Recitation 5. About abstract classes --and graphics windows

Introduction Consider a class Shape and some subclasses of it:

```
import java.awt.*;
// A Shape has a position given by (x,y) coordinates
public class Shape {
  public int x;      // (x,y) is the upper-left corner
  public int y;      // of the shape on the window

  // = a descriptor of this Shape
  public String toString()
    {  return "(" + x + ", " + y + ")";  }
}
```

```
// Constructor: a shape at (xp, yp)
public Shape (int xp, int yp)
        {  x= xp;  y= yp;  }

// draw this shape using Graphics g
public void drawShape(Graphics g)
        {  }
```

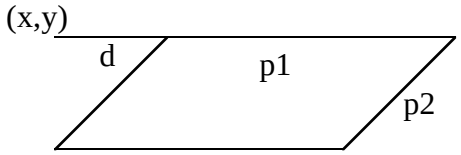
****

```
import java.awt.*;

// A parallelogram with horizontal length l1
// and the other length l2,whose top line starts d
units to the right of point (x,y)
public class Parallelogram extends Shape {
   int p1; // length of horizontal side
   int p2; // length of other sied. If right-leaning,
   int d;   // If d positive, right leaning:top line starts
         // d units to right of point (x,y). If d neg.,
                // left-leaning: bottom line starts abs(d)
                // units to the right of point (x,y)

   // Constructor: parallelogram at (x, y) of side
   // lengths p1 (horiz. side) and p2, leaning factor d
   public Parallelogram(int xp, int yp, int p1,
                              int p2, int d) {
     super(x, y);
     this.p1= p1; this.p2= p2; this.d= d;
   }

   // = description of this parallelogram
   public String toString() {
     return "parallelogram at " + super.toString() +
        ", sides " + p1 + " and " + p2 + ", distance " +
        d + " from " + x;
   }
}
```




```
// Draw parallelogram using graphics g
public void drawShape(Graphics g) {
  // Set xt and xb to the horizontal
  // coordinates of left pt of top
  // and bottom lines
    int xt= x+d;
    int xb= x;
    if (d < 0)
      {xt= x; xb= x+(-d);}
 // Set yb to the vertical coordinate of the
 //  bottom left point
    int yb= y+(int)(Math.round(
          Math.sqrt(p2*p2 - d*d)));

  // Draw the four lines
    g.drawLine(xt, y, xt+p1, y);
    g.drawLine(xt+p1, y, xb+p1, yb);
    g.drawLine(xb+p1, yb, xb, yb);
    g.drawLine(xb, yb, xt, y);
}
```

It should be easy to see how to write a class Square that extends Parallelogram, a class Circle that extends Shape, a class Triangle that extends Shape, a class RightTriangle that extends Triangle, and so on.

There are two problems associated with class Shape.

Problem 1. Class Shape is not meant to be instantiated, but there is no way to prohibit this.

The only thing that all shapes have in common is a POSITION, given by a pair (x,y) of coordinates, and the sole purpose of class Shape is to provide this position for any shape that is an instance of a subclass of it.

Problem 2. Method Shape.drawShape is present only so that other classes will override it, and all subclasses are expected to override it, but there is no way to force them to do so.

If we didn't define Shape.drawShape, then it couldn't be overridden, and the following would be illegal:

```
Shape s= new Parallelogram();   s.drawShape();
```

However, Shape.drawShape really doesn't do anything, and it is expected that each subclass will override it. But we can't force the subclass to override it.

The solution is to use an abstract class. Let us rewrite class Shape as follows:

```
import java.awt.*;
// A Shape has a position given by (x,y) coordinates
public abstract class Shape {
  public int x;      // (x,y) is the upper-left corner
  public int y;      // of the shape on the window

  // Constructor: a shape at (xp, yp)
  public Shape (int xp, int yp)
     {  x= xp;  y= yp;  }
}
```

```
// = a descriptor of this Shape
  public String toString()
     {  return "(" + x + ", " + y + ")";  }

  // draw this shape using Graphics g
  public abstract void
       drawShape(Graphics g);
```

Method drawShape now contains the prefix abstract, and its body has been replaced by a semicolon. It is an "abstract method". Also, the first line of the class definition has the prefix abstract. This means that the class itself is abstract.

Rules concerning an abstract class. There are a few rules concerning the use of keyword abstract.

Rule 1. A class that is defined to be abstract cannot be instantiated. For example, with class Shape as just defined, the expression **new** Shape() is illegal, and the program will not compile. So, if you don't want the user to be able to instantiate a class, just make it abstract.

Rule 2. If a method of a class is abstract, then the class must be defined to be abstract as well. Thus, since method drawShape abstract, we class Shape must also be abstract.

Rule 3. If a method of a class C is abstract, then a subclass of C either must also be abstract or must implement the method. So, making a method abstract forces any non-abstract subclass to implement the method.

Our two problems are therefore solved SYNTACTICALLY. For problem 1, make class Shape abstract, and it is syntactically illegal to instantiate it. For problem 2, make drawShape abstract, and subclasses MUST implement it.

.

Drawing in a Graphics window

Classes JFrame, JPanel, Canvas, and some others, have the facility to draw lines, circles, text, etc. on them. Method paint is defined in them (it does nothing), and it can be overridden. It is called BY THE SYSTEM whenever the system thinks the window should --panel, canvas, etc.-- should be redrawn, perhaps because it just became visible. Here's an example:

```
import java.awt.*;

// A simple graphics window.
public class GraphicsG extends Frame {
    /* Repaint the Graphics Window with a titled black
       rectangle and a titled red circle. */
    public void paint(Graphics g) {
        g.setColor(Color.black);
        g.drawRect(77,10,40,30);
        g.drawString("rectangle",77,60);
        g.setColor(Color.red);
        g.drawOval(15,10,30,30);
        g.drawString("circle",15,60);
  }
}
```

```
/** Main program for first graphics application.
public class Draw {
    /* Create a graphics window named d*/
    public static void main(String args[]) {
        GraphicsG d = new GraphicsG();
        d.resize(200,150);
        d.move(0,75);
        d.setTitle("Graphics Window");
        d.show();
}
```

YOU DON'T CALL METHOD paint; the system does. If you want the frame repainted, call method repaint().

Parameter g of method paint has type Graphics. Graphics contains many methods for drawing in the frame (panel, canvas, etc.). Here are a few of them:

Method call g.setColor(Color.black);
Figures and text can be drawn in different colors. Other colors: lightGray, gray, darkGray, black, red, pink, orange, yellow, green, magenta, cyan, blue

Statement Color c= g.getColor();
Store the current drawing color in a new variable c.

Method call g.drawLine(x1, y1, x2, y2);
Using the current color, draw the line from pixel (x1,y1) to pixel x2,y2.

Method call g.drawRect(x, y, w, h);
Using the current color, draw the outline of a rectangle with upper left pixel (x,y), width w, height h

Method call g.fillRect(x, y, w, h);
Using the current color, fill in the rectangle that would be drawn using g.fillRect(x,y,w,h);

Method call g.drawOval(x, y, w, h);
Using the current color, draw the outline of the largest ellipse that fits in the rectangle that would be drawn using g.drawRect(x, y, w, h);

Method call g.fillOval(x, y, w, h);
Using the current color, fill in the largest ellipse that would be drawn using g.drawOval(x, y, w, h);

Method call g.drawString(s, x, y);
s is a String. Using the current color, draw the characters in s beginning at pixel (x,y).