CS 121 – Intro to Programming:Java - Lecture 23

Announcements

Program 5 - Sudoku - due Friday at 4:30
Program 6 - Scrabble - due Monday (OWL)
Ch 11 OWL exercises due 11/26
Ch 12 OWL exercises up today
I’m open for business 130-230 today

Final Exam: Wednesday, Dec 17, 8 AM
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)
import java.awt.*; import javax.swing.*;

double class EyePanel extends JPanel{

    public EyePanel(Color g){
        setPreferredSize(new Dimension(500,500));
        setBackground(g);
    }

    public void paintComponent(Graphics g){
        super.paintComponent(g);
        g.drawRect(20,20,300,60);
        g.drawOval(20,20,300,60);
        g.drawOval(140,20,60,60);
        g.fillOval(165,45,10,10);
        g.drawString("A green eye ",20,100);
    }
}
increasing x

- (10, 30)

increasing y

- (22, 105)
Two big themes, revisited

Factor out the general framework, using inheritance (good news)

“Write once, run everywhere” principle in Java starts to break down in Java when we begin to do graphics (bad news)
Inheritance and the frame/panel machinery

We’ll split off the frame code and make a general purpose “display” window.

We’ll develop a simple, general mechanism for adding panels to the code

This will shift the work of GUI development to a panel or panels
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){
        p.setPreferredSize(new Dimension(500,400));    c.add(p); }

    public void showFrame(){
        this.pack();
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); }
}
}
Java’s event model
What modern computing is like..

What Scanner-based interactions are like..
(a script)

Your program needs to know:

1) What event to listen for
2) Clarify who’s listening
3) Provide a mechanism for event reaction
The basic script for event handling

- Create components for generating events
- Identify listener for events
- Connect listener, event generator
- Enable listener to listen and act
- Describe actions when events are triggered
import java.awt.*;
import javax.swing.*;
import java.awt.event.*; // needed for event handling

public class WinkPanel extends JPanel implements ActionListener{
    JButton quit = new JButton("Quit");
    JButton wink = new JButton("Wink");
    boolean open = true;

    public WinkPanel(){
        setPreferredSize(new Dimension(400,600));
        this.add(quit);  // place button in panel
        quit.addActionListener(this);
        this.add(wink);  // place button in panel
        wink.addActionListener(this);
    }
}
public void paintComponent(Graphics g){
    super.paintComponent(g);
    if (open){
        g.drawOval(150,150,200,40);
        g.drawOval(230,150,40,40);
        g.fillOval(245,165,10,10); }
    else g.drawOval(150,150,200,40);
}

public void actionPerformed(ActionEvent e){
    if (e.getSource() == quit)
        System.exit(0);
    else
        if(e.getSource() == wink){
            open = !open;
            repaint();
        } }
}
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);
}

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.

An 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();
        BabyControlPanel4 b = new BabyControlPanel4(p);
        display.add(b);
        display.showFrame();
    }
}

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;
    Point[] pts;
}
public BabyControlPanel4(PointPix p){
    model = p;
    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();
        }
    }
}
A square root machine. You enter a (non-negative) real number, and the panel reports a sequence of successive square roots, until you are at 1.0.
Components
Quit button
Roots button
startValue text field
A label requesting the start

Other instance variables
A double value x, to hold successive roots
An int value y
A drop value (could be a constant)
JButton quit = new JButton("Quit");
JButton roots = new JButton("Take roots");
JTextField startVal = new JTextField(15);
JLabel label =
    new JLabel("Enter start value");
double x = 0.0;
int y = 150;
int drop = 20;
The constructor
Must add components in the proper order
What’s the listener?
How to hook up listener with components...
public RootPanel() {

    setPreferredSize(new Dimension(600,600));
    setBackground(Color.green);
    this.add(quit); // place button in panel
    quit.addActionListener(this); // panel is listener
    this.add(label);
    this.add(startVal);
    this.add(roots); // place button in panel
    roots.addActionListener(this); // panel is listener
}
The header line:
How to make it a JPanel?
How to make it capable of listening?
public class RootPanel extends JPanel
    implements ActionListener
How are actions set up and taken?
What are the sources of actions?
And what are the behaviors?
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == quit)
        System.exit(0);
    else if(e.getSource() == roots) {
        x = Double.parseDouble(startVal.getText());
        if (x < 1.0){System.out.println("bad input, try again");}

        else
            repaint();
    }
}
public void paintComponent(Graphics g) {
    y=150;
    super.paintComponent(g);
    while (x > 1.000001){
        x = Math.sqrt(x);
        y = y + drop;
        g.drawString(""+x,100,y);
    }
}