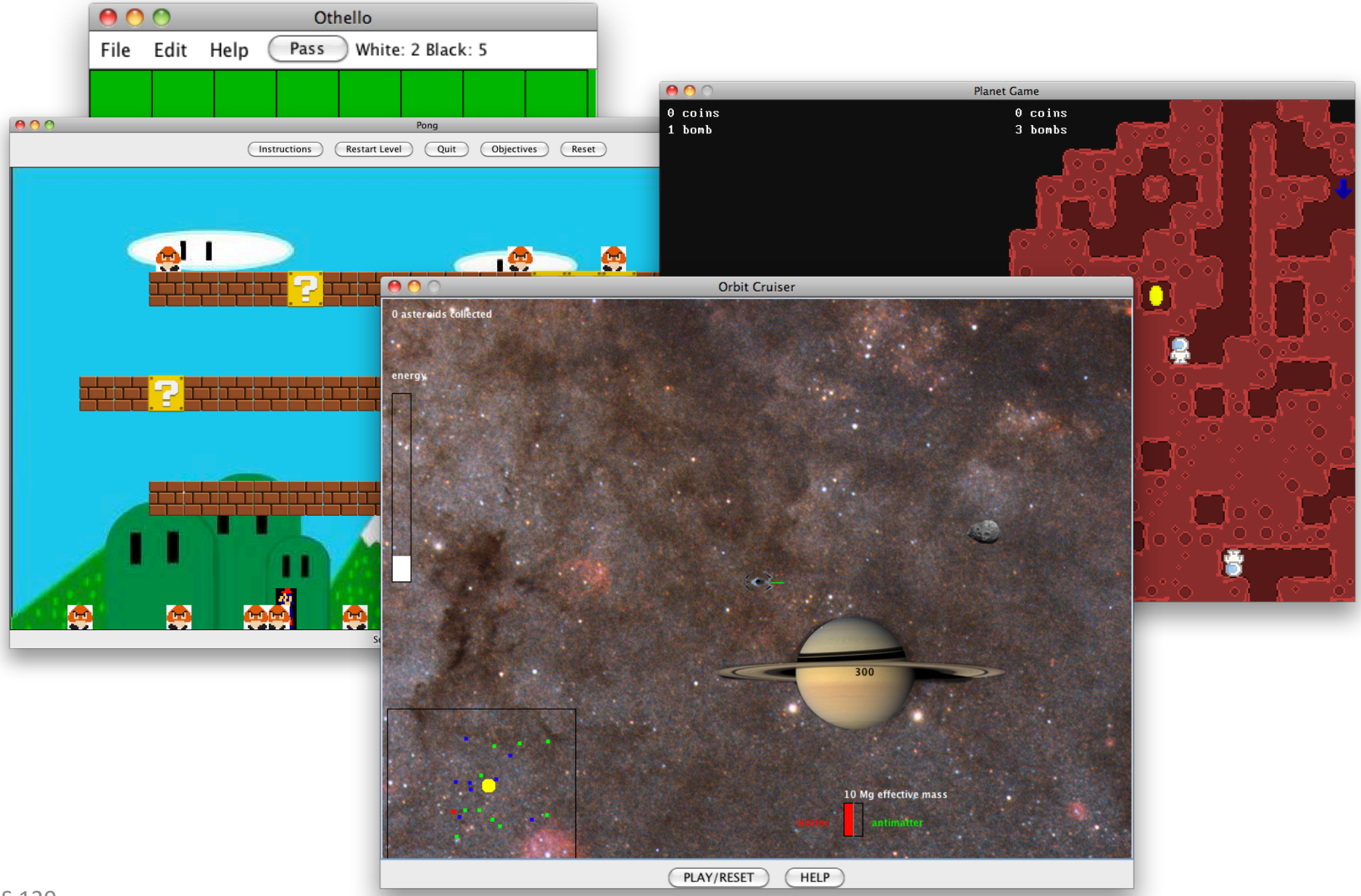
# Programming Languages and Techniques (CIS120)

Lecture 34

## Swing I: Drawing and Event Handling

Chapter 29

# HW9: Game Project



# Swing

Java's GUI library

# Quiz

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

1. Nope
2. No, but I've used a different GUI library in Java
3. Yes, but I didn't really understand how it worked
4. 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

| | GUI (OCaml) | Swing |
|---|---|---|
| Graphics Context | `Gctx.gctx` | `Graphics` |
| Widget type | `Widget.widget` | `JComponent` |
| Basic Widgets | `button`<br>`label`<br>`checkbox` | `JButton`<br>`JLabel`<br>`JCheckBox` |
| Container Widgets | `hpair, vpair` | `JPanel, Layouts` |
| Events | `event` | `ActionEvent`<br>`MouseEvent`<br>`KeyEvent` |
| Event Listener | `mouse_listener`<br>`mouseclick_listener`<br>(any function of type `event -> unit`) | `ActionListener`<br>`MouseListener`<br>`KeyListener` |

# Swing practicalities

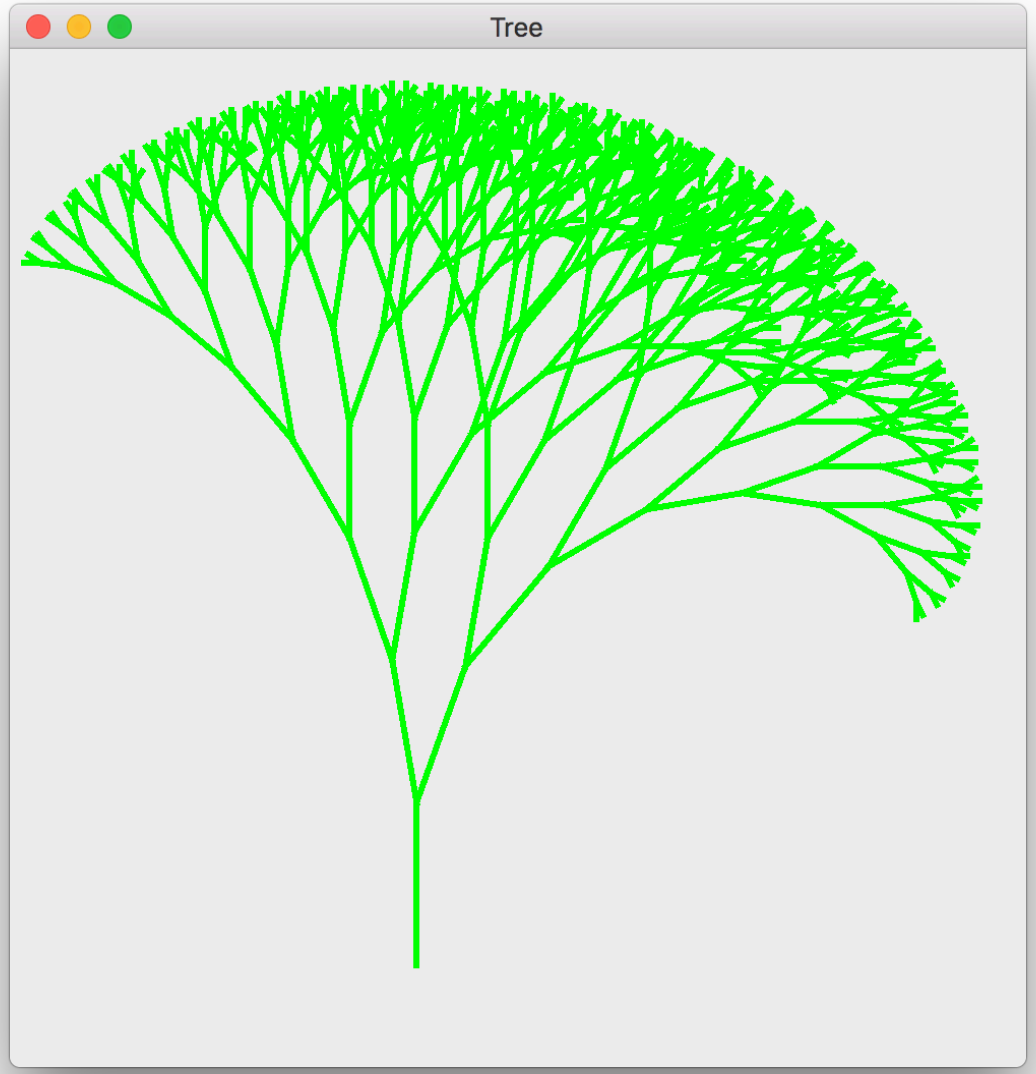
- Java library for GUI development
  - javax.swing.*
- Built on existing library: AWT
  - java.awt.*
  - When there are two versions of something, use Swing's. (e.g., java.awt.Button vs. javax.swing.JButton)
    - The "JFoo" version is usually the one you want, not plain "Foo"
- Portable
  - Communicates 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



# 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…

# 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);
    }
}
```

# 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

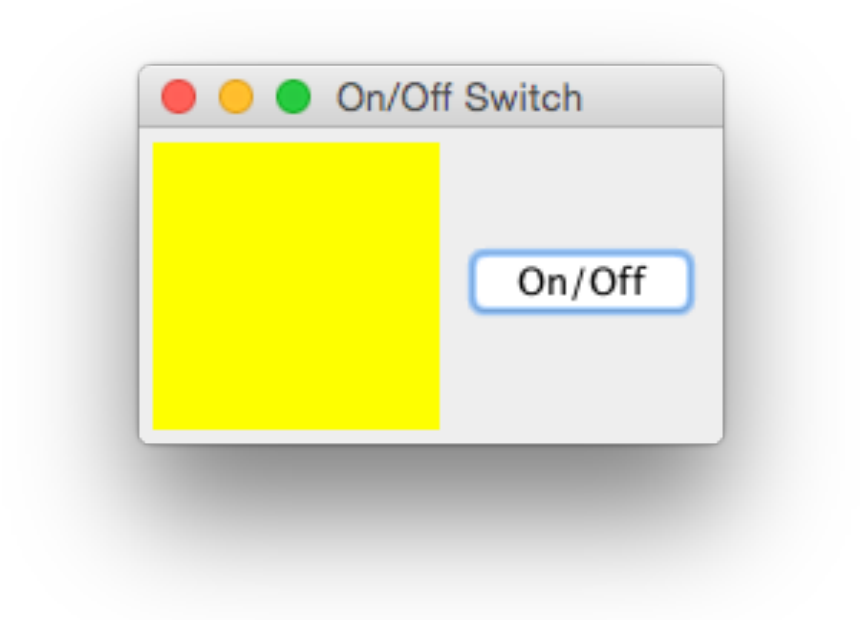
- Represents a top-level window
  - Displayed directly by OS (looks different on Mac, PC, etc.)
- Contains JComponents
- Can be moved, resized, iconified, closed

```
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.