Data Structures for Java

William H. Ford
William R. Topp

Interfaces

Bret Ford

Object Composition

- Class (client class) contains one or more objects of another class (supplier class).
- Each instance of the client class has an instance of a supplier class as subobject
- Termed the “has-a” relationship.

Inheritance in Java

- An “is a” relationship that involves sharing of attributes and methods among classes.
- A superclass defines a common set of attributes and operations.
- A subclass extends the resources in a superclass by adding its own data and methods.

Abstract Classes

- Define an abstract method in a class by preceding the signature with the keyword abstract and replacing the method body by “;”. Place the keyword abstract in the class header.
- Each subclass of an abstract class must override all of the abstract methods in the superclass.
- A program cannot create an instance of an abstract class.
- Provides only resources for a subclass and method declarations that can be used with polymorphism.

The Java Interface

- An interface is a *pure abstract* classlike structure.
- Contains only public abstract methods and public final static data.
- Classes implement the interface rather than extending it.
- Serves as a template for its implementing classes.

The Java Interface (concluded)

- Interface reference variables can be assigned references from any implementing class.
- Calling a method with an interface reference variable invokes polymorhpism.
- An interface may be implemented by more than one class.

Declaring an Interface

- Use the keyword *interface*.

  Example:

```
public interface InterfaceName
{
   public static final DATA_NAME = <value>;

   // method declaration uses only a signature
   public returnType methodName(<parameter list>);
}
```

Implementing an Interface

- The relationship between a class and an interface is established by using the keyword *implements*.
- The implementing class must provide method declarations for each method in the interface.

```
// class interface declaration
public class ClassName implements InterfaceName
{
   <implement each interface method>
}
```

The Measurement Interface

- The Measurement interface is a template for the definition of geometric classes that define the area and perimeter of shapes.

```
public interface Measurement
{
   public static final double PI = 3.14159265;

   double area();
   double perimeter();
}
```

Circle Class

```
public class Circle implements Measurement
{
   private double radius;

   // creates an instance with the specified radius
   public Circle(double radius)
   { this.radius = radius; }

   // interface method area() returns PI*radius(squared)
   public double area()
   { return PI * radius * radius; }

   // interface method perimeter() returns 2*PI*radius
   public double perimeter()
   { return 2 * PI * radius; }
```

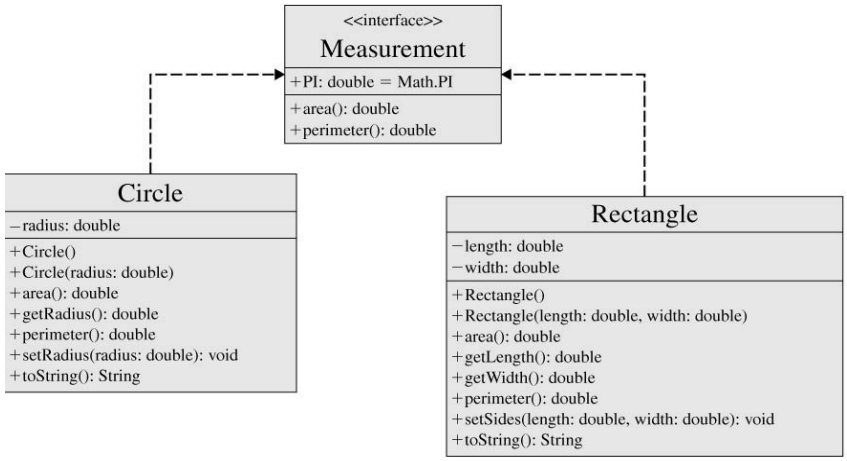
Circle Class (2)

```
    // access and update methods
    public double getRadius()
    { return radius; }

    public void setRadius(double radius)
    { this.radius = radius; }

    // returns a description of the object
    public String toString()
    { return "Circle with radius " + radius; }
}
```

UML for the Measurement Hierarchy



The Measurement interface and two implementing classes.

Using an Interface Type

- An interface reference variable can be assigned any object from a class that implements the interface, and polymorphism applies.

```
InterfaceName ref;
ImplementingClass obj = new ImplementingClass(...);

ref = obj;
// calls methodName() in ImplementingClass
ref.methodName();
```

resize()

```
public static void resize(Measurement m,
       double pct)
{
   if (m instanceof Rectangle)
   {
      // cast m as a Rectangle
      Rectangle r = (Rectangle)m;
      // use setSides() to resize length and width
      r.setSides(r.getLength()*pct,
                 r.getWidth()*pct);
   }
```

resize() (2)

```
        else if (m instanceof Circle)
        {
            // cast m as a Circle
            Circle c = (Circle)m;

            // use setRadius() to resize the radius
            c.setRadius(c.getRadius()*pct);
        }
    }
```

Interface Inheritance Hierarchy

- A parent interface may be used to derive a child interface. Use the extends keyword.

Example:

```
public interface DiagonalMeasurement extends Measurement
{
    public double diagonal();
}

public class Circle implements DiagonalMeasurement
{
    <declarations of area(), perimeter(), and diagonal()>
}
```

Extending Interface Implementations

- A class that implements an interface can be extended. The subclass implements the interface also.

Example:

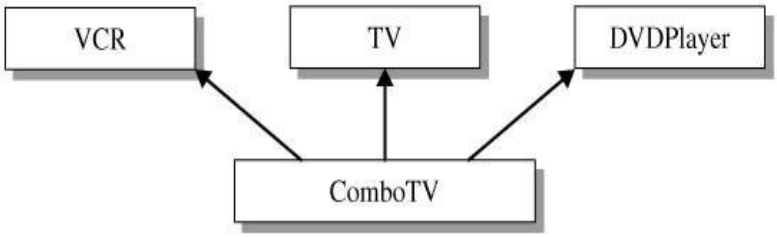
```
public class Rectangle implements Measurement
{ ... }

public class Square extends Rectangle
{
   <methods area(), perimeter() inherited from Rectangle
}
```

Multiple Interfaces and Inheritance

- Java only allows a class to extend one superclass (single inheritance).
- Some languages like C++ allow a class to inherit multiple superclasses.

Multiple Interfaces and Inheritance (continued)



TV, DVDPlayer, and VCR describe different electronic devices. Electronic stores carry ComboTV units that combine features from each of the devices.

Java Answer to Multiple Inheritance

- Allow a superclass to implement multiple interfaces.