CS 121 – Intro to Programming:Java - Lecture 22

Announcements

Program 5 - Sudoku - due Friday
Program 6 - Scrabble - due Monday
Ch 11 OWL exercises due 11/26
Ch 12 OWL exercises up today

Final Exam: Monday, Dec 17, 8 AM
Graphics and GUIs, or Graphical User Interfaces.

GUI (Graphical User Interface)s are pervasive -

Java gets at GUIs in two ways: via Applets - these are programs that are suitable for transporting over the web, and displaying with a browser;

and application programs, which you run directly (not through a browser).

The two concepts are pretty close. Here we’ll work on the latter.
good day!
import java.awt.*;
import javax.swing.*;

public class BabyGraphics{

    public static void main(String[] args){
        JFrame frame = new JFrame("Starter Work");
        Container c = frame.getContentPane();
        EyePanel p = new EyePanel(Color.green);
        c.add(p);
        frame.pack();
        frame.setVisible(true);
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE);
    }
}

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.*;

public 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); }
}
}
import java.awt.*;

public class FanTester{

    public static void main(String[] args){
        try{
            DisplayWindow display = new DisplayWindow();
            FanPanel w = new FanPanel(Color.green);
            display.add(w);
            display.showFrame();
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}
import java.awt.*;
import javax.swing.*;

public class FanPanel extends JPanel{

    public FanPanel(Color c){
        setPreferredSize(new Dimension(700,700));
        setBackground(c);
    }

    public void paintComponent(Graphics g){
        super.paintComponent(g);
        for( int j = 0; j < 20; j++)
            g.drawLine(0,0,300,j*20);
    }
}

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

public class TryoutPanel extends JPanel{
    private Color myColor;
    public TryoutPanel(Color c){
        myColor = c;
    }
}
public void paintComponent(Graphics g){
    super.paintComponent(g);
    setForeground(myColor);
    g.drawString("top",10,50);
    g.drawLine(10,60, 200,60);
    g.drawString("middle",10,80);
    g.drawLine(10,90, 200,90);
    g.drawString("bottom",10,110);
    g.drawLine(10,120, 200,120);
    g.drawRect(200,300,100,50);
    g.drawOval(200,300,100,50);
    for(int j = 0; j < 9; j++)
        g.drawOval(50,200, 10 + 20*j, 210 - 20*j);
}
public class WindowDriver{
    public static void main(String[] args){
        DisplayWindow d =
                new DisplayWindow();
        TryoutPanel p =
                new TryoutPanel(Color.blue);
        d.addPanel(p);
        d.showFrame();
    }
}
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
import java.awt.*;

public class SimpleButtonDriver{

    public static void main(String[] args){
        DisplayWindow d = new DisplayWindow();
        SimpleControlPanel p =
        new SimpleControlPanel();
        d.addPanel(p);
        d.showFrame();
    }
}
}
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 SimpleControlPanel extends JPanel implements ActionListener{

    JButton quit = new JButton("Quit"); // make button object

    public SimpleControlPanel(){
        setBackground(Color.red);
        this.add(quit); // place button in panel (what's this??)
        quit.addActionListener(this); // panel is listener for button
    }

    public void actionPerformed(ActionEvent e){ // what happens
        if (e.getSource() == quit) System.exit(0); }
}
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();
        }   }
}
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();
        }
}