OBJECT ORIENTED PROGRAMMING

Affefah Qureshi

Department of Computer Science

Iqra University, Islamabad Campus.

PARTIAL IMPLEMENTATIONS

- If a class includes an interface but does not fully implement the methods defined by that interface, then that class must be declared as abstract.

```
interface Callback {
   void callbackMethod1();
   void callbackMethod2();}

abstract class Incomplete implements Callback {
   int a, b;

   void show() {
      System.out.println(a + " " + b);
   }

   @Override
   public void callbackMethod1() {
      System.out.println("Callback method 1 implemented.");
   }}
```

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 we include that interface in a class (that is, when we "implement" the interface), all of those variable names will be in scope as constants.
- If an interface contains no methods, then any class that includes such an interface doesn't actually implement anything.

VARIABLES IN INTERFACES

```
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 AskMe 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());
  }
}
```

RANDOM VARIABLE CLASS

- Import the class java.util.Random.
- A random variable is an object that behaves like a random number generator in that it produces a pseudorandom number sequence. The distribution of the values produced depends on the class of random variable used.
- RandomVariable interface provides the single method nextDouble. Given an instance, say rv, of a class that implements the RandomVariable interface, repeated calls of the form
- rv.nextDouble ();

METHODS OF JAVA RANDOM CLASS

- some of the methods of java Random class.
- **nextBoolean():** This method returns next pseudorandom which is a boolean value from random number generator sequence.
- **nextDouble():** This method returns next pseudorandom which is double value between 0.0 and 1.0.
- **nextFloat():** This method returns next pseudorandom which is float value between 0.0 and 1.0.
- **nextInt():** This method returns next int value from random number generator sequence.
- **nextInt(int n):** This method return a pseudorandom which is int value between 0 and specified value from random number generator sequence.

EXAMPLE

```
import java.util.Random;
public class RandomNumberExample {
        public static void main(String[] args) {
                //initialize random number generator
                Random random = new Random();
                //generates boolean value
                System.out.println(random.nextBoolean());
                //generates double value
                System.out.println(random.nextDouble());
                //generates float value
                System.out.println(random.nextFloat());
                //generates int value
                System.out.println(random.nextInt());
                //generates int value within specific limit
                System.out.println(random.nextInt(20));   } }
```

Output:

```
false
0.30986869120562854
0.6210066
-1348425743
18
```

INTERFACES CAN BE EXTENDED

- 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.

INTERFACES CAN BE EXTENDED

```
// One interface can extend another.
interface A {
   void meth1();
   void meth2();
 }
interface B extends A {
   void meth3();
 }
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 IFExtend {
public static void main(String arg[]) {
        MyClass ob = new MyClass();
        ob.meth1();
        ob.meth2();
        ob.meth3();
   }  }
```

WHY DEFAULT METHODS?

- If a large number of classes were implementing this interface, we need to track all these classes and make changes to them. This is not only tedious but error-prone as well.
- To resolve this, Java introduced default methods. Default methods are inherited like ordinary methods.

EXAMPLE

Output:
The area of the rectangle is 30
I have 4 sides.
The area of the square is 25
I can get sides of a polygon.

```
interface Polygon {
  void getArea();
  // default method
  default void getSides() {
    System.out.println("I can get sides of a
polygon.");
  } }
class Rectangle implements Polygon {
  public void getArea() {
    int length = 6;
    int breadth = 5;
    int area = length * breadth;
    System.out.println("The area of the
rectangle is " + area);
    }
   // overrides the getSides()
    public void getSides() {
      System.out.println("I have 4 sides.");
    }
  }
  class Square implements Polygon {
    public void getArea() {
      int length = 5;
      int area = length * length;
    System.out.println("The area of the square
is " + area);
  }
}
class Main {
  public static void main(String[] args) {

    // create an object of Rectangle
    Rectangle r1 = new Rectangle();
    r1.getArea();
    r1.getSides();

    // create an object of Square
    Square s1 = new Square();
    s1.getArea();
    s1.getSides();
  }
}
```