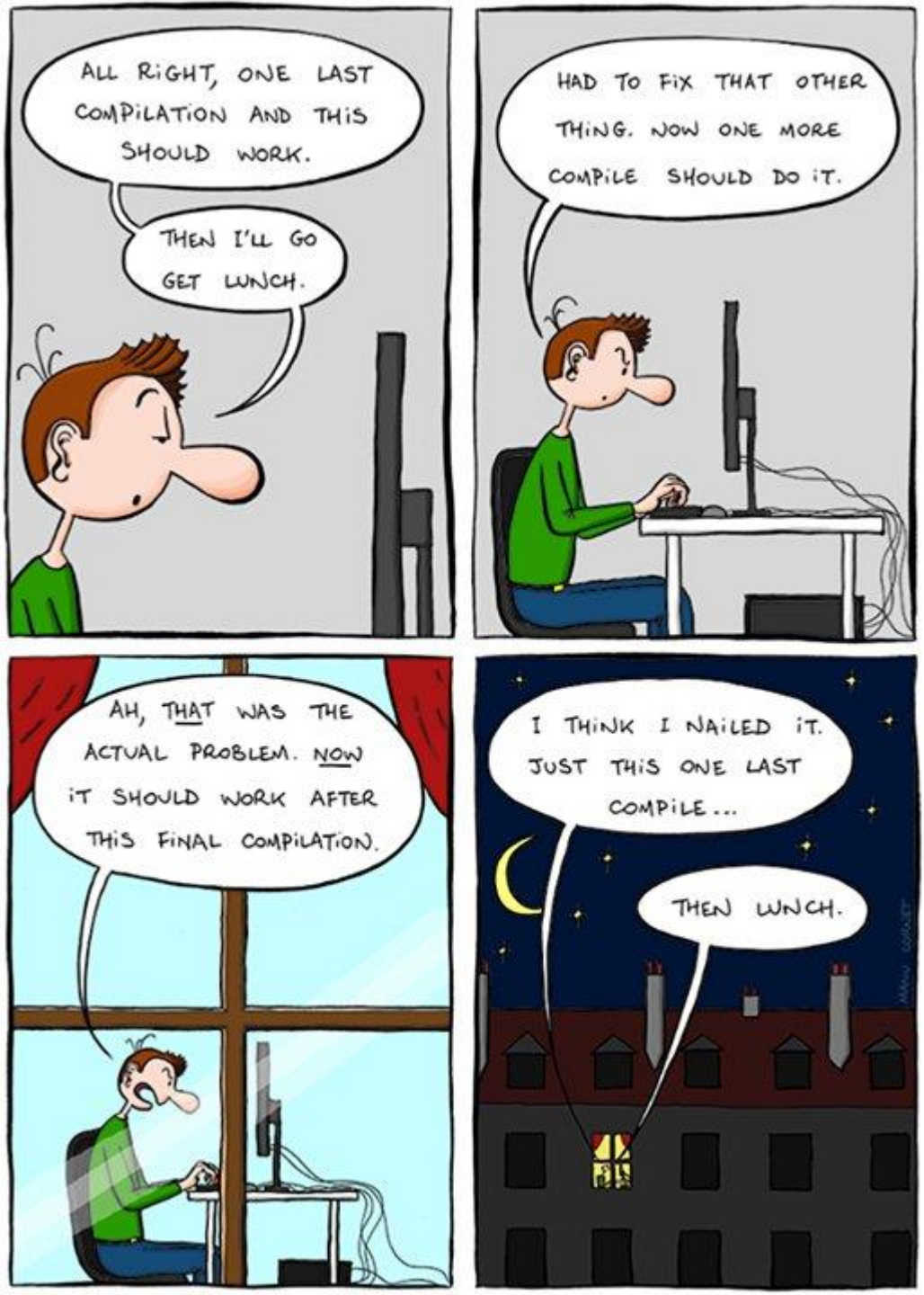
CS 142

Lecture 25: GUIs

Thanks to Marty Stepp and Stuart Reges for parts of these slides



Why GUIs

GUI: Graphical User Interface

- write programs where the user doesn't have to interact with the console
- more practice using built in objects
- Try event-driven programming in Java

History of GUIs in Java

- **Abstract Windowing Toolkit** (**AWT**): *(JDK 1.0 - 1.1)*
  - Maps general Java code to each operating system's real GUI system.
  - *Problems:* Limited to lowest common denominator; clunky to use.
- **Swing**: *(JDK 1.2+)*
  - Paints GUI controls itself pixel-by-pixel rather than handing off to OS.
  - *Benefits:* Features; compatibility; OO design.
- **Java FX**: *(JDK 7.6+)*
  - Adds many features for more sophisticated interfaces including CSS.

GUI Parts

- **window**: A first-class citizen of the graphical desktop.
  - Also called a *top-level container*.
  - examples: frame, dialog box, applet

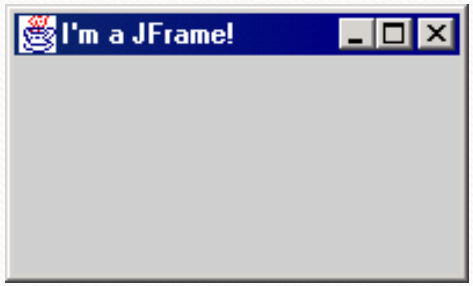
- **component**: A GUI widget that resides in a window.
  - Also called *controls* in many other languages.
  - examples: button, text box, label

- **container**: A logical grouping for storing components.
  - examples: panel, box

```
import javax.swing.*;
```

JFrame

A frame is a graphical window that can be used to hold other components



- `public JFrame()`
  `public JFrame(String title)`
  Creates a frame with an optional title.

- `public void setTitle(String text)`
  Puts the given text in the frame's title bar.

- `public void setDefaultCloseOperation(int op)`
  Makes the frame perform the given action when it closes. Common value:
  `JFrame.EXIT_ON_CLOSE`

- `public void add(Component comp)`
  Places the given component or container inside the frame.
  - *How would we add more than one component to the frame?*

- *NOTE*: Call `setVisible(true)` to make a frame appear on screen after creating it.

JFrame Properties

- JFrames have the following unique properties that you can get or set in your graphical programs:

name	type	description	methods
default close operation	int	what should happen when frame is closed	getDefaultCloseOperation, setDefaultCloseOperation
icon image	Image	icon in the window's title bar	getIconImage, setIconImage
layout	LayoutManager	how the frame should position its components	getLayout, setLayout
resizable	boolean	whether the window can be resized	isResizable, setResizable
title	String	window's title bar text	getTitle, setTitle

Component Properties

- All components also have the following properties:

name	type	description	methods
background	`Color`	background color	`getBackground`, `setBackground`
enabled	`boolean`	whether the component can be interacted with	`isEnabled`, `setEnabled`
font	`Font`	font used to display any text on the component	`getFont`, `setFont`
foreground	`Color`	foreground color	`getForeground`, `setForeground`
location	`Point`	(x, y) position of component on screen	`getLocation`, `setLocation`
size	`Dimension`	width, height of component	`getSize`, `setSize`
preferred size	`Dimension`	width, height that the component wants to be	`getPreferredSize`, `setPreferredSize`
visible	`boolean`	whether the component can be seen on screen	`isVisible`, `setVisible`

JButton and JLabel

The most common component—a button is a clickable onscreen region that the user interacts with to perform a single command

A text label is simply a string of text displayed on screen in a graphical program. Labels often give information or describe other components

- `public JButton(String text)`
  `public JLabel(String text)`
  Creates a new button / label with the given string as its text.

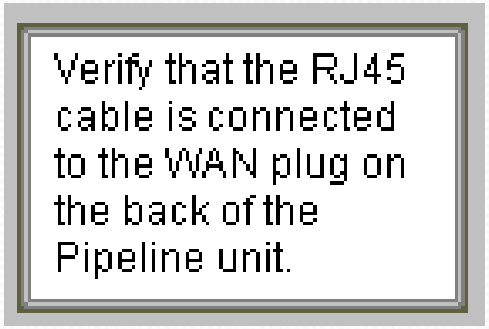
- `public String getText()`
  Returns the text showing on the button / label.

- `public void setText(String text)`
  Sets button / label's text to be the given string.

JTextField and JTextArea

A text field is like a label, except that the text in it can be edited and modified by the user. Text fields are commonly used for user input, where the user types information in the field and the program reads it

A text area is a multi-line text field



- `public JTextField(int columns)`
- `public JTextArea(int lines, int columns)`
  Creates a new text field that is the given number of columns (letters) wide.

- `public String getText()`
  Returns the text currently in the field.

- `public void setText(String text)`
  Sets field's text to be the given string.

Components Example

- The following program attempts to show two buttons:

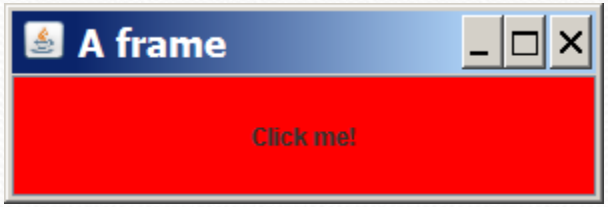
```
import java.awt.*;
import javax.swing.*;

public class ComponentsExample1 {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(new Dimension(300, 100));
        frame.setTitle("A frame");

        JButton button1 = new JButton();
        button1.setText("I'm a button.");
        button1.setBackground(Color.BLUE);
        frame.add(button1);

        JButton button2 = new JButton();
        button2.setText("Click me!");
        button2.setBackground(Color.RED);
        frame.add(button2);

        frame.setVisible(true);
    }
}
```



Layout Problems

- The preceding program added two buttons to the frame, but only one appeared on the screen.

- **layout manager**: An object contained inside frames and other graphical containers that decides the position and size of the components inside the container.

- The default layout manager sizes each component added to occupy the full window space.

Changing Layouts

- We can correct the program's appearance by changing the frame's layout manager.

- Change the layout by calling the `setLayout` method on the frame and passing a layout manager object.
  - We will see several layout managers later.
  - We'll use one called a `FlowLayout`, which sizes each component to its preferred size and positions them in left-to-right rows.

  - If the following line is added to the preceding program just before calling `setVisible(true)`, its appearance will be:

```
frame.setLayout(new FlowLayout());
```

Event-driven Programming

Event-driven Programming

- program's execution is indeterminate
- on-screen components cause *events* to occur when they are clicked / interacted with
- events can be handled, causing the program to respond, *driving* the execution thru events (an "event-driven" program)

Action events: `ActionEvent`

- most common / simple event type in Swing
- represent an action occurring on a GUI component
- created by:
  - button clicks
  - check box checking / unchecking
  - menu clicks
  - pressing Enter in a text field
  - etc.

Listening for Events

- attach *listener* to component
- listener's appropriate method will be called when event occurs (e.g. when the button is clicked)
- for Action events, use `ActionListener`

Writing an `ActionListener`

```
// part of Java; you don't write this
public interface ActionListener {
    public void actionPerformed(ActionEvent event);
}

// Prints a message when the button is clicked.
public class MyActionListener
        implements ActionListener {

    public void actionPerformed(ActionEvent event){
        JOptionPane.showMessageDialog(null,
                "An event occurred!");
    }
}
```

Attaching an `ActionListener`

```
JButton button = new JButton("button 1");
ActionListener listener = new MyActionListener();
button.addActionListener(listener);
```

- now `button` will print `"Event occurred!"` when clicked
  - `addActionListener` method exists in many Swing components