Programming Languages and Techniques (CIS120)

Lecture 33
April 19, 2019

Swing I: Drawing and Event Handling
Chapter 29
Announcements

• Java Swing lectures
  – Today: Drawing and event handling
  – Monday 4/22: Inner classes and layout
  – Wednesday 4/24: The “model/view/controller” idiom

• Game Project Complete Code
  – Due: May 1st
    – *NO LATE SUBMISSIONS*
    – *Grading will be by demo. Stay tuned for details!*
Game Project
Swing

Java's GUI library
Have you ever used the Swing library to build a Java app before?

Nope

No, but I've used a different GUI library in Java

Yes, but I didn't really understand how it worked

Yes, I'm an expert
Why study GUIs (yet again)?

- Most common example of event-based programming
- Heavy and effective use of OO inheritance
- Case study in library organization
  - and some advanced Java features
- Ideas applicable everywhere:
  - Web apps
  - Mobile apps
  - Desktop apps
- Fun!
## Terminology overview

<table>
<thead>
<tr>
<th></th>
<th>GUI (OCaml)</th>
<th>Swing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics Context</td>
<td>Gctx.gctx</td>
<td>Graphics</td>
</tr>
<tr>
<td>Widget type</td>
<td>Widget.widget</td>
<td>JComponent</td>
</tr>
<tr>
<td>Basic Widgets</td>
<td>button, label, checkbox</td>
<td>JButton, JLabel, JCheckBox</td>
</tr>
<tr>
<td>Container Widgets</td>
<td>hpair, vpair</td>
<td>JPanel, Layouts</td>
</tr>
<tr>
<td>Events</td>
<td>event</td>
<td>ActionEvent, MouseEvent, KeyEvent</td>
</tr>
<tr>
<td>Event Listener</td>
<td>mouse_listener, mouseclick_listener (any function of type event -&gt; unit)</td>
<td>ActionListener, MouseListener, KeyListener</td>
</tr>
</tbody>
</table>
Swing practicalities

• Java library for GUI development
  – javax.swing.*

• Built on existing library: AWT
  – java.awt.*
  – When there are two versions of something, use Swing’s.
    (e.g., java.awt.Button vs. javax.swing.JButton)
    • The “Jxxx” version is usually the one you want, not plain “xxx”

• Portable
  – Communicates with underlying OS's native window system
  – Same Java program looks appropriately different when run on PC, Linux, and Mac
Simple Drawing

DrawingCanvas.java
DrawingCanvasMain.java
Fractal Drawing Demo
private static void fractal(Graphics gc, int x, int y, double angle, double len) {
    if (len > 1) {
        double af = (angle * Math.PI) / 180.0;
        int nx = x + (int)(len * Math.cos(af));
        int ny = y + (int)(len * Math.sin(af));
        gc.drawLine(x, y, nx, ny);
        fractal(gc, nx, ny, angle + 20, len - 8);
        fractal(gc, nx, ny, angle - 10, len - 8);
    }
}
How do we draw a picture?

• In the OCaml GUI HW, we created widgets whose `repaint` function used the graphics context to draw an image

```ocaml
let w_draw : widget = {
    repaint = (fun (gc:gctx) ->
        fractal (with_color gc green)
            200 450 270 80);
    size = (fun () -> (200,200));
    handle = (fun () -> ());
}
```

• In Swing, the preferred idiom is to `extend` the class JComponent...
Fundamental class: JComponent

- Analog of widget type from OCaml GUI project
  - *(Terminology: widget == JComponent)*
- Subclasses should *override* methods of JComponent
  - `paintComponent` (like repaint, displays the component)
  - `getPreferredSize` (like size, calculates the size of the component)
- Events are handled by listeners (don't need to use overriding...)
- Richer functionality
  - minimum/maximum size
  - font
  - foreground/background color
  - borders
  - what is visible
  - many more...
Simple Drawing Component

```java
public class DrawingCanvas extends JComponent {

    // paint the drawing panel on the screen
    public void paintComponent(Graphics gc) {
        super.paintComponent(gc);

        // set the pen color
        gc.setColor(Color.GREEN);

        // draw a fractal tree
        fractal(gc, 200, 450, 270, 80);
    }

    // give the size of the drawing panel
    public Dimension getPreferredSize() {
        return new Dimension(200, 200);
    }
}
```

How do we put this component on the screen?
**JFrame**

- Represents a top-level window
  - Displayed directly by OS (looks different on Mac, PC, etc.)
- Contains JComponents
- Can be moved, resized, iconified, closed

```java
public void run() {
    JFrame frame = new JFrame("Tree");

    // set the content of the window to be our drawing
    frame.getContentPane().add(new DrawingCanvas());

    // make sure the application exits when the frame closes
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // resize the frame based on the size of the panel
    frame.pack();

    // show the frame
    frame.setVisible(true);
}
```
User Interaction
Task: Program an application that displays a button. When the button is pressed, it toggles a “lightbulb” on and off.

Key idea: use a ButtonListener to toggle the state of the "lightbulb"
OnOffDemo

The Lightswitch GUI program in Swing.
class LightBulb extends JComponent {
    private boolean isOn = false;

    public void flip() {
        isOn = !isOn;
    }

    public void paintComponent(Graphics gc) {
        if (isOn) {
            gc.setColor(Color.YELLOW);
        } else {
            gc.setColor(Color.BLACK);
        }
        gc.fillRect(0, 0, 100, 100);
    }

    public Dimension getPreferredSize() {
        return new Dimension(100, 100);
    }
}

Remember the private state of the lightbulb

Draw the Light bulb here using the graphics context

Set the size of the window
Main Class

public class OnOff implements Runnable {
    public void run() {
        JFrame frame = new JFrame("On/Off Switch");
        JPanel panel = new JPanel();
        frame.getContentPane().add(panel);
        LightBulb bulb = new LightBulb();
        panel.add(bulb);
        JButton button = new JButton("On/Off");
        panel.add(button);
        button.addActionListener(new ButtonListener(bulb));
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new OnOff());
    }
}
class ButtonListener implements ActionListener {
    private LightBulb bulb;
    public ButtonListener (LightBulb b) {
        bulb = b;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        bulb.flip();
        bulb.repaint();
    }
}

Note that “repaint” does not necessarily do any repainting now! It is simply a notification to Swing that something needs repainting.