CS360 Lecture 7
Polymorphism

Tuesday, February 24, 2004

Last Time

We covered the following topics:

- Constants: in Java they do not have to be initialised at the time of declaration
- Inheritance: notion of super/subclasses, direct/indirect inheritance
- Protected instance variables
- Overriding methods
- `toString()` method
- Multiple level inheritance
- Cloning

Today we will cover:

- Polymorphism: Let us program in general
- Abstract classes
- Programming in the specific
- Interfaces

Relationships Between Classes in an Inheritance Hierarchy

Question: Can an object of a subclass be treated as an object of its superclass?

Question: In the point/circle/cylinder hierarchy we created last time, is it possible to create an array that could hold objects of any of those classes?

Polymorphism

Question: What is polymorphism?

Polymorphism means that a given category of objects can exhibit multiple identities at the same time. A Cat object is also a FourLegged object and Animal object all at the same time. The cat object should be able to use the FourLegged class methods and the Animal class methods.

Question: How did we indicate polymorphism in C++?

```
import javax.swing.JOptionPane;

public class ArrayOfObjects
{
  public static void main( String[] args )
  {
    String output = "";
```

```
    Point3 points[] = new Point3[4];

    points[0] = new Circle4( 43, 53, 1.4 );
    points[1] = new Point3( 76, 34 );
    points[2] = new Circle4();
    points[3] = new Point3();

    for(int i=0; i<4; i++)
      output += "\n\n" + points[i].toString();

    JOptionPane.showMessageDialog( null, output );

    System.exit( 0 );
  }
}
```

Question: What would happen if I changed the array above to be an array of Circle4 instead of an array of Point3?

Question: What do you think would happen if we added the following line to the code above?

```
output += points[0].getRadius();
```

A subclass object is a superclass object, but they are different.

A subclass object can be treated as a superclass object, but the reverse is not true.

Assigning a subclass object's reference to a superclass-type variable is safe. However, this reference can only be used to invoke superclass methods.

Question: When would we want to perform polymorphism? Could you think of examples?

Abstract Classes

Abstract classes are classes that never get instantiated. I.e. no objects are ever created of these classes.

Question: What would be the purpose of such a class?

As an example of a hierarchy that contains an abstract class, let us imagine that we have an abstract class `Shape` that cannot be instantiated. From this `Shape` class we can derive the concrete classes `Square`, `Circle` and `Triangle`.

The class `Shape` specifies that all its subclasses must contain methods for drawing and calculating the area. The methods in class `Shape` are specified as being abstract since

they don't contain any implementation code. The implementation code is specified in the subclasses.

In Java both the class and its methods need to be declared as being abstract. This is different from C++ where only the methods are declared as abstract.

Any class that contains an abstract method must be declared as abstract. Each concrete subclass of an abstract superclass must provide concrete implementations of the superclass's abstract methods.

Although abstract classes cannot be instantiated directly, we can use abstract classes to declare variables that can hold references to objects of any concrete class derived from the abstract classes.

For example, imagine that we have an abstract class `Shape` from which the concrete class `Point3` is derived, and `Circle4` is derived from `Point3`. It is possible to make the following declaration:

```
Shape shapes[] = new Shape[4];
shapes[0] = new Point3( 23, 43 );
shapes[1] = new Circle4( 45, 43, 4.2  );
```

Final Methods and Classes

A method that is declared as final in the superclass cannot be overridden in any of the subclasses. Thus private and static methods are implicitly final.

Final classes are classes that cannot be superclasses. In other words, no subclasses can be derived from final superclasses.

Polymorphism and Abstract Class Example

A company pays its employees on a weekly basis. The company has four types of employees:

- Salaried Employees: paid a fixed weekly salary
- Hourly Employees: paid by the hour and receive overtime pay
- Commission Employees: paid a percentage of their sales
- Salaried-Commission Employees: paid a base salary plus a percentage of sales.

For the current pay period, salaried-commission employees will be receiving a 10% addition to their salaries.

Design an object-oriented hierarchy to represent the situation. Specify which classes will be abstract.

Imagine that the following code is used to test the payroll hierarchy.

```
public class PayrollSystemTest
{
  public static void main( String[] args )
  {
    DecimalFormat twoDigits = new DecimalFormat
                                              ( "0.00" );

    Employee employees[] = new Employee[ 4 ];

    employees[ 0 ] = new SalariedEmployee
                       ( "John", "Smith",
                     "111-11-1111", 800.00 );
    employees[ 1 ] = new CommissionEmployee
                       ( "Sue", "Jones",
                     "222-22-2222", 10000, .06 );
    employees[ 2 ] = new BasePlusCommissionEmployee
                    ( "Bob", "Lewis",
                  "333-33-3333", 5000, .04, 300 );
    employees[ 3 ] = new HourlyEmployee
                    ( "Karen", "Price",
                  "444-44-4444", 16.75, 40 );

    String output = "";

    for ( int i = 0; i < employees.length; i++ )
    {
      output += employees[ i ].toString();

      if ( employees[ i ] instanceof
                        BasePlusCommissionEmployee )
      {
         BasePlusCommissionEmployee currentEmployee
               = ( BasePlusCommissionEmployee )
                 employees[ i ];

        double oldBaseSalary =
                     currentEmployee.getBaseSalary();
        output += "\nold base salary: $" +
                     oldBaseSalary;

        currentEmployee.setBaseSalary( 1.10 *
                                       oldBaseSalary );
        output += "\nnew base salary with 10%
                     increase is: $" +
                   currentEmployee.getBaseSalary();

       }
```

```
        output += "\nearned $" +
                    employees[ i ].earnings() + "\n";
    }

    for ( int j = 0; j < employees.length; j++ )
         output += "\nEmployee " + j + " is a " +
               employees[ j ].getClass().getName();

    JOptionPane.showMessageDialog( null, output );
    System.exit( 0 );
  }
}
```

Interfaces

An interface is a class that consists solely of abstract methods.

Remember that we said that Java does not support multiple inheritance. In other words, each class in Java can inherit from one class only.

Since this can be excessively restrictive, Java does support classes implementing an arbitrary number of interfaces.

```
abstract class Shape
{
  abstract protected double area();
  abstract protected double circumference();
}

class Circle extends Shape
{
  protected double r;
  protected static double PI = 3.14159;
  public Circle( double r ) { this.r = r; }
  public double area() { return PI*r*r; }
  public double circumference() { return 2*PI*r; }
}

class Rectangle extends Shape
{
  double w, h;
  public Rectangle( double w, double h )
  {
    this.w = w;
    this.h = h;
  }
```

```
  public double area() { return w * h; }
  public double circumference() { return 2*(w+h); }
}

interface Drawable
{
  public void setColor( Color c );
  public void setPosition( double x, double y );
  public void draw( DrawWindow dw );
}

class DrawableRectangle extends Rectangle
                                    implements Drawable
{
  private Color c;
  private double x, y;
  public DrawableRectangle( double w, double h )
  {
     super( w, h );
  }
  public void setColor( Color c ) { this.c = c; }
  public void setPosition( double x, double y )
  {
    this.x = x;
    this.y = y;
  }
  public void draw( DrawWindow dw )
  {
     dw.drawRect( x, y, w, h, c );
  }
}

class DrawableCircle extends Circle
                                    implements Drawable
{
  private Color c;
  private double x, y;
  public DrawableCircle( double rad ) {super(rad);}
  public void setColor( Color c ) { this.c = c; }
  public void setPosition( double x, double y )
  {
    this.x = x;
    this.y = y;
  }
  public void draw( DrawWindow dw )
  {
     dw.drawCircle( x, y, r, c );
```

```
  }
}

class Color { int R, G, B; }

class DrawWindow
{
  public DrawWindow() {};
  public void drawRect( double x, double y,
                        double width, double height,
                       Color col )
  {
    System.out.println( "Code for drawing a rect
                         needs to be invoked" );
  }
  public void drawCircle( double x, double y,
                          double radius, Color col )
  {
    System.out.println( "Code for drawing a circle
                         needs to be invoked" );
  }
}

class Test
{
  public static void main( String args[] )
  {
    Shape shapes[] = new Shape[3];
    Drawable drawables[] = new Drawable[3];

    DrawableCircle dc = new DrawableCircle( 1.1 );
    DrawableRectangle dr1 = new
                  DrawableRectangle( 2.5, 3.5 );
    DrawableRectangle dr2 = new
                  DrawableRectangle( 2.3, 4.5 );

    shapes[0] = dc;
    shapes[1] = dr1;
    shapes[2] = dr2;

    drawables[0] = dc;
    drawables[1] = dr1;
    drawables[2] = dr2;

    int total_area = 0;
    DrawWindow dw = new DrawWindow();
    for( int i=0; i<shapes.length; i++ )
```

```
    {
      total_area += shapes[i].area();
      drawables[i].setPosition( i*10.0, i*10.0 );
      drawables[i].draw( dw );
    }
    System.out.println( "Total area = " +
                                          total_area );
  }
}
```