CS360 Lecture 6
Inheritance

Thursday, February 19, 2004

Last Time

We compared C++ and Java. Specifically, we covered:

- How objects are constructed in Java. An object reference is created that refers to the object. It differs from pointers because the pointer cannot be dereferenced and if the JVM performed memory management and moved the object, the reference will still refer to the object.
- Composition. This is when a class contains instances of other classes.
- Passing objects to methods. When an object is passed to a method in Java, it is the object reference that gets copied into the method. How are primitive data types passed in Java?
- Static class variables and methods.
- Java garbage collection.

Constants

In Java you indicate that a variable of a primitive data type is constant by placing the word final in front of it. For example:
Final int N = 100;
Final double PI = 3.14;
Final int[] a = new int[] {0, 1, 2, 3};

In C++, any variable that is declared as a constant must also be initialised at the same time. This is not the case in Java. You could declare a constant in one statement, and then initialise it later in the program.

Look at the next example.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class IncrementTest extends JApplet
                           implements ActionListener {
   private Increment incrementObject, increment2;
   private JButton button;

   public void init()
   {
      incrementObject = new Increment( 5 );
      increment2 = new Increment( 3 );
```

```
      Container container = getContentPane();

      button = new JButton( "Click to increment" );
      button.addActionListener( this );
      container.add( button );
   }


   public void actionPerformed( ActionEvent
                                        actionEvent )
   {
      incrementObject.increment();
      increment2.increment();
      showStatus(
               incrementObject.toIncrementString()
               + " and "
               + increment2.toIncrementString() );
   }

}

class Increment {
   private int count = 0;
   private int total = 0;
   private final int INCREMENT;

   public Increment( int incrementValue )
   {
      INCREMENT = incrementValue;
   }

   public void increment()
   {
      total += INCREMENT;
      ++count;
   }

   public String toIncrementString()
   {
      return "After increment " + count +
               ": total = " + total;
   }

} // end class IncrementTest
```

Inheritance

Inheritance is a form of software reuse in which classes are created by absorbing an existing class's data (attributes) and methods (behaviours) and embellishing them with new or modified capabilities.

We have already seen inheritance in action. What class did we use?

Why is inheritance a good thing?

The original class in Java is called the superclass and the new class is called the subclass.

In a class hierarchy where there is more than one level, the superclass could be direct or indirect.

Java does not support multiple inheritance. What does this mean?

Protected Instance Variables

So far we have seen that classes, instance variables and methods can be declared private or public. What is the difference?

Question: If a superclass contains private instance variables, are these accessible from the subclass?

The protected access modifier makes instance variables and methods available to subclasses, but inaccessible from anywhere else (e.g. an instance of a class).

Overriding Methods

It is possible for subclasses to either just add to the superclass or modify the operation of the methods. In this case the subclass can call the superclass method by using the super keyword followed by the dot (.).

Let's look at an example.

Point2 is the superclass that will later be extended.

What does Point2 extend?
Why is there no data validation for x and y in class Point2?

Constructors are not inherited, but subclass constructors implicitly call the superclass constructor.

Method toString

Every method inherits from class Object. Object contains a method called toString that can be overridden by all subclasses. The method in the Object class is used as a placeholder for the other methods.

What effect will changing the instance variables in class Point2 to private have on the program?

```
public class Point2 {
   protected int x;
   protected int y;

   public Point2()
   {
      // implicit call to Object constructor occurs
   }

   public Point2( int xValue, int yValue )
   {
      x = xValue;
      y = yValue;
   }

   public void setX( int xValue )
   {
      x = xValue;
   }

   public int getX()
   {
      return x;
   }

   public void setY( int yValue )
   {
      y = yValue;
   }

   public int getY()
   {
      return y;
   }

   public String toString()
   {
      return "[" + x + ", " + y + "]";
   }
}
```

```
public class Circle3 extends Point2 {
   private double radius;
   public Circle3()
   {
      // implicit call to Point2 constructor occurs
   }

   public Circle3( int xValue, int yValue,
                                   double radiusValue )
   {
      // implicit call to Point2 constructor occurs
      x = xValue;
      y = yValue;
      setRadius( radiusValue );
   }

   public void setRadius( double radiusValue )
   {
      radius = ( radiusValue < 0.0 ? 0.0
                                   : radiusValue );
   }

   public double getRadius()
   {
      return radius;
   }
   public double getDiameter()
   {
      return 2 * radius;
   }

   public double getCircumference()
   {
      return Math.PI * getDiameter();
   }

   public double getArea()
   {
      return Math.PI * radius * radius;
   }
   public String toString()
   {
      return "Center = [" + x + ", " + y
                          + "]; Radius = " + radius;
   }

}
```

Finally, CircleTest3 is used to test both classes

```
import java.text.DecimalFormat;
import javax.swing.JOptionPane;

public class CircleTest3 {

   public static void main( String[] args )
   {
      Circle3 circle = new Circle3( 37, 43, 2.5 );

      String output = "X coordinate is "
                        + circle.getX() +
                      "\nY coordinate is "
                        + circle.getY() +
                      "\nRadius is "
                        + circle.getRadius();

      circle.setX( 35 );
      circle.setY( 20 );
      circle.setRadius( 4.25 );

      output += "\n\nThe new location and radius
               of circle are\n" + circle.toString();

      DecimalFormat twoDigits =
                        new DecimalFormat( "0.00" );

      output += "\nDiameter is " +
         twoDigits.format( circle.getDiameter() );

      output += "\nCircumference is " +
         twoDigits.format(
                        circle.getCircumference() );

      output += "\nArea is " +
               twoDigits.format( circle.getArea() );

      JOptionPane.showMessageDialog( null,output );

      System.exit( 0 );

   }
}
```

In the previous example we used protected instance variables in the superclass in order for the subclass to access them. This does improve performance slightly. Why?

However, it is better to use private instance variables to encourage proper software engineering. There are several problems with using protected instance variables:

- Subclass could assign invalid values to superclass instance variables.
- Subclasses should only depend on the superclass services. If the superclass implementation changes, then the subclass will also need to be changed.

What do we need to change in our classes in order to use private instance variables in the superclass?

Multiple Level Inheritance

Let us add to the complexity of our program. Write a class Cylinder that extends the Circle class. Cylinder should contain one instance variable for the height and methods for getting the height, area and volume. The class should also contain two constructors, a method to set the height and a toString method.

Object Cloning in Java

What is the output of the following:

```
class User {
     public String name;
     public int age;
     public User( String str, int n)
     {
          name = str;
          age = n;
     }
}

class Test {
     public static void main( String[] args )
     {
          User u1 = new User( "bob", 112 );
          System.out.println( u1.name );
          User u2 = u1;
          u2.name = "bonno";
          System.out.println( u1.name );
     }
}
```

What should we do in order to create an exact copy of an object? There are no copy constructors in Java like there are in C++.

The answer is cloning!

```
class User implements Cloneable {
     public String name;
     public int age;
     public User( String str, int n)
     {
          name = str;
          age = n;
     }
     public Object clone()
                    throws CloneNotSupportedException
     {
        return super.clone();
     }
}

class Test {
     public static void main( String[] args )
     {
          User u1 = new User( "dolly", 112 );
          User u2 = null;
          try {
               u2 = (User) u1.clone();
          } catch (CloneNotSupportedException e){}

          System.out.println( u1.name );
          System.out.println( u2.name );

          u2.name = "bonno";
          System.out.println( u2.name );
     }
}
```