Interfaces
in JAVA

Muzaffar Iqbal Farooqi

Interfaces

- Traditionally interface is like an abstract class where …
  - There are no instance variables
  - There are no constructors
  - All the methods are abstract
  - It cannot be instantiated just like the abstract class.

Interfaces

- It is a blueprint of a class.
- It is used to achieve abstraction and multiple inheritance in Java.
- The syntax for defining an interface is similar to that of defining a class except the word **`interface`** is used in place of **`class`**

Example.java

```
public interface Example
{
    public void abc();
    public void xyz(int x, double y);
}
```

Interfaces

- An interface serves a function similar to a base class, though it is not a base class
  - Some languages allow one class to be derived from two or more different base classes
  - This *multiple inheritance* is not allowed in Java
  - Instead, Java's way of approximating multiple inheritance is through interfaces

Interfaces

- An interface and all of its method headings should be declared public
  - They cannot be given private, protected, or default
- When a class implements an interface, it must make all the methods in the interface public
- If a class implements an interface and do not define all the methods of the interface then the class becomes abstract and must be declared as abstract class.

Interfaces…

- To *implement an interface*, a class must do two things:
  1. It must include the phrase

     `implements` ***`Interface_Name`***

     at the start of the class definition
     - If more than one interface is implemented, each is listed, separated by commas
  2. The class must implement *all* the method headings listed in the definition(s) of the interface(s)

Interface Example

Math.java | Calculator.java

```
public interface Math
{
    public int add();
    public int subtract();
    public int multiply();
    public double divide();
}
```

Math.java | **Calculator.java**

```
public class Calculator implements Math
{   int a;
    int b;
    public Calculator()
    {
    }
    public Calculator(int a, int b)
    {
        this.a=a;
        this.b=b;
    }
    public int add()
    {
        return a+b;
    }
    public int subtract()
    {
        return a-b;
    }
    public int multiply()
    {
        return a*b;
    }
    public double divide()
    {
        return (double)a/b;
    }
}
```

Implementation

Math.java | Calculator.java | **Runner.java**

```
public class Runner
{
    public static void main(String args[])
    {
        int a=9;
        int b=2;
        Calculator c=new Calculator(a,b);
        System.out.println(a+" + " + b + " = "+c.add());
        System.out.println(a+" - " + b + " = "+c.subtract());
        System.out.println(a+" * " + b + " = "+c.multiply());
        System.out.println(a+" / " + b + " = "+c.divide());
    }
}
```

General Output

```
--------------------Configuration: <Default>--------------------
9 + 2 = 11
9 - 2 = 7
9 * 2 = 18
9 / 2 = 4.5

Process completed.
```

Derived Interfaces

- Like classes, an interface may be derived from a base interface
  - This is called *extending* the interface
  - The derived interface must include the phrase

    ```
    extends BaseInterfaceName
    ```
- A concrete class that implements a derived interface must have definitions for any methods in the derived interface as well as any methods in the base interface

Derived Interface

```
public interface Math1
{
    public int add();
    public int subtract();
}
```

```
public interface Math2 extends Math1
{
    public int multiply();
    public double divide();
}
```

```
public class Calculator implements Math2
{   int a;
    int b;
    public Calculator()
    {
    }
    public Calculator(int a, int b)
    {
        this.a=a;
        this.b=b;
    }
    public int add()
    {
        return a+b;
    }
    public int subtract()
    {
        return a-b;
    }
    public int multiply()
    {
        return a*b;
    }
    public double divide()
    {
        return (double)a/b;
    }
}
```

Implementing Multiple Interfaces

- A class may implement multiple interfaces at the same time.
- In this case, the class must define all the methods of the interfaces being implemented.
- If the class fails to define any method then the class becomes abstract.

Implementing Multiple Interfaces

```
public interface Math1
{
    public int add();
    public int subtract();
}
```

```
public interface Math2
{
    public int multiply();
    public double divide();
}
```

```
public class Calculator implements Math1, Math2
{    int a;
     int b;
     public Calculator()
     {
     }
     public Calculator(int a, int b)
     {
          this.a=a;
          this.b=b;
     }
     public int add()
     {
          return a+b;
     }
     public int subtract()
     {
          return a-b;
     }
     public int multiply()
     {
          return a*b;
     }
     public double divide()
     {
          return (double)a/b;
     }
}
```

Defining Constants in Interfaces

- An interface can contain defined constants in addition to or instead of method headings
  - Any variables defined in an interface must be public, static and final
  - Because this is understood, Java allows these modifiers to be omitted
- Any class that implements the interface has access to these defined constants

Defining Constants in Interfaces

```
public interface MyInterface
{
    public static double PI=(double) 22/7;
    public double areaOfCircle(double radius);
}
```

```
public class MyCircle implements MyInterface
{

    public MyCircle()
    {
    }
    public double areaOfCircle(double radius)
    {
        return MyInterface.PI * radius * radius;
    }
}
```

Java 8 Improvement

- Since Java 8, a drastic change has been made in interfaces.
- Since Java 8, interface can have complete methods including body but must be declared as static.
- It was not possible in interfaces before Java 8.

Static Methods in Interfaces

```
package interfaces;
public interface MyInterface
{
    public static double PI=(double) 22/7;
    public double areaOfCircle(double radius);
    public static void testMethod()
    {
        System.out.println("THis is a test method defined in interface...");
    }
}
```

```
package interfaces;
public class Test
{
    public static void main(String args[])
    {
        System.out.println("Value of PI is "+MyInterface.PI);
        MyInterface.testMethod();
    }
}
```

General Output

```
--------------------Configuration: <Default>--------------------
Value of PI is 3.142857142857143
THis is a test method defined in interface...

Process completed.
```

Pitfall: Inconsistent Interfaces

- When a class implements two interfaces:
  - One type of inconsistency will occur if the interfaces have constants with the same name, but with different values
  - Another type of inconsistency will occur if the interfaces contain methods with the same name but different return types
- If a class definition implements two inconsistent interfaces, then that is an error, and the class definition is illegal

Summary - Class Vs. Interface

- An interface is different from a class in several ways, including…
  - You cannot instantiate an interface.
  - An interface does not contain any constructors.
  - All of the non-static methods in an interface are abstract.
  - An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
  - An interface is not extended by a class; it is implemented by a class.
  - An interface can extend multiple interfaces.

QUESTIONS

?