an interface we use keyword: implements

```
// Java program to demonstrate working of
// interface.
import java.io.*;

// A simple interface
interface In1
{
    // public, static and final
    final int a = 10;

    // public and abstract
    void display();
}

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

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

Output:

```
Geek
10
```

A real-world example:

Let's consider the example of vehicles like bicycle, car, 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.

```
import java.io.*;

interface Vehicle {

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

class Bicycle implements Vehicle{

    int speed;
    int gear;

     // to change gear
    @Override
    public void changeGear(int newGear){

        gear = newGear;
    }

    // to increase speed
    @Override
    public void speedUp(int increment){

        speed = speed + increment;
    }

    // to decrease speed
    @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;

    // to change gear
    @Override
    public void changeGear(int newGear){

        gear = newGear;
    }

    // to increase speed
    @Override
    public void speedUp(int increment){

        speed = speed + increment;
    }

    // to decrease speed
    @Override
    public void applyBrakes(int decrement){

        speed = speed - decrement;
    }

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

}
class GFG {

    public static void main (String[] args) {

        // creating an inatance of Bicycle
        // doing some operations
        Bicycle bicycle = new Bicycle();
        bicycle.changeGear(2);
        bicycle.speedUp(3);
        bicycle.applyBrakes(1);

        System.out.println("Bicycle present state :");
        bicycle.printStates();
```

```
        // creating instance of the bike.
        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
```

New features added in interfaces in JDK 8

1. Prior to JDK 8, interface could not define implementation. We can now add default implementation for interface methods. This default implementation has 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.

```
// An example to show that interfaces can
// have methods from JDK 1.8 onwards
interface In1
{
    final int a = 10;
    default void display()
    {
        System.out.println("hello");
    }
}

// A class that implements the interface.
class TestClass implements In1
{
```

```
    // 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 which can be called independently without an object. Note: these methods are not inherited.

```
// An example to show that interfaces can
// have methods from JDK 1.8 onwards
interface In1
{
    final int a = 10;
    static void display()
    {
        System.out.println("hello");
    }
}

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

Output :

```
hello
```

Important points about interface or summary of article:

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

New features added in interfaces in JDK 9

From Java 9 onwards, interfaces can contain following also

1. Static methods
2. Private methods
3. Private Static methods

Related articles:

- Access specifier of methods in interfaces
- Access specifiers for classes or interfaces in Java
- Abstract Classes in Java
- Comparator Interface in Java
- Java Interface methods
- Nested Interface in Java

This article is contributed by **Mehak Kumar.** and **Nitsdheerendra**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above