Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Java : Implementing Interfaces/5b1e4d1a7becc0459deade7e

TUTORIAL

Java : Implementing Interfaces

Topics

Implementing Interfaces

Once an interface has been defined, one or more classes can implement that interface. To implement an interface, include the implements clause in a class definition, and then create the methods defined by the interface. The general form of a class that includes the implements clause looks like this:

```
access class classname [extends superclass]
[implements interface [,interface...]]
{
        // class-body
}
```

Here, access is either public or not used. If a class implements more than one interface, the interfaces are separated with a comma. If a class implements two interfaces that declare the same method, then the same method will be used by clients of either interface. The methods that implement an interface must be declared public. Also, the type signature of the implementing method must match exactly the type signature specified in the interface definition.

Here is a small example class that implements the Callback interface shown earlier.

```
class Client implements Callback
{
        // Implement Callback's interface
        public void callback(int p)
        {
                System.out.println("callback called with " + p);
        }
```

}

Notice that callback( ) is declared using the public access specifier. When you implement an interface method, it must be declared as public. It is both permissible and common for classes that implement interfaces to define additional members of their own. For example, the following version of Client implements callback( ) and adds the method nonIfaceMeth( ):

```
class Client implements Callback
{
        // Implement Callback's interface
        public void callback(int p)
        {
                System.out.println("callback called with " + p);
        }
        void nonIfaceMeth()
        {
                System.out.println("Classes that implement
interfaces may also define other members, too.");
        }
}
```

Variables in Interfaces

We can use interfaces to import shared constants into multiple classes by simply declaring an interface that contains variables which are initialized to the desired values. When you include that interface in a class (that is, when you "implement" the interface), all of those variable names will be in scope as constants. Following code shows the use of variables in interfaces:

Java

```java
import java.util.Random;
interface SharedConstants
```

```
{
  int NO = 0;     int YES = 1;    int MAYBE = 2;
  int LATER = 3;  int SOON = 4;   int NEVER = 5;
}

class Question implements SharedConstants
{
  Random rand = new Random();
  int ask()
  {
    int prob = (int) (100 * rand.nextDouble());
    if (prob < 30)
      return NO; // 30%
    else if (prob < 60)
      return YES; // 30%
    else if (prob < 75)
      return LATER; // 15%
    else if (prob < 98)
      return SOON; // 13%
    else
      return NEVER; // 2%
  }
}

class Main implements SharedConstants
{
  static void answer(int result)
  {
    switch(result)
    {
      case NO:
        System.out.println("No");
        break;
      case YES:
        System.out.println("Yes");
        break;
      case MAYBE:
        System.out.println("Maybe");
        break;
      case LATER:
        System.out.println("Later");
        break;
      case SOON:
        System.out.println("Soon");
        break;
      case NEVER:
        System.out.println("Never");
        break;
    }
  }

  public static void main(String args[])
  {
    Question q = new Question();
    answer(q.ask());
    answer(q.ask());
    answer(q.ask());
    answer(q.ask());
  }
}

```

Notice that this program makes use of one of Java's standard classes: Random. This class provides pseudorandom numbers. It contains several methods which allow you to obtain random numbers in the form required by your program. In this example, the method nextDouble( ) is used. It returns random numbers in the range 0.0 to 1.0. In this sample program, the two classes, Question and Main, both implement the SharedConstants interface where NO, YES, MAYBE, SOON, LATER, and NEVER are defined. Inside each class, the code refers to these constants as if each class had defined or inherited them directly.

Extending Interfaces

One interface can inherit another by use of the keyword extends. The syntax is the same as for inheriting classes. When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance chain. Following is an example:

Java

```java
// One interface can extend another.
interface A {
  void meth1();
  void meth2();
}

// B now includes meth1() and meth2() -- it adds meth3().
interface B extends A {
  void meth3();
}

// This class must implement all of A and B
class MyClass implements B {
  public void meth1() {
    System.out.println("Implement meth1().");
  }
  public void meth2() {
    System.out.println("Implement meth2().");
  }
  public void meth3() {
    System.out.println("Implement meth3().");
  }
}

class Main
{
  public static void main(String arg[])
  {
    MyClass ob = new MyClass();
    ob.meth1();
    ob.meth2();
    ob.meth3();
  }
}

```

As an experiment you might want to try removing the implementation for meth1( ) in MyClass. This will cause a compile-time error. As stated earlier, any class that implements an interface must implement all methods defined by that interface, including any that are inherited from other interfaces.

Important features of interfaces

- 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.
- 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.