Overview — Please Read!

You have now studied the basic components of Java classes: instance variables (representing the object’s properties) and methods (representing the object’s capabilities). But with what we know so far it is very difficult to do even simple tasks. For instance, suppose all we want to do is to plot a few points on the screen. With what we know now, even this is unnecessarily complicated. For example, here is a portion of a program to plot six points, using only features we’ve seen so far in Java (plus a method, “`Integer.parseInt`”, that I have shown to only a few students):

```java
public class Plot6 extends Frame {
    // Constructor -- inputs the point locations and then plots each
    // point as a 1-by-1 rectangle:
    public Plot6() {
        // First point: ask user to enter x, y values of point to be plotted:
        int x = Integer.parseInt(JOptionPane.showInputDialog("First x"));
        int y = Integer.parseInt(JOptionPane.showInputDialog("First y"));
        Rectangle p1 = new Rectangle(x,y);
        p1.setSize(1,1);

        // Second point: ask user to enter x, y values of point to be plotted:
        x = Integer.parseInt(JOptionPane.showInputDialog("Second x"));
        y = Integer.parseInt(JOptionPane.showInputDialog("Second y"));
        p1 = new Rectangle(x,y);
        p1.setSize(1,1);
    }

    ... and so on. It is inconvenient for the user to have to input x and y values, and it is inconvenient to have to keep using the “`setSize`” method each time we create a new rectangle.

    What we would like is a class named “Pt” that behaves like the `Rectangle` class, but has a different default size, namely 1-by-1. We would also like to be able to just let the user click on the screen where a new point is to go, rather than enter points through the keyboard.

    We can achieve the first goal by using the notion of inheritance. We will create a class named `Pt` that `inherits` all the properties and capabilities of the `Rectangle` class, but that has the additional quality that it is just a single pixel wide and a single pixel high. We can achieve the second goal by using some capabilities of the `Rectangle` class that we have not previously seen, namely, the ability
to recognize and respond to mouse events. Currently, this ability is latent — in order to make use of it, we must create a subclass of Rectangle that includes code to “activate” the mouse-specific features.

Your goal today is to create a very simple drawing program that lets the user click on the screen to create a line drawing.

Pt — a subclass of the Rectangle class

Enter the following code in a file named “Pt.java” (for now you may omit the comments):

```java
import wheels.users.*;

/**
 * Pt -- a subclass of the wheels.users.Rectangle class
 * Objects in the Pt class are points at specified locations in the
 * surrounding frame.
 */

public class Pt extends Rectangle {

/**
 * Constructor -- construct a point by giving it an (x,y) location.
 */
 public Pt(int x, int y) {
   super(); // First, create a Rectangle (the superclass)
   this.setColor(java.awt.Color.BLACK);
   this.setLocation(x,y);
   this.setSize(1,1);
 }

 /**
 * Override the default constructor:
 */
 public Pt() { } // empty body
}

Now create a second file named “PlotPoints.java” containing this code:

```java
import wheels.users.*;

public class PlotPoints extends Frame {

   public PlotPoints() {
     Pt pt = new Pt(10,10);
     pt = new Pt(20,20);
     pt = new Pt(20,30);
     pt = new Pt(30,30);
     pt = new Pt(30,20);
   }
```
pt = new Pt(30,40);
}

public static void main(String[] args) {
    PlotPoints demo = new PlotPoints();
}
}

Compile and run the “PlotPoints” program to make sure that it really does plot points.

Of course, this only partially meets our requirements. We would like the user to be able to click on the frame where the points should go.

**PlotRegion — another subclass of the Rectangle class**

Enter the following code into a file named “PlotRegion.java” (once again, you may omit the comments to save time):

```java
import wheels.users.*;
import java.awt.Color;
import java.awt.event.*;

public class PlotRegion extends Rectangle {

    /**
     * Constructor -- build a PlotRegion whose upper left corner is at (x1, y1) and whose lower right corner is at (x2, y2).
     */
    public PlotRegion(int x1, int y1, int x2, int y2) {
        super(); // create a Rectangle
        int width = x2 - x1 + 1; // calculate horizontal size
        int height = y2 - y1 + 1; // calculate vertical size

        this.setLocation(x1, y1); // place the upper left corner
        this.setSize(width, height); // set the dimensions
        this.setFillColor(Color.WHITE); // plot on a white background
        this.setFrameColor(Color.BLACK); // black border all around
    }

    /**
     * Override the default constructor:
     */
    public PlotRegion() {
    }

    /**
     * Method "mouseClicked" is inherited from Rectangle, but the Rectangle class doesn't do anything when the mouse is clicked;
     * override this method so that it does something:
     */
    public void mouseClicked(MouseEvent e) {
```
Due Tues., 27 Feb.

// Find out where the mouse was clicked:
int x = e.getX();
int y = e.getY();

// Plot a new point where the mouse was clicked
Pt p = new Pt(x,y);

And here’s the test code, which should go into a file named “PlotPoints2.java”:

import wheels.users.*;

public class PlotPoints2 extends Frame {
    private PlotRegion _plotArea;
    private ConversationBubble _instructions;

    public PlotPoints2() {
        _plotArea = new PlotRegion(0,0,200,200);
        _instructions = new ConversationBubble("Click in the square");
        _instructions.setLocation(205,205);
    }

    public static void main(String[] args) {
        PlotPoints2 demo = new PlotPoints2();
    }
}

Compile and execute PlotPoints2.java to see what it does.

Experiments

Before we get to the main goal of the lab, here are some experiments for you to try:

1. Almost anything can go in the method named “mouseClicked”. For instance, if we want the PlotRegion to change to green when we click on it, we can write:

        public void mouseClicked(MouseEvent e) {
            this.setColor(Color.GREEN);
        }

Modify your PlotRegion.java class as just shown. Test it to make sure that it really does turn green when you click on it.

Now try each of the following modifications to mouseClicked. As you complete each one, raise your hand so I can see it:

(a) [THIS ITEM REMOVED!]
(b) Have the PlotRegion increase in size by ten pixels in each direction. This is tricky — first, you need to get its current size using the \texttt{getWidth} and \texttt{getHeight} methods and saving these in a pair of local variables. Then use the values in these local variables along with a call to \texttt{setSize} to achieve the desired effect.

(c) Create an ellipse in the center of the PlotRegion. (Be careful — what if your PlotRegion is not located in the upper left corner of the frame?)

2. You can also add other methods to a subclass that are not shared by the parent class. For instance, one of the things we might like to know about a PlotRegion is the location of the last place where the mouse was clicked. You can create a pair of instance variables in the PlotRegion class to hold this information and update these values inside the \texttt{mouseClicked} method. You can then write a pair of “get” methods to retrieve these values (for instance, \texttt{"getLastX()"} and \texttt{"getLastY()"}). Try this. (You won’t immediately see any way to test this — wait until Wednesday’s class!)

3. Other experiments: try the first few programming projects (“Modifying Programs”) at the end of chapter 3 (page 125). The source code for BlobApp, etc., can be downloaded from the course web site.

\textbf{The Lab Assignment}

Create a Java class named “\texttt{DrawingTool}” that extends \texttt{Rectangle}. It should have the property that, whenever the user clicks the mouse inside the drawing tool, a line will be drawn from the last point clicked to the most recent point clicked. The first time the mouse is clicked, only a single point is plotted; after that, each mouse click adds one line to the drawing. (You will need an “if” statement to check for this — see Wednesday’s class notes.) You may also use the \texttt{Pt} class from today’s lab if you wish.

Write a program that tests your drawing tool.

Show me the working program and hand in the fully commented Java code.

We will talk about this in class on Wednesday if you don’t know how to begin.