Programming Languages and Techniques (CIS120)

Lecture 33

Swing I: Drawing and Event Handling

Chapter 29

Announcements

- HW8: TwitterBot
  - Available on the web site
  - Due: Tuesday, November 26th at 11:59pm
  - This is a *new* project (replacing SpellChecker), so please start early!

- HW9a: (see next slide)
  - Due this Friday!

- Regrade requests for Midterm 2 due by Friday.

HW9: Game project

- Game Design Proposal Milestone Due: (8 points)
  Friday November 22nd at NOON = 11:59AM!!!!
  - (Should take about 1 hour)
  - Submit on GRADESCOPE
  - TAs will give you feedback over the weekend

- Final Program Due: (92 points)
  Monday, December 9th at 11:59pm
  - Submit zipfile online, submission *only* checks if your code compiles
  - Eclipse is STRONGLY recommended for this project
  - May distribute your game (after the deadline) if you do not use any of our code
- Grade based on demo with your TA during reading days
  - Grading rubric on the assignment website
  - Recommendation: don't be too ambitious.

- *NO LATE SUBMISSIONS PERMITTED*

Coding: Histogram.java

WordScanner.java

Histogram.java

Histogram Example Key Ideas

- Implementing an Iterator<T>
  - maintain a "cursor" and some invariants that relate hasNext() and next()

- FileReader (and FileWriter) are easy to use.
  - often want BufferedReader / BufferedWriter for "line-at-a-time" access and better performance
  - need to use appropriate exception handling to deal with IOExceptions

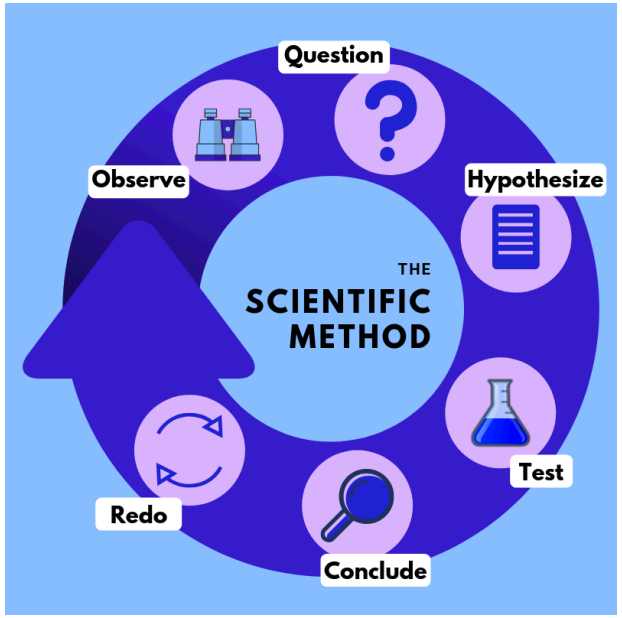
- Histogram is just a Map<String,Integer> object
  - see also: ProbabilityDistribution in TwitterBot HW

- Easy to write "command line" (a.k.a. terminal apps)
  - use the args[] input to main for inputs
  - write to System.out for output
  - run by doing: "java –cp <bindirc> ClassName"

Some Advice on Debugging

Use the Scientific Method

1. Make an observation / ask a question
   - One of my test cases fails!
   - Which assertion? What exception? What is the stack trace?
2. Formulate a hypothesis
   - Could I have passed null as bar to foo.munge(bar)?
3. Conduct an experiment
   - Modify the program to try to confirm or refute the hypothesis.
   - *Don't* make random changes!
   - Predict the outcome of your experiment
   - Re-run test cases, or execute the program
4. Analyze the results
   - Did the modified code behave as expected?
5. Draw conclusions / Report results
   - Create a new test case (if appropriate)



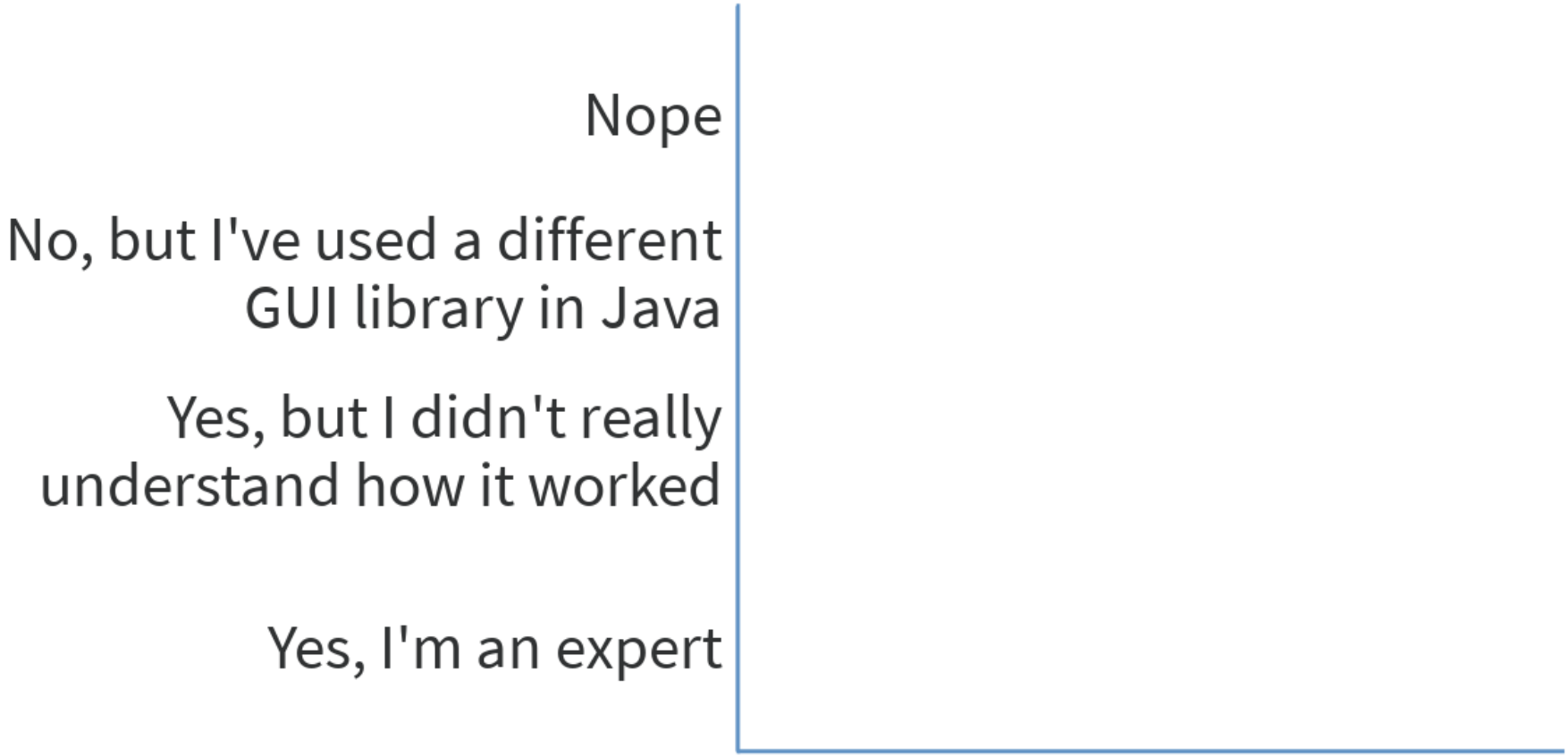
Observing Behavior

- Understand exceptions and their stack traces
  - They give you a lot of information
- If you are using Eclipse, it is worth taking a little time to learn how to use the debugger!
  - See Piazza for a Quick Start tutorial
- Simple print statements are also very effective!
  - Confirm or disprove hypothesis
  - e.g.: The code reached "HERE!" (or not)

Swing

Java's GUI library

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



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

	GUI (OCaml)	Swing
Graphics Context	`Gctx.gctx`	`Graphics`
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`
	(any function of type `event -> unit`)
		`ActionListener`
		`MouseListener`
		`KeyListener`

Swing practicalities

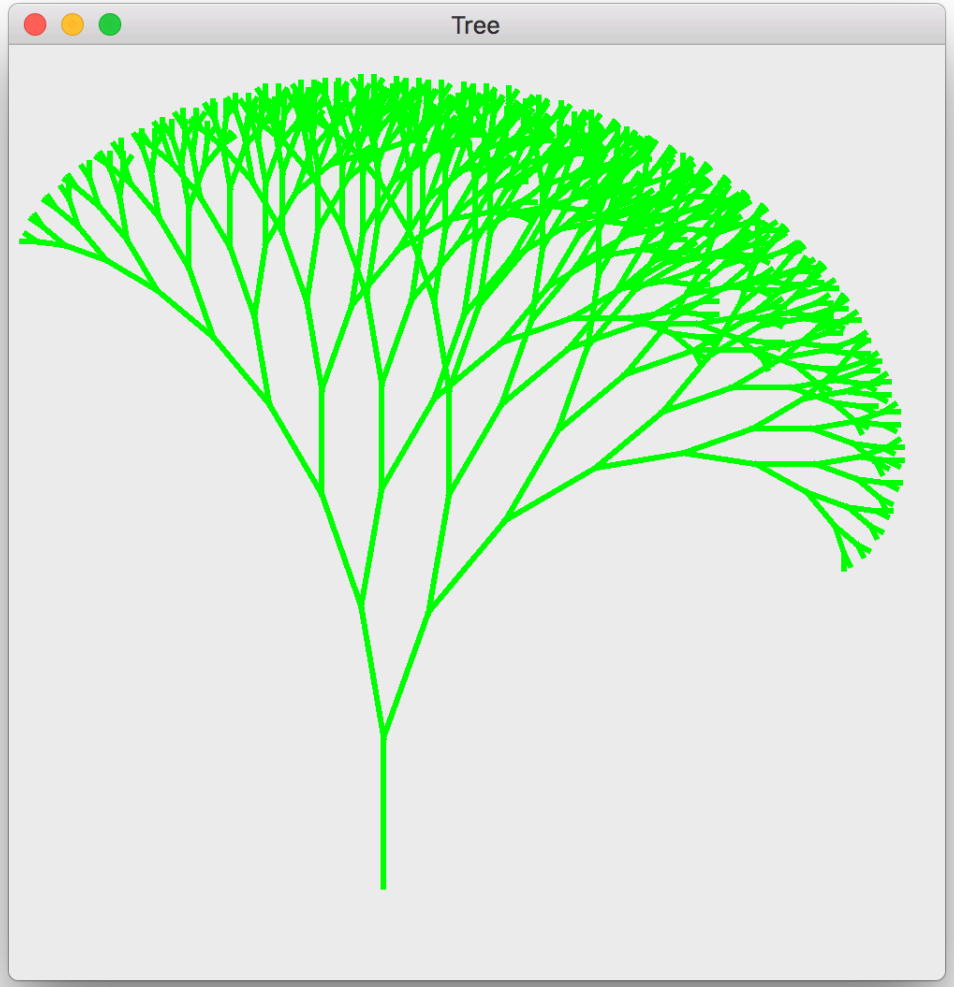
- Java library for GUI development
  - javax.swing.*
- Built on existing library: AWT
  - java.awt.*
  - When there are two versions of something, use Swing's. (e.g., java.awt.Button vs. javax.swing.JButton)
    - The "JFoo" version is usually the one you want, not plain "Foo"
- Portable
  - Communicates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

Simple Drawing

DrawingCanvas.java

DrawingCanvasMain.java

Fractal Drawing Demo



Recursive function for drawing

```
private static void fractal(Graphics2D 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.setStroke(new BasicStroke(3));
        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

```
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 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

```
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((Graphics2d)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

```
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

Start Simple: Lightswitch

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.

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 now! It is simply a notification to Swing that something needs repainting.