Programming Languages and Techniques (CIS1200)

Lecture 32

Swing I: Drawing and Event Handling

Chapter 29

Announcements

- HW08: ChatterBot
  - Available now; due on Thursday, April 17th
  - Practice with I/O and Collections

- Game Design Proposal Milestone Due: (8 points)
  Tuesday, April 15th at Midnight = 11:59PM!
  - (Should take about 1 hour)
  - Submit on GRADESCOPE
  - TAs will give you feedback

Announcements

- TA position applications are available
  - CIS 1100, 1200, 1600, 1210 (see link on Ed)
  - Other CIS classes (see https://www.cis.upenn.edu/ta-information/)
  - Accepting applications until Friday, April 18th
  - Intro CIS TA Panel: Tonight! April 14th 7-8:30pm, Berger Auditorium

- Guest Lecturer (Dr. Zdancewic) Wednesday and Friday
  - Recordings / slides may be delayed
  - Wednesday: “Swing II: Inner Classes and Layout”
  - Friday: “Code *is* Data”

Swing

Java's GUI library

33: Have you ever used the Swing library to build a Java app before?

0

Nope

0%

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

0%

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

0%

Yes, I'm an expert

0%

Why study GUIs (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

	GUI Library (OCaml)	Swing Classes (Java)
Graphics Context	`Gctx.gctx`	`Graphics, Graphics2D`
Widget type	`Widget.widget`	`JComponent`

Basic Widgets	`button`
	`label`
	`checkbox`
		`JButton`
		`JLabel`
		`JCheckBox`

Container Widgets	`hpair, vpair`	`JPanel, Layouts`

Events	`event`	`ActionEvent`
		`MouseEvent`
		`KeyEvent`

Event Listener	`mouse_listener`
	`mouseclick_listener`
	(functions of type `event -> unit`)
		`ActionListener`
		`MouseListener`
		`KeyListener`

Swing practicalities

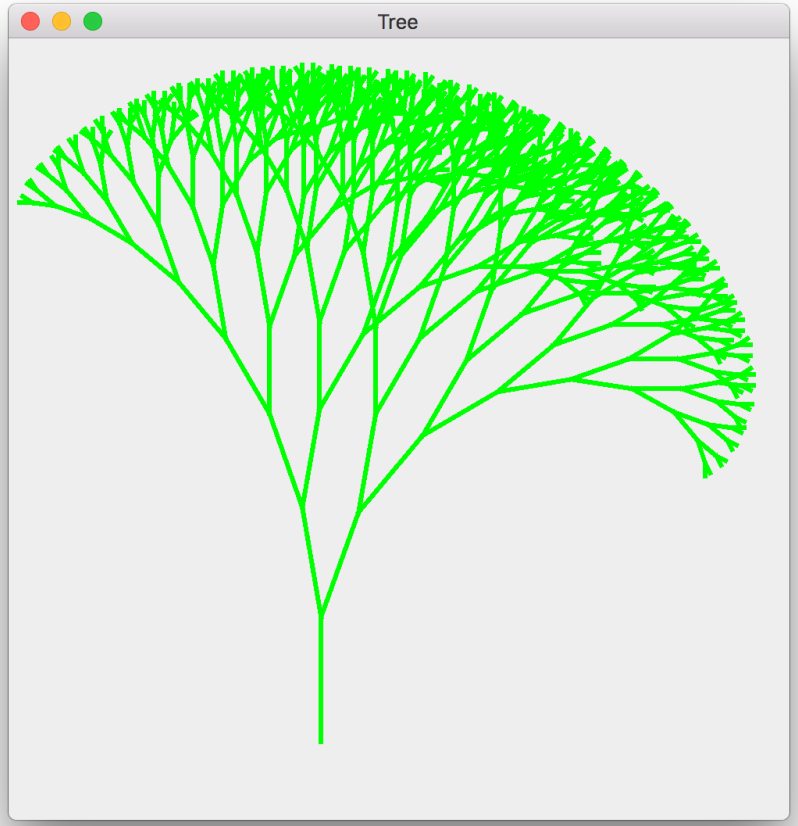
- Java library for GUI development
  - javax.swing.*
- Built on older library: AWT
  - java.awt.*
  - When there are two versions of something, use Swing's. (e.g., javax.swing.JButton instead of java.awt.Button)
    - The "JFoo" version is usually the one you want, not plain "Foo"
- Portable
  - Communicates with underlying OS's native window system
  - Same Java program looks appropriately different when run in the browser and on PC, Linux, Mac, etc.

Simple Drawing

DrawingCanvas.java

DrawingCanvasMain.java

Fractal Drawing Demo



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

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

   size    = (fun () -> (200,200));

   handle  = (fun () -> ())
}
```

OCaml

- In Swing, the preferred idiom is to *extend* the JComponent class …

Fundamental Swing Class: JComponent

- Analog of *widget type* from OCaml GUI project
- 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*
  - no need for overriding here
- Rich functionality
  - minimum/maximum size
  - font
  - foreground/background color
  - borders
  - visibility
  - much more…

Step 1: Recursive function for drawing

```
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 turn this into a GUI component?

Step 2: Simple Drawing Component

```
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);
        ((Graphics2d)gc).setStroke(new BasicStroke(3));

        // 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?

Step 3: JFrame

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

```
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);
}
```

(Plus a bit of boilerplate to call run from main…)

Swing User Interaction

Start Simple: Light Switch

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 Lightbulb GUI program in Swing.

Display the Lightbulb

```
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());
  }
}
```

Open frame and make a panel

Create bulb and button

Start the (Swing) application

Making the Button Do Something

```
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 right now! It is simply a notification to Swing that something needs repainting. (This is a difference from our OCaml GUI library.) But it is required.

An Awkward Comparison

```java
class ButtonListener implements ActionListener {
    private LightBulb bulb;
    public ButtonListener (LightBulb b) {
        bulb = b;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        bulb.flip();
        bulb.repaint();
    }
}

// somewhere in run …
LightBulb bulb = new LightBulb();
JButton button = new JButton("On/Off");
button.addActionListener(new ButtonListener(bulb));
```

Java

```ocaml
let bulb, bulb_flip = make_bulb ()
let onoff,_, bnc = button "On/Off"
;; bnc.add_event_listener (mouseclick_listener bulb_flip)
```

OCaml

Too much “boilerplate”!

- ButtonListener really only needs to do bulb.flip() and repaint
- But we need all this extra boilerplate code to build the class
- Often we will instantiate a given Listener class in a GUI *exactly one time*

```
class ButtonListener implements ActionListener {
    private LightBulb bulb;
    public ButtonListener (LightBulb b) {
        bulb = b;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        bulb.flip();
        bulb.repaint();
    }
}
```

This is a job for…