Announcements

Program 7 available, due 12/11
All embedded, OWL assignments up, various due dates; see CourseWeek link on website
Final Exam: 10:30-12:30 12/17, Boyden Gym
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...
JMenu  JButton  JLabel  fillRect  drawRect
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"

- Increasing x
  - (10, 30)
- Increasing y
  - (22, 105)
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 much of the windowing machinery (which will be common to most of what we do) into a single class called

DisplayWindow
```java
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); } }
```
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 usage work in Java, programs needs to know:

1) What event to listen for
2) Clarify who’s listening
3) Provide a mechanism for event reaction
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(){
        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
Saving State in Event-based Programming
A more sophisticated example

Our same JPanel/DisplayWindow model

The JPanel object we create will have a changing state associated with it.

Analogous - of course - to the "age" field of an Infant object.

State acts like a kind of memory

    myKid.anotherMonth();

Uses the "remembered" age of myKid to figure the new age of the kid.
public class FlatLineDriver{

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

public class FlatLineDriver{

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

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

public class FlatLineDriver{

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

Construct p

d.add(p)

driver

FlatLinePanel p

JPanel

JFrame

DisplayWindow d
public class FlatLinePanel extends JPanel implements ActionListener {

    JButton draw = new JButton("Draw Line");
    private int height = 100;

    public FlatLinePanel()
    { setPreferredSize(new Dimension(500,700));
      this.add(draw);  // places button
      draw.addActionListener(this);
    }
public class FlatLinePanel extends JPanel implements ActionListener{

    JButton draw = new JButton("Draw Line");
    private int height = 100;

    public FlatLinePanel()
    {
        this.add(draw);  // places button
        draw.addActionListener(this);
    }
}
public class FlatLinePanel extends JPanel implements ActionListener{

    JButton draw = new JButton("Draw Line");
    private int height = 100;

    public FlatLinePanel()
    {
        this.add(draw);  // places button
        draw.addActionListener(this);
    }
}
public void paintComponent(Graphics g){
    super.paintComponent(g); // clears screen
    g.drawLine(0,height,500,height); }

public void actionPerformed(ActionEvent e){
    if(e.getSource() == draw)
    {
        height = height + 20; // changes state
        repaint();
    }
}
public void paintComponent(Graphics g){
    super.paintComponent(g); // clears screen
    g.drawLine(0,height,500,height); }

public void actionPerformed(ActionEvent e){
    if(e.getSource() == draw)
    {
        height = height + 20;//changes state
        repaint();
    }
}
public void paintComponent(Graphics g){
    super.paintComponent(g); // clears screen
    g.drawLine(0,height,500,height);
}

public void actionPerformed(ActionEvent e){
    if(e.getSource() == draw) {
        height = height + 20; // changes state
        repaint();
    }
}
One more example of very basic event-based programming - with one new component:

A JTextField

This component lets you type controlling information at the panel

(so: different than the Scanner-based keyboard input we're used to)
import java.awt.*;
import javax.swing.*
import java.awt.event.*;

public class PhrasePanel extends JPanel
    implements ActionListener {
    JButton quit = new JButton("Quit");
    JButton place = new JButton("Place");
    JTextField xVal = new JTextField(5);
    JTextField yVal = new JTextField(5);
    JTextField phrase = new JTextField(20);
    int x, y;
public PhrasePanel() {
    setPreferredSize(new Dimension(400,600));
    setBackground(Color.green);
    this.add(quit); // place button in panel
    quit.addActionListener(this);
    this.add(place); // place button in panel
    place.addActionListener(this);
    this.add(xVal); this.add(yVal);
    this.add(phrase); } // textfields: no listener

    public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.drawString(phrase.getText(),x,y);
    }
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == quit)
        System.exit(0);
    else if (e.getSource() == place) {
        x = Integer.parseInt(xVal.getText());
        y = Integer.parseInt(yVal.getText());
        repaint();
    }
}
Now: a very big idea

In complex graphics programming, separate the underlying computational model from the rendering code.

Example: a program that, at each mouse click, draws up to 100 random red points in a window.

The model: generates 100 random points

The renderer: takes those points, and on a JPanel, makes the appropriate number of suitably positioned red circles.
public class PointTester{

    public static void main(String[] args) {
        DisplayWindow display = new DisplayWindow();
        PointPix p = new PointPix(); // p is the “model”
        BabyControlPanel4 b = new BabyControlPanel4(p);
        display.add(b);
        display.showFrame();
    }
}

    model (reference to model) passed to panel; panel will invoke, manipulate model
import java.awt.*;

public class PointPix{

    private Point[] points = new Point[100];

    public void genPoints(){
        for (int j = 0; j < 100; j++)
            points[j] = genRanPoint();
    }

    public Point genRanPoint(){
        return(new Point((int)(400*Math.random()),
                         (int)(400*Math.random())));
    }

    public Point[] getPoints(){return points;}
}

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

public class BabyControlPanel4 extends JPanel implements ActionListener{

    JButton quit = new JButton("Quit");
    JButton points = new JButton("Points");
    JTextField count = new JTextField(5);
    int pointCount = 0;
    PointPix model; // the key model component
    Point[] pts;
public BabyControlPanel4(PointPix p) {
    model = p;  // assignment links model to panel
    this.add(quit);  // place button in panel
    quit.addActionListener(this);
    this.add(points);  // place button in panel
    points.addActionListener(this);
    this.add(count);
}
public void paintComponent(Graphics g){
    super.paintComponent(g);
    g.setColor(Color.red);
    for(int j = 0; j < Math.min(100, pointCount); j++){
        g.fillOval((int)pts[j].getX(), (int)pts[j].getY(), 8, 8);
    }
}
public void actionPerformed(ActionEvent e){
    if (e.getSource() == quit)
        System.exit(0);
    else
        if(e.getSource() == points){
            pointCount =
                Integer.parseInt(count.getText());
            model.genPoints();
            pts = model.getPoints();
            repaint();
        }
}
Programming Assignment Seven - a flying saucer
Book ideas

• The Learning Dashboard

• Adaptive Embedded Questions

• Problem Statistics