Graphics and GUIs

CSE219, Computer Science III
Stony Brook University
http://www.cs.stonybrook.edu/~cse219
So where were we?

• We learned to use Swing to:
  • create a window (JFrame)
  • arrange components in our window
  • react to user interactions

• Note: so far, we are only using pre-made components
  • JButton, JTextField, etc.
  • these have pre-programmed options for:
    • rendering
    • behavior
What is next?

- How do we render the regions?
- How do we listen for mouse interactions on regions?

For example: to render and play the Legend of Zelda (Nintendo).
Where will be draw?

- On a JPanel
  - Why?
    - because they’re blank canvases

- To do this:
  - define your own JPanel
  - customize how its rendered
Drawing on a JPanel

- Define a class that extends JPanel
  - include necessary instance variables
    - data about the items to be drawn
    - images, locations, etc.
  - override the `paintComponent` method
    - inherited from `JComponent`
    - do all image (and shape & text) rendering inside this method

- `paintComponent`
  - called on all GUI components
  - we can plug in our own custom rendering
public class MyPanel extends JPanel {
  // INSTANCE VARIABLES CONTAINING // EVERYTHING TO BE RENDERED
  public void paintComponent(Graphics g) {
    // code for drawing will go here
  }  
}
public void paintComponent(Graphics g)

• Never call this method yourself

• You will be tempted to.

• Instead, to redraw your panel
  – call repaint() on your panel object
    • in response to events
    • or in response to a timer (for animation)
An approach to defining `paintComponent`

- **Anticipate changes in what gets rendered**
  - think dynamically

- **Render according to instance variables**

- **Carefully map out your logic**
  - remember: design, design, design
The Graphics class

- Stores settings for rendering:
  - current *Color* to use (text and shapes)
  - current *Font* to use (text)
  - current *Stroke* (line thickness) to use

- Has methods for rendering
  - images, rectangles, ovals, *polygons*, *lines*, and *text*

- Specify all locations and sizes in pixels
  - (0, 0) coordinate is top-left corner of panel
Some Methods of the Graphics Class

• To change rendering settings:
  - setColor
  - setFont
  - setStroke

• To render:
  - drawPolygon
  - drawRect
  - drawImage
  - drawOval
  - drawString
  - drawLine
  - fillPolygon
  - fillRect
  - fillImage
  - fillOval
Colors

• `setColor`
  – select a color used for all subsequent drawing. Ex:

• 13 Standard Colors:
  – black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow
  – Example: `component1.setColor(Color.red);`

• More than 4 billion custom colors:
  – `new Color(int red, int green, int blue)`
  – Red, blue, and green are values from 0-255
  – Example: `component1.setColor(new Color(255, 0, 0));`
Recap: respond to mouse clicks on our panel

- Define our own event handler (a MouseListener)
  - make it an inner class of panel

- Respond inside one of 5 methods:
  - mousePressed
  - mouseReleased
  - mouseClicked
  - mouseEntered
  - mouseExited

- For your panel:
  - declare your handler as an instance variable
  - construct your handler
  - register your handler with the panel
Recap: respond to mouse dragging on our panel

- Define our own event handler (a `MouseMotionListener`)
  - make it an inner class of panel
- Respond inside one of 2 methods:
  - `mouseDragged`
  - `mouseMoved`
- For your panel:
  - declare your handler as an instance variable
  - construct your handler
  - register your handler with the panel
Example: mixing Graphics & Events

- Create a GUI with a panel on which we will render lots of random irregular pentagons
  - random shape and color

- When the user double clicks on the panel
  - will clear panel

- Wherever the user drags the mouse on the panel:
  - more pentagons will be added
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PentaFrame extends JFrame {
    private PentaPanel pentaPanel;

    public PentaFrame() {
        super("Penta Frame");
        setExtendedState(MAXIMIZED_BOTH);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        layoutGUI();
    }

    public void layoutGUI() {
        pentaPanel = new PentaPanel();
        this.add(pentaPanel, BorderLayout.CENTER);
    }

    public static void main(String[] args) {
        PentaFrame frame = new PentaFrame();
        frame.setVisible(true);
    }
}
class PentaPanel extends JPanel {
    // EVENT HANDLERS
    private MouseHandler mouseHandler;
    private MouseMotionHandler mouseMotionHandler;

    // WE'LL USE THESE VECTORS FOR STORING THE VERTICES
    // AND COLORS OF OUR PENTAGONS
    private Vector<int[][]> xPoints;
    private Vector<int[][]> yPoints;
    private Vector<Color> colors;
public PentaPanel()
{
    // AND OUR PENTAGRAM VERTEX ARRAYS
    xPoints = new Vector<int[]>(());
    yPoints = new Vector<int[]>(());
    colors = new Vector<Color>(());

    // LET'S LISTEN FOR MOUSE CLICKS
    // AND DRAGGING ON THE PANEL
    mouseHandler = new MouseHandler();
    this.addMouseListener(mouseHandler);
    mouseMotionHandler = new MouseMotionHandler();
    this.addMouseMotionListener(mouseMotionHandler);
}
public void paintComponent(Graphics g) {
    super.paintComponent(g);

    // DRAW ALL THE PENTAGONS
    for (int i = 0; i < xPoints.size(); i++) {
        int[] xVertices = xPoints.get(i);
        int[] yVertices = yPoints.get(i);
        Color color = colors.get(i);
        g.setColor(color);
        g.fillPolygon(xVertices, yVertices, 5);
        g.setColor(Color.black);
        g.drawPolygon(xVertices, yVertices, 5);
    }
}
class MouseHandler extends MouseAdapter
{
    public void mouseClicked(MouseEvent me)
    {
        if (me.getClickCount() == 2)
        {
            xPoints.clear();
yPoints.clear();
colors.clear();
repaint();
        }
    }
}
class MouseMotionHandler extends MouseMotionAdapter
{
    public void mouseDragged(MouseEvent me)
    {
        // THE LOCATION OF WHERE WE'RE DRAGGING THE MOUSE
        int x = me.getX();
        int y = me.getY();

        // THESE WILL BE THE VERTICES
        int[] xs = new int[5];
        int[] ys = new int[5];

        // TOP CENTER POINT
        xs[0] = x;
        ys[0] = y - (int)(Math.random() * 20) - 1;

        // TOP-RIGHT POINT
        xs[1] = x + (int)(Math.random() * 15) + 1;
        ys[1] = y - (int)(Math.random() * 10) - 1;
    }
}
// BOTTOM-RIGHT POINT
xs[2] = x + (int)(Math.random() * 10) + 1;
ys[2] = y + (int)(Math.random() * 15) + 1;

// BOTTOM-LEFT POINT
xs[3] = x - (int)(Math.random() * 10) - 1;
ys[3] = y + (int)(Math.random() * 15) + 1;

// TOP-LEFT POINT
xs[4] = x - (int)(Math.random() * 15) - 1;
ys[4] = y - (int)(Math.random() * 10) - 1;

xPoints.add(xs);
yPoints.add(ys);
int r = (int)(Math.random() * 256);
int g = (int)(Math.random() * 256);
int b = (int)(Math.random() * 256);
colors.add(new Color(r, g, b));
repaint();
}