CS 121 - Intro to Programming: Lecture 26

Announcements
4/28 - embedded 13 due
5/3 - OWL assignment #13 due
5/5 - final programming assignment(s) due

Final Exam: Tues May 11, 1:30 PM, Totman
Java’s event model
What modern computing is like

Your program needs to know:
1) What to listen for
2) Identify listener
4) Marry 1 and 2
3) Provide a mechanism for reacting properly to events
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.*;

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); // listener for button: panel
    }

    public void actionPerformed(ActionEvent e){
        if (e.getSource() == quit) System.exit(0);
    }
}
So far we’ve seen several Swing components for providing/displaying input:

Jbutton, JTextField, JTextArea, JLabel, JMenu, etc.

We’ve also considered the mouse as a component for input
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class Locator extends JPanel implements MouseListener{
    int x,y;
    String location = "";

    public Locator(){
        addMouseListener(this);//panel listens for mouse
    }

    public void paintComponent(Graphics g){
        g.drawString(location,x,y);
    }
}
public void mouseClicked(MouseEvent e) {
    x = e.getX();  y = e.getY();  
    location = "(" + x + "," + y + ")";
    repaint();
}

public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
public void mousePressed(MouseEvent e) {}
public void mouseReleased(MouseEvent e) {}
The main points:

• **MouseListener** - not **ActionListener**

• The listener registration code is different

• There are five methods in the interface - but here we’re only using 1!

• Another interface: **MouseMotionListener** (deals with mouse motion, not just button pressing..)
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class HandDraw extends JPanel {
    implements MouseMotionListener{

        int oldX = 0; int oldY = 0;
        int newX = 0; int newY = 0;

        public HandDraw(){
            addMouseMotionListener(this);
        }
    }
public void paintComponent(Graphics g) {
    g.drawLine(oldX, oldY, newX, newY);
}

public void mouseDragged(MouseEvent e) {
    oldX = newX; oldY = newY;
    newX = e.getX(); newY = e.getY();
    repaint();
}

public void mouseMoved(MouseEvent e) { }
LineClicker2 - a review example..
Layouts: Classes that allow you to organized GUI components geographically

FlowLayout
GridLayout
BorderLayout

Another deeper point: Panels can be nested (that is, embedded in other panels)

Following the lead of the text, we'll do a dummy calculator: +,-,\*,/, and four other components
Flow Layout
import java.awt.*;
import javax.swing.*;

public class FlowDisplay extends JPanel {
    JButton plus = new JButton("+");
    JButton minus = new JButton("-");
    JButton times = new JButton("x");
    JButton divide = new JButton("/");
    JButton quit = new JButton("Quit");
    JLabel enterLabel = new JLabel("enter: ");
    JTextField enter = new JTextField(8);
    JLabel result = new JLabel("0.0");
public FlowDisplay() {
    setLayout(new FlowLayout);
    add(plus);
    add(minus);
    add(times);
    add(divide);
    add(enterLabel);
    add(enter);
    add(result);
    add(quit);
}
Grid Layout
public GridDisplay()
{
    setLayout(new GridLayout(4,2));
    add(plus);
    add(minus);
    add(times);
    add(divide);
    add(enterLabel);
    add(enter);
    add(result);
    add(quit);
}

// note: can't address cells directly
Display

+  -  x  /

enter:  0.0

Quit

Border Layout
import java.awt.*;
import javax.swing.*;
public class BorderDisplay extends JPanel {
    JButton plus = new JButton("+");
    JButton minus = new JButton("-");
    JButton times = new JButton("x");
    JButton divide = new JButton("/");
    JButton quit = new JButton("Quit");
    JLabel enterLabel = new JLabel("enter: ");
    JTextField enter = new JTextField(8);
    JLabel result = new JLabel("0.0");

    JPanel northPanel = new JPanel();
    JPanel centerPanel = new JPanel();
public BorderDisplay()
{
   // note embedded JPanels
   setLayout(new BorderLayout()); // carve up territory
   northPanel.add(plus);
   northPanel.add(minus);
   northPanel.add(times);
   northPanel.add(divide);
   add(northPanel, BorderLayout.NORTH);
   centerPanel.add(enterLabel);
   centerPanel.add(enter);
   centerPanel.add(result);
   add(centerPanel, BorderLayout.CENTER);
   add(quit, BorderLayout.SOUTH);
}
}
Revisit Red - Dots example

PointTester example
Displays up to 100 random points, but they can fall in the “zone” where the control hardware lives

We’ll re-implement this using Border Layout, where we’ll confine the dots strictly to the center zone
Structure of separation

**MVC - Model-View-Controller pattern**

- **Model** - where underlying computation occurs
- **View** - where results are rendered graphically
- **Controller** - where model/view controlled

- In fairly simple toy examples we’ll merge the viewer and controller
controller  

model  

view
SOUTH listener: DotsBorder object (this)
NORTH listener: DotsBorder object (this)

CENTER needs its own drawing machinery, so we need to develop a CenterPanel paintComponent, etc. It will need access to the model.
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class DotsBorder extends JPanel implements ActionListener{
    JButton quit = new JButton("Quit");
    JButton points = new JButton("Points");
    JTextField count = new JTextField(5);
    JPanel northPanel = new JPanel();
    CenterPanel centerPanel = new CenterPanel(Color.red);

    int pointCount = 0;
    PointPix model;
    Point[] pts;
public DotsBorder(PointPix p) {
    setPreferredSize(new Dimension(400, 600));
    setLayout(new BorderLayout());
    model = p;
    quit.addActionListener(this);
    northPanel.add(points);
    points.addActionListener(this);
    northPanel.add(count);
    add(northPanel, BorderLayout.NORTH);
    add(quit, BorderLayout.SOUTH);
    add(centerPanel, BorderLayout.CENTER);
}
private class CenterPanel extends JPanel{
    private Color clr;

    public CenterPanel(Color c){clr = c;}

    public void paintComponent(Graphics g){
        super.paintComponent(g);
        g.setColor(clr);
        for(int j =0; j < Math.min(100,pointCount); j++){
            g.fillOval((int)pts[j].getX(),
                        (int)pts[j].getY(),8,8);
        }
    }
}

Private instance variable in outer class
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();
            centerPanel.repaint();
        }
    }
}
private pts....
private model

pts .....
centerPanel

DotsPanel

inner class