Chapter 10 – Interfaces

Dr Kafi Rahman

Assistant Professor @CS

Truman State University

Using Interfaces

- We want to determine average of the area of a number of classes
  - However, each class has their own way of calculating the area
- We can create an interface and implement the interface for each class:

```
public interface Measurable {
      public double calcArea();
}
```

Using Interfaces (cont)

- For example, the Circle class can implement the Measurable interface in its definition

```
public class Circle implements Measurable {

        private double radius;

        public Circle(double r) {
                this.radius = r;
        }


        public double calcArea() {
                return Math.PI * Math.pow(this.radius, 2.0);
        }

}
```

Using Interfaces (cont)

- For example, the Rectangle class can implement the Measurable interface in its definition

```
public class Rectangle implements Measurable {

        private double width, length;

        public Rectangle(double w, double l) {
                this.width = w;
                this.length = l;
        }


        public double calcArea() {
                return this.width * this.length;
        }

}
```

Using Interfaces (cont)

- Therefore, determining the average of the area of the objects of these classes would be straightforward

```
public static double calcAverage(Measurable[] measArray) {
        double average = 0;
        if (measArray != null && measArray.length == 0)
            return average;

        for (Measurable myObj : measArray) {
            average += myObj.calcArea();
        }

        return average / measArray.length;
    }
```

Using Interfaces for Callbacks

- Limitations of Measurable interface:
    - Can add Measurable interface only to classes under your control
- Callback: a mechanism for specifying code that is executed at a later time
    - Problem: the responsibility of measuring lies with the added objects themselves.
    - Alternative: give the average method both the data to be averaged and a method of measuring.
- Create an interface:

```
public interface MeasureInterface {
      double measureCalc(Object obj);
}
```

Using Interfaces for Callbacks

- A specific callback is obtained by implementing the Measurer interface:

```
public class MeasureSquare implements MeasureInterface {

    public double measureCalc(Object obj) {
        CSquare squareObj = (CSquare) obj;
        double area =  squareObj.getWidth() * squareObj.getWidth();
        return area;
    }

}
```

- Must cast from Object to Square:
  - CSquare squareObj = (CSquare) obj;

Using Interfaces for Callbacks

- The code that makes the call to the callback receives an object of class that implements this interface:

- The average method simply makes a callback to the measure method whenever it needs to measure any object.

```
public static double calcAverage(Object[] objArray, MeasureInterface measure) {
            double average = 0;
            if (objArray.length == 0)
                  return average;

            for (Object myObj : objArray) {
                  average += measure.measureCalc(myObj);
            }

            return average / objArray.length;
      }
```

Using Interfaces for Callbacks

- To compute the average area of squares:
    - construct an object of the MeasureSquare class and pass it to the average method:
- The average method will ask the sqrMeasurer object to measure area of each square objects.

```
MeasureInterface sqrMeasurer = new MeasureSquare();

CSquare[] squares = { new CSquare(4),
                      new CSquare(3), new CSquare(6) };

double average = calcAverage(squares, sqrMeasurer);
System.out.printf("The average is: %.2f", average);
```

Using Interfaces for Callbacks

- The Main class (which holds the average method) is decoupled from the class whose objects it processes (Rectangle).
- We provide a small "helper" class AreaMeasurer, to process rectangles.

```
public static double calcAverage(Object[] objArray, MeasureInterface measure) {
            double average = 0;
            if (objArray.length == 0)
                  return average;

            for (Object myObj : objArray) {
                  average += measure.measureCalc(myObj);
            }

            return average / objArray.length;
      }
```

Measurer

- Let us review the demo

Self Check 10.17

- How can you use the average method of this section to find the average length of String objects?

- Answer: Implement a class StringMeasurer that implements the Measurer interface.

Self Check 10.19

- Write a method max with three arguments that finds the larger of any two objects, using a Measurer to compare them.
- Answer:

```
public static Object max(Object a, Object b, Measurer m)
{
    if (m.measureCalc(a) > m.measureCalc(b))
        {
            return a;
        }
    else { return b; }
}
```

Lambda Expressions

- Using a method such as average is a lot of work
    - Instead, we can use a lambda expression
- Works with interfaces that have a single abstract method
- Such interfaces are called functional interfaces…
    - …because instances are similar to mathematical functions

- Lambda expression specifies:
    - Parameters
    - Code for computing the returned value

Lambda Expressions

- Example of a lambda expression: A function that gets the area of a triangle object

```
(Object obj) -> {
                CSquare sqareObjct = (CSquare) obj;
                return sqareObjct.getWidth() * sqareObjct.getWidth();
        };
```

Lambda Expressions (cont)

- In Java, a lambda expression cannot stand alone.
- It must be assigned to a variable whose type is a functional interface:

```
MeasureInterface squareMeasurer = (Object obj) -> {
                 CSquare sqareObjct = (CSquare) obj;
                 return sqareObjct.getWidth() * sqareObjct.getWidth();
           };
```

- Now the following actions occur:
  - A class is defined that implements the functional interface.
  - The single abstract method is defined by the lambda expression.
  - An object of that class is constructed.
  - The variable is assigned a reference to that object.

Lambda Expressions (cont)

- Then the parameter variable to the calcAverage function is initialized by using the object:

```
MeasureInterface squareMeasurer = (Object obj) -> {
                    CSquare sqareObjct = (CSquare) obj;
                    return sqareObjct.getWidth() * sqareObjct.getWidth();
            };

CSquare[] squares = { new CSquare(1), new CSquare(2),
                      new CSquare(3), new CSquare(4) };

double average = calcAverage(squares, squareMeasurer);
System.out.printf("The average is: %.2f", average);
```

Inner Classes

- Trivial class can be declared inside a method:

```
public class MeasurerTester
{
      public static void main(String[] args)
      {
              public class MeasureSquare implements MeasureInterface
              {
                      . . .
              }
              . . .
              MeasureInterface sqrMeasurer = new MeasureSquare();

              CSquare[] squares = { new CSquare(4), new CSquare(3),
                                    new CSquare(6) };

              double average = calcAverage(squares, sqrMeasurer);
              System.out.printf("The average is: %.2f", average);

              . . .
      }
}
```

- An inner class is a class that is declared inside another class.

Inner Classes

- We can also declare inner class inside an enclosing class, but outside its methods.
  - It is available to all methods of enclosing class:

- Compiler turns an inner class into a regular class file with a strange name:
  MeasurerTester$1AreaMeasurer.class

- Inner classes are commonly used for utility classes that should not be visible elsewhere in a program.

Inner Classes

```
public class MeasurerTester
{
     public class MeasureSquare implements MeasureInterface
     {
                . . .
     }

     public static void main(String[] args)
     {
          MeasureInterface sqrMeasurer = new MeasureSquare();
          CSquare[] squares = { new CSquare(4), new CSquare(3),
                                new CSquare(6) };

          double average = calcAverage(squares, sqrMeasurer);
          System.out.printf("The average is: %.2f", average);
          . . .
     }
}
```

Self Check 10.21

- When would we place an inner class inside a class but outside any methods?
- Answer: When the inner class is needed by more than one method of the classes.

Questions?