Announcements

Program #6 write-up now includes a note that explains the structure of my solution (and thus suggests a way to proceed if you haven’t worked that out yet).

Final embedded hwk, OWL hwk dates set. See CourseWeek link as usual.

Final programming assignment will be due 12/11

Final exam - December 17, 10:30 AM
Where we are:

We’ve seen all the major constructions in the language - statement level / class level

Reuse - packaging - classes, inheritance

Codification - “... let’s all be on the same page..”

These ideas come together most broadly with graphics
Graphics

Next/last big topic. Fun, dramatic, useful, etc.

Interactive events driven by buttons, mouse clicks, and so forth the dominant setting for modern computing
The Opoly game - a simplified version of the game Monopoly

Let’s look at it at work...
Features of graphical Opoly

• Runs in its own window
• Actions are triggered by button clicks and menu selection
• The picture changes with each triggered action
• Simple shapes are drawn on the screen

How does all of this happen?
A JFrame is like a window frame.

It comes with a content pane, a container that holds things (surfaces..)

We’ll place on the surface something we can draw on / write on / add hardware to: a JPanel

Then we do just that: we write on it, draw on it, color it.
JFrame (frame)

contentPane

JPanel (panel - is “affixed” to contentPane)
Drawing area is organized "upside down"
import java.awt.*;
import javax.swing.*;

public class FirstGraphics{

    public static void main(String[] args){
        JFrame frame = new JFrame("Getting Started");
        Container c = frame.getContentPane();
        BabyGeoPanel p = new BabyGeoPanel(Color.green);
        c.add(p); // add panel to frame's container
        frame.pack(); // prepares frame for display
        frame.setVisible(true); // reveals frame (window)

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

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

public class BabyGeoPanel extends JPanel{

    public BabyGeoPanel(Color g){
        setPreferredSize(new Dimension(700,300));
        setBackground(g);
    }

    public void paintComponent(Graphics g){
        super.paintComponent(g);
        g.setColor(Color.red);
        g.fillRect(10,20,100,100);
        g.setColor(Color.blue);
        g.drawOval(200,20,300,100);
        g.setColor(Color.black);
        g.drawRect(200,20,300,100);
        g.drawLine(0,0,200,20);
    }
}
We consolidate (package!) much of the windowing machinery (which will be common to most of what we do) into a single class called

DisplayWindow
import java.awt.*; import javax.swing.*;
public class DisplayWindow extends JFrame{
    private Container c;
    public DisplayWindow(){
        super("Display");
        c = this.getContentPane();
    }

    public void addPanel(JPanel p){
        c.add(p);
    }

    public void showFrame(){
        this.pack();
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); }
}

import java.awt.*; import javax.swing.*; import java.util.*;

public class BabyGraphics2{

    public static void main(String[] args){
        DisplayWindow d = new DisplayWindow();
        PositionPanel p = new PositionPanel(Color.red);

        Scanner s = new Scanner(System.in);
        int x = s.nextInt(); int y = s.nextInt();
        p.setPt(x,y);
        d.add(p);
        d.showFrame();
    }
}

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

public class PositionPanel extends JPanel{

    private int x, y;
    private Color c;

    public PositionPanel(Color c){
        super();
        setPreferredSize(new Dimension(500,500));
        this.c = c;
    }

    public void setPt(int a, int b){x = a; y = b;}
}
public void paintComponent(Graphics g) {
    super.paintComponent(g); // clears surface
    g.setColor(c);
    g.drawOval(x, y, 2, 2); // a tiny circle (ht = width)
    g.setColor(Color.black);
    String s = "(" + x + "," + y + ")";
    g.drawString(s, x, y); // writes on drawing surface
}

Note: NW dot corner at (x,y) String s at (x,y)
red dot
What we know so far..

• We can put up a window, pass it a panel (JPanel)
• We can draw ovals, circles, lines, rectangles..
• We can set colors
Java’s event model

What Scanner-based interactions are like.. (a script, as in BabyGraphics2)

What modern computing is like - it’s all clicking

To make mouse-related events in useable Java, programs needs to know:

1) What event to listen for
2) Clarify who’s listening
3) Provide a mechanism for reacting to event
   -- think about Gopoly
Most conspicuous component - the JButton

(from Swing library - javax.swing)
import java.awt.*;

public class BabyButtonDriver{

    public static void main(String[] args){
        DisplayWindow d = new DisplayWindow();
        BabyControlPanel p = new BabyControlPanel();
        d.addPanel(p);
        d.showFrame();
    }
}


import java.awt.*; import javax.swing.*;  
import java.awt.event.*; // needed for event handling

public class BabyControlPanel extends JPanel implements ActionListener{
    JButton quit = new JButton("Quit"); // make button object

    public BabyControlPanel(){
        setPreferredSize(new Dimension(500,500));
        setBackground(Color.red);
        this.add(quit); // place button in panel
        quit.addActionListener(this); // panel is listener for button
    }

    public void actionPerformed(ActionEvent e){
        if (e.getSource() == quit) // when panel hears button
            System.exit(0); // quit!
    }
}
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class BabyControlPanel2 extends JPanel implements ActionListener{

    JButton quit = new JButton("Quit"); // a quit button
    JButton color = new JButton("Toggle Color"); // color btn

    boolean toggle = false;

    public BabyControlPanel2(){
        setBackground(Color.red);
        this.add(quit);
        quit.addActionListener(this);
        this.add(color);
        color.addActionListener(this);
    }
}
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == quit) System.exit(0); //quit!

    else if (e.getSource() == color) {
        if (toggle) setBackground(Color.red);
        else setBackground(Color.blue);
        toggle = !toggle;
    }
}
}
BabyControlPanel2 summary

Two buttons (JButtons)

Note the `ActionListener` interface - one method to implement -- actionPerformed

Note `getSource()` method - from what class?

Note import statements

Heart of the matter: the “listening” mechanism
The basic script for event handling

• Create components for generating an event - a button, for example
• Identify listener for event
• Link listener, event generator
• Enable listener to listen and act
• Describe actions to take when events are triggered
One more basic example: a panel holds state.

```java
public class OvalsPanel extends JPanel implements ActionListener{

    JButton quit = new JButton("Quit");
    JButton ovals = new JButton("Ovals");
    int yVal = 10;
    boolean even = true;

    public OvalsPanel(){
        setPreferredSize(new Dimension(500,500));
        this.add(quit);  // place button in panel
        quit.addActionListener(this);
        this.add(ovals);
        ovals.addActionListener(this);
    }
}
```
public void paintComponent(Graphics g){
    super.paintComponent(g);
    if (even) g.setColor(Color.blue);
    else g.setColor(Color.red);
    g.fillOval(100,100 + yVal,100,30);
}

public void actionPerformed(ActionEvent e){
    if (e.getSource() == quit)
        System.exit(0);
    else
        if(e.getSource() == ovals){
            yVal = yVal + 15; even = !even; //state changed!
            repaint();
        }
}