Interface in java with example programs

BY CHAITANYA SINGH | FILED UNDER: OOPS CONCEPT

In the last tutorial we discussed abstract class which is used for achieving partial abstraction. Unlike abstract class an interface is used for full abstraction. Abstraction is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user(See: Abstraction). In this guide, we will cover **what is an interface in java**, why we use it and what are rules that we must follow while using interfaces in Java Programming.

What is an interface in Java?

Interface looks like a class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body, see: Java abstract method). Also, the variables declared in an interface are public, static & final by default. We will cover this in detail, later in this guide.

What is the use of interface in Java?

As mentioned above they are used for full abstraction. Since methods in interfaces do not have body, they have to be implemented by the class before you can access them. The class that implements interface must implement all the methods of that interface. Also, java programming language does not allow you to extend more than one class, However you can implement more than one interfaces in your class.

Syntax:
Interfaces are declared by specifying a keyword “interface”. E.g.:

```
interface MyInterface
{
   /* All the methods are public abstract by default
    * As you see they have no body
    */
   public void method1();
   public void method2();
}
```

Example of an Interface in Java

This is how a class implements an interface. It has to provide the body of all the methods that are declared in interface or in other words you can say that class has to implement all the methods of interface.

Do you know? class `implements` interface but an interface `extends` another interface.

```
interface MyInterface
{
   /* compiler will treat them as:
    * public abstract void method1();
    * public abstract void method2();
    */
   public void method1();
   public void method2();
}
class Demo implements MyInterface
{
   /* This class must have to implement both the abstract methods
    * else you will get compilation error
    */
   public void method1()
   {
        System.out.println("implementation of method1");
   }
   public void method2()
   {
        System.out.println("implementation of method2");
   }
   public static void main(String arg[])
   {
        MyInterface obj = new Demo();
        obj.method1();
   }
}
```

Output:

```
implementation of method1
```

You may also like to read: Difference between abstract class and interface

Interface and Inheritance

As discussed above, an interface can not implement another interface. It has to extend the other interface. See the below example where we have two interfaces Inf1 and Inf2. Inf2 extends Inf1 so If class implements the Inf2 it has to provide implementation of all the methods of interfaces Inf2 as well as Inf1.

Learn more about inheritance here: Java Inheritance

```
interface Inf1{
   public void method1();
}
interface Inf2 extends Inf1 {
   public void method2();
}
public class Demo implements Inf2{
```

```
   /* Even though this class is only implementing the
    * interface Inf2, it has to implement all the methods
    * of Inf1 as well because the interface Inf2 extends Inf1
    */
    public void method1(){
        System.out.println("method1");
    }
    public void method2(){
        System.out.println("method2");
    }
    public static void main(String args[]){
        Inf2 obj = new Demo();
        obj.method2();
    }
}
```

In this program, the class `Demo` only implements interface `Inf2`, however it has to provide the implementation of all the methods of interface `Inf1` as well, because interface Inf2 extends Inf1.

Tag or Marker interface in Java

An empty interface is known as tag or marker interface. For example Serializable, EventListener, Remote(java.rmi.Remote) are tag interfaces. These interfaces do not have any field and methods in it. Read more about it here.

Nested interfaces

An interface which is declared inside another interface or class is called nested interface. They are also known as inner interface. For example Entry interface in collections framework is declared inside Map interface, that's why we don' use it directly, rather we use it like this: `Map.Entry`.

Key points: Here are the key points to remember about interfaces:
1) We can't instantiate an interface in java. That means we cannot create the object of an interface

2) Interface provides full abstraction as none of its methods have body. On the other hand abstract class provides partial abstraction as it can have abstract and concrete(methods with body) methods both.

3) `implements` keyword is used by classes to implement an interface.

4) While providing implementation in class of any method of an interface, it needs to be mentioned as public.

5) Class that implements any interface must implement all the methods of that interface, else the class should be declared abstract.

6) Interface cannot be declared as private, protected or transient.

7) All the interface methods are by default **abstract and public**.

8) Variables declared in interface are **public, static and final** by default.

```
interface Try
{
   int a=10;
   public int a=10;
   public static final int a=10;
   final int a=10;
   static int a=0;
}
```

All of the above statements are identical.

9) Interface variables must be initialized at the time of declaration otherwise compiler will throw an error.

```
interface Try
{
      int x;//Compile-time error
}
```

Above code will throw a compile time error as the value of the variable x is not initialized at the time of declaration.

10) Inside any implementation class, you cannot change the variables declared in interface because by default, they are public, static and final. Here we are implementing the interface “Try” which has a variable x. When we tried to set the value for variable x we got compilation error as the variable x is public static **final** by default and final variables can not be re-initialized.

```
class Sample implements Try
{
  public static void main(String args[])
  {
     x=20; //compile time error
  }
}
```

11) An interface can extend any interface but cannot implement it. Class implements interface and interface extends interface.

12) A **class** can implement any **number of interfaces**.

13) If there are **two or more same methods** in two interfaces and a class implements both interfaces, implementation of the method once is enough.

```
interface A
{
   public void aaa();
}
interface B
{
   public void aaa();
}
class Central implements A,B
{
   public void aaa()
   {
        //Any Code here
   }
   public static void main(String args[])
   {
        //Statements
    }
}
```

14) A class cannot implement two interfaces that have methods with same name but different return type.

```
interface A
{
   public void aaa();
}
interface B
{
   public int aaa();
}

class Central implements A,B
{

   public void aaa() // error
   {
   }
   public int aaa() // error
   {
   }
   public static void main(String args[])
   {

   }
}
```

15) Variable names conflicts can be resolved by interface name.

```
interface A
{
    int x=10;
}
interface B
{
```

```
    int x=100;
}
class Hello implements A,B
{
    public static void Main(String args[])
    {
       /* reference to x is ambiguous both variables are x
        * so we are using interface name to resolve the
        * variable
        */
       System.out.println(x);
       System.out.println(A.x);
       System.out.println(B.x);
    }
}
```

Advantages of interface in java:

Advantages of using interfaces are as follows:

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