Lecture 7

GUI Applications

Announcements for This Lecture

The Exam

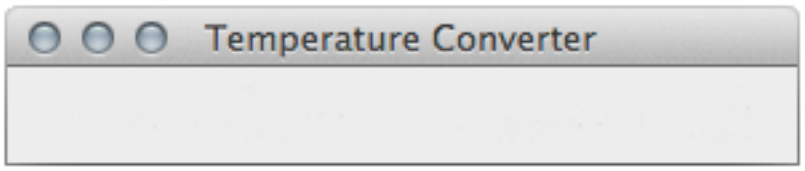
- There is no Exam!

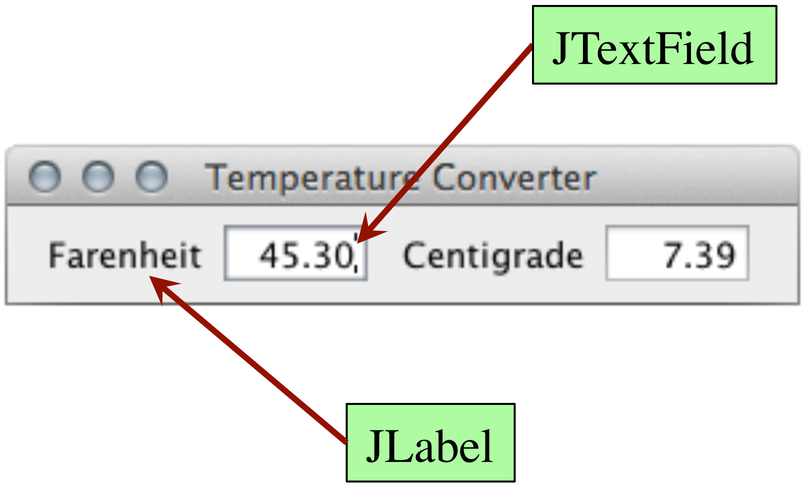
Assignment 3

- Get started on JMan!
  - Want first attempt next Wed
  - MUST submit something
  - Will grade before Fall Break
- Work until 85% correct
  - Do not need a perfect
  - But do not get infinite retries
  - Only three retries after first
  - Retry deadlines posted later

The Limitations of JFrame

- JFrame is just a Window
  - Can resize it
  - Can close it
  - Not much else
- To do more, you need **GUI components**
  - Items inside a JFrame
  - Ex: Buttons, Text Boxes
- Two main Java packages
  - java.awt: “old GUI”
  - javax.swing: “Swing GUI”





AWT vs. Swing

Abstract Window Toolkit

- Uses the standard interface
  - Mac looks like Mac
  - Windows like Windows
- Violates Java "portability"
  - **Demo**: AWTFile.java
- Very rarely used today
  - Handling input is messy
  - But superclass of Swing classes, so have to include

Swing API

- Codename that "stuck"
- Has pluggable look & feel
  - Mac can look like Windows
  - Default same on all platforms
  - Demo: SwingFile.java
- Now the default component collection in Java
  - Very easy to use
  - Programmers like uniformity

Swing Components

`JButton`: a pushbutton that can be clicked by mouse

`JCheckbox`: can be on (true) or off (false)

`JComboBox`: a popup menu of user choices

`JLabel`: a text label

`JList`: scrolling list of user-chooseable items

`JScrollbar`: a scroll bar

`JTextField`: allows editing of a single line of text

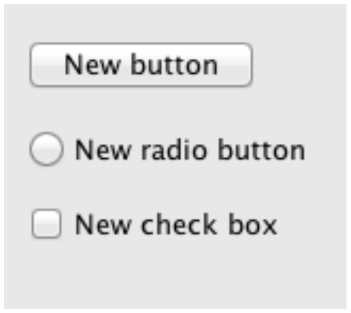
`JTextArea`: multiline region for displaying and editing text

`JPanel`: used for containing and grouping components

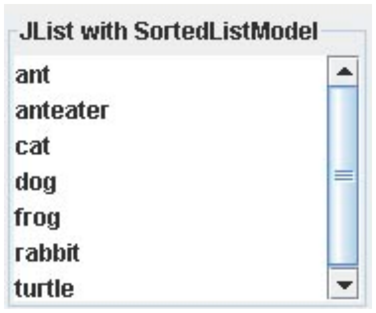
`JDialog`: window used for user input

`JFrame`: top-level window with frame and border

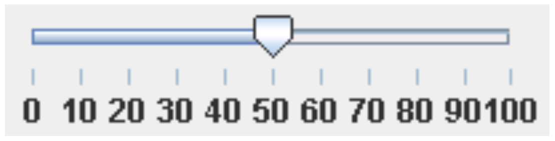
…



Buttons



List

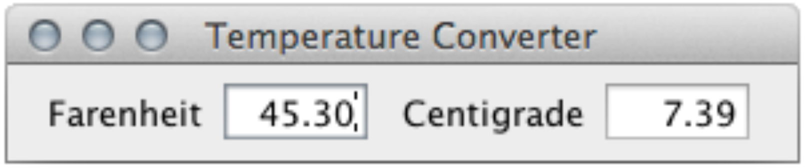


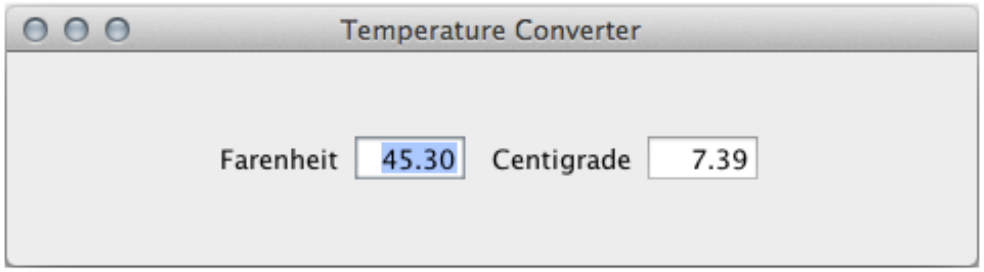
Slider

Main Challenges in GUI Applications

Layout

- Arranging items the screen
  - Java has many components
  - But where do they go?
- **Challenge**: Resizing
  - Want components to "behave nicely" as you resize
  - Change size of components
  - Change padding in between
- LayoutManagers do both





Main Challenges in GUI Applications

Layout

- Arranging items the screen
  - Java has many components
  - But where do they go?
- **Challenge**: Resizing
  - Want components to "behave nicely" as you resize
  - Change size of components
  - Change padding in between
- LayoutManagers do both

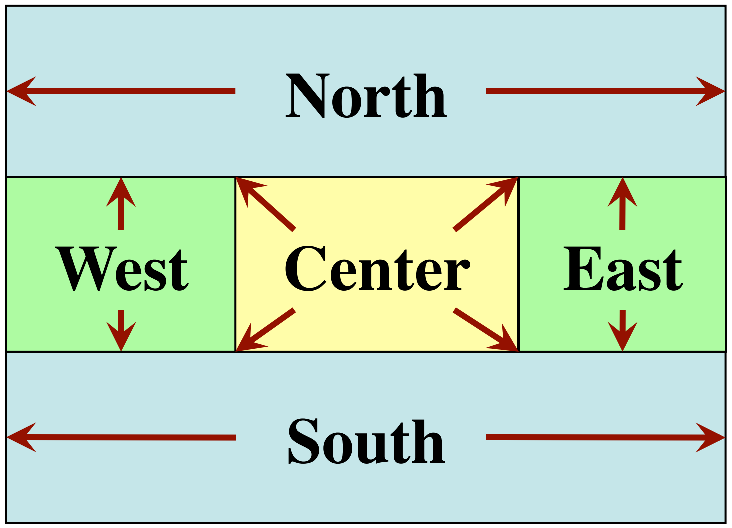
Input Handling

- Many types of input
  - button pushed
  - text typed
  - mouse clicked …
- Want app to react to input
  - Otherwise GUI looks pretty, but does nothing
- **Main focus of GUI code**

There are a Lot of Different Layouts

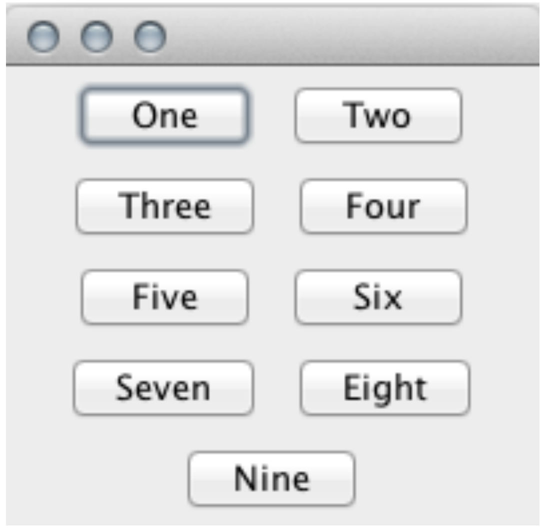
BorderLayout

- Container has 5 directions
  - When add, specify direction
  - **Demo**: TestBorder.java



FlowLayout

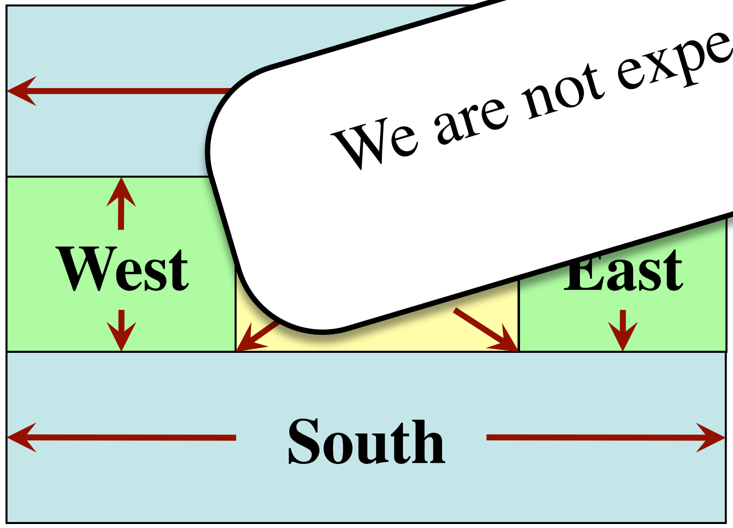
- Use a left-to-right "flow"
  - If row fills, start on next row
  - **Demo**: TestFlow.java



There are a Lot of Different Layouts

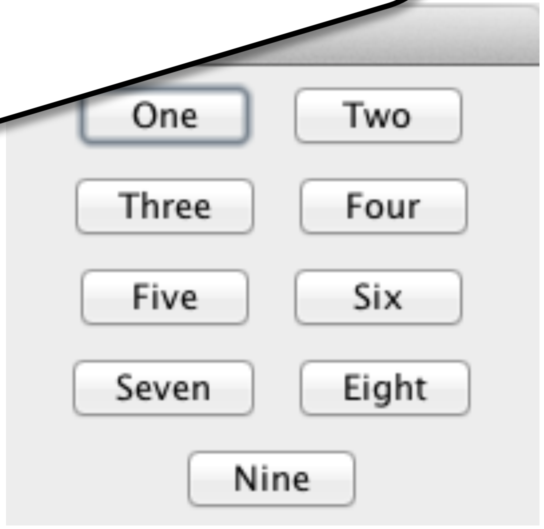
BorderLayout

- Container has 5 directions
  - When add, specify direction
  - **Demo**: TestBorder.java



FlowLayout

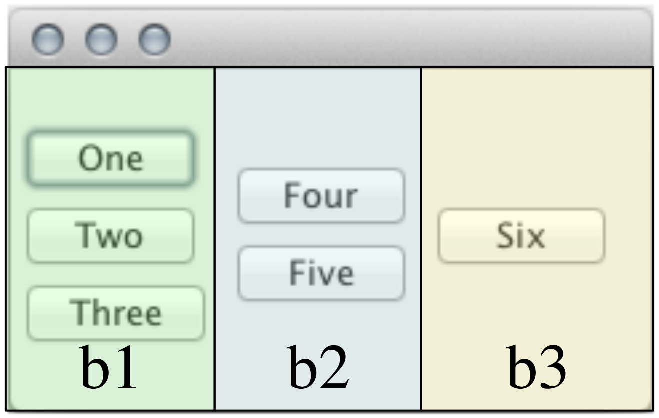
- Use a left-to "flow"
  - next row
  - va



We are not expecting you to master this.

BoxLayout: The Best for Beginners

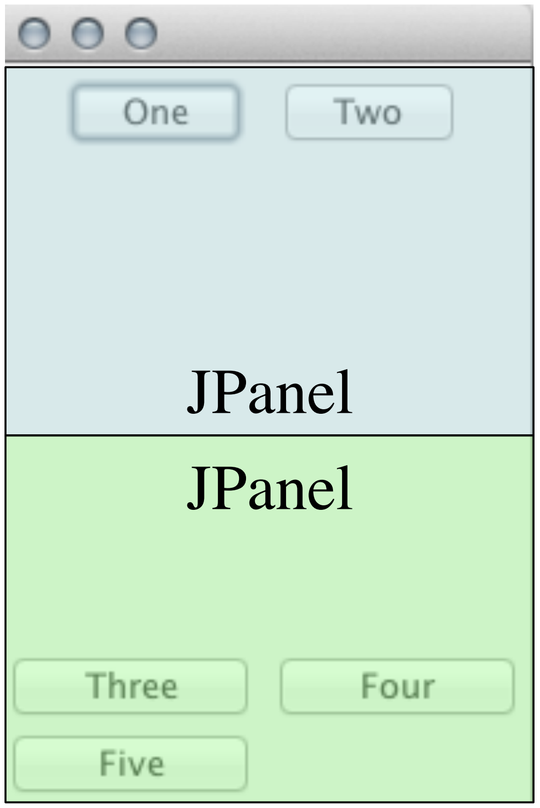
- BoxLayout
  - Arranges components in line
  - No wrap (like FlowLayout)
  - Either horizontal/vertical
- Box: JPanel w/ BoxLayout
  - Box b1= **new** Box(BoxLayout.Y_AXIS);
  - Makes layout quick
- **Demo**: BoxGrouping.java



- Nested boxes
  - Three vertical boxes
  - Inside horizontal box

Nesting Layouts

- Want more interesting layouts
  - **Idea**: nest layouts in each other
  - Can get fine padding control
- Useful class: JPanel
  - Invisible component
  - Container for other components
  - Can take a LayoutManager
- **Demo**: PanelGrouping.java



Main Challenges in GUI Applications

Layout

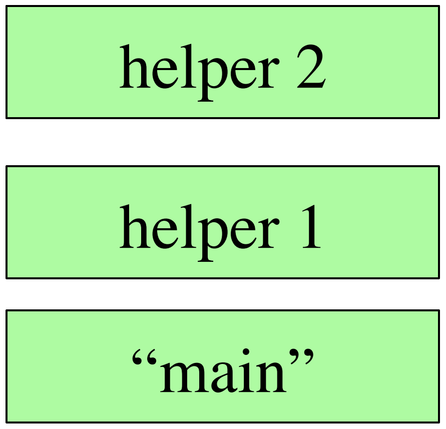
- Arranging items the screen
  - Java has many components
  - But where do they go?
- **Challenge**: Resizing
  - Want components to "behave nicely" as you resize
  - Change size of components
  - Change padding in between
- LayoutManagers do both

Input Handling

- Many types of input
  - button pushed
  - text typed
  - mouse clicked …
- Want app to react to input
  - Otherwise GUI looks pretty, but does nothing
- **Main focus of GUI code**

Traditional Programming

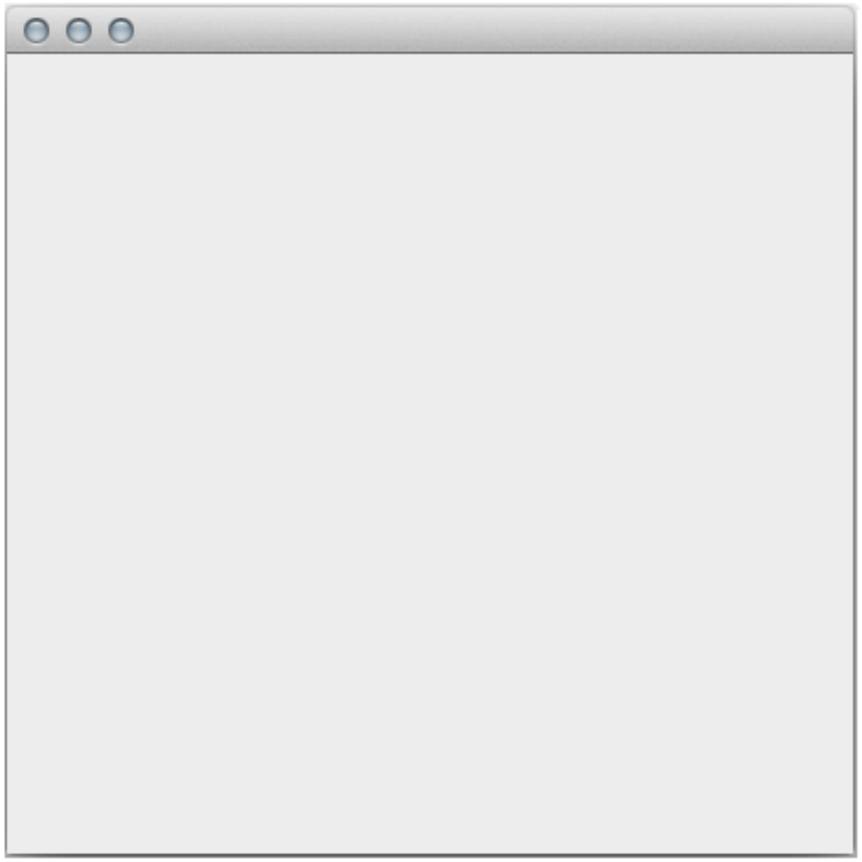
- Have a "main" method
  - Call in Interactions pane
  - Call in JUnit test
  - …somewhere else?
- Other methods are helper methods to "main" one
- Big reason for DrJava
  - Usually only one "main"
  - Interactions pane allows all methods to be "main"



Program ends when "main" is done

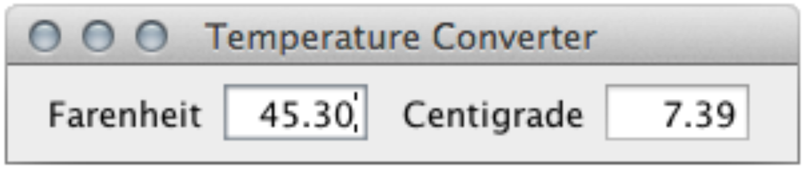
JFrame is Different

- Compile Demo.java
- Type in Interactions pane:
  - Demo.createFrame()
- What happens?
  - Method completes (Interactions pane is free)
  - But the program still runs (JFrame is present)
- Close window to stop



The Event Loop

- Instantiating a JFrame creates an "event loop"
  - Runs until window closed
  - Body checks for user input
  - Input generates "events"
- Events are objects
  - Hold input information
  - Mouse location clicked
  - Key typed
- But what to do with events?



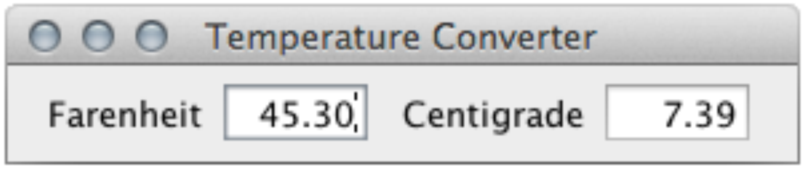
Starts

```
while ( JFrame is showing ) {
   Check for user input;
   Generate event for input;
   ????
   ????
}
```

Java provides this loop.
You do not write it.

Listeners

- A Listener is a class with methods to respond to input
  - Handles buttons in JMan
  - Each method is a GUI button
  - Support other types of input
- Program registers Listeners with an event type
  - Event loop finds a Listener for the current event type
  - Calls a Listener method
  - Event is passed as argument

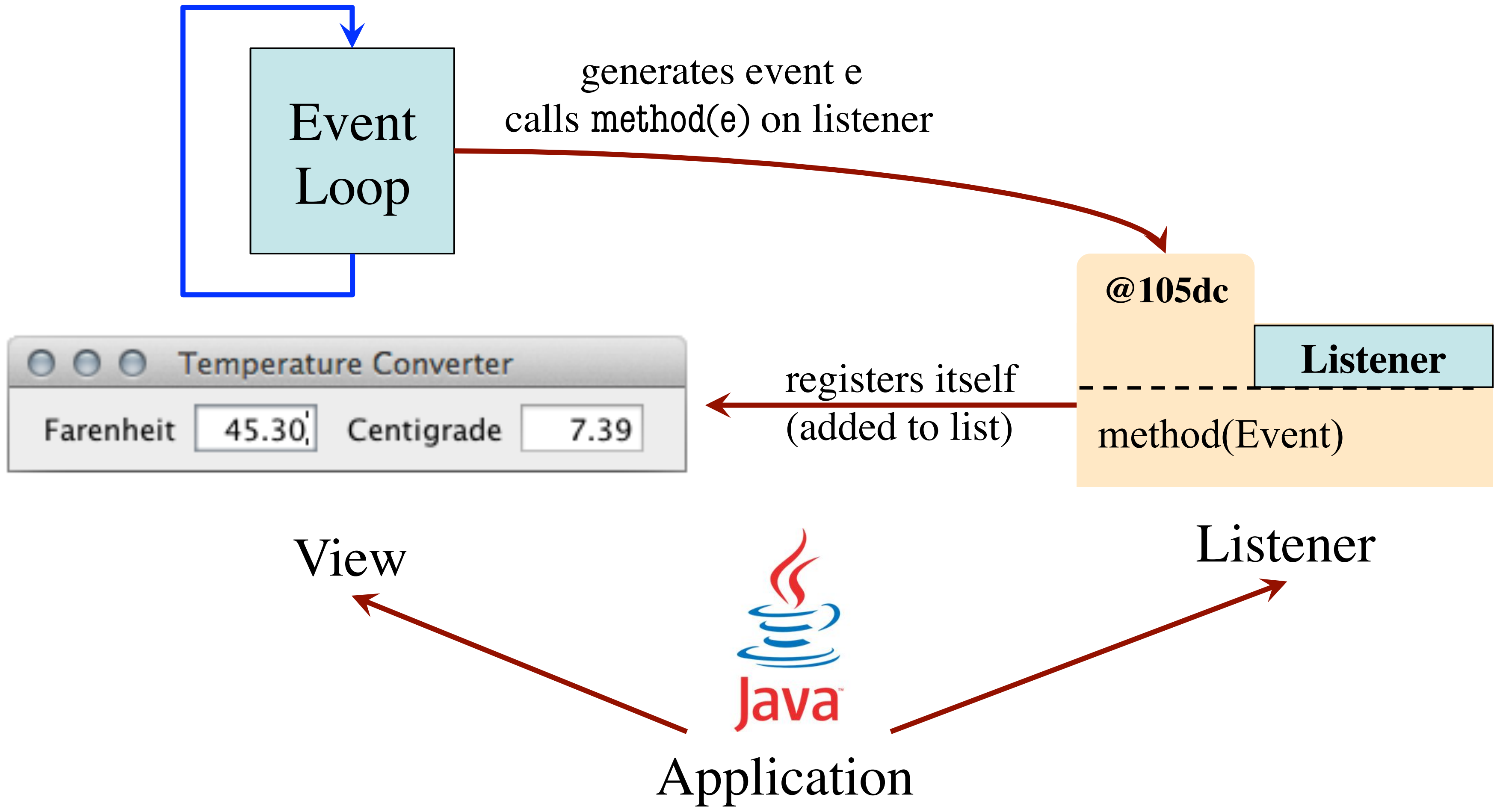


Starts

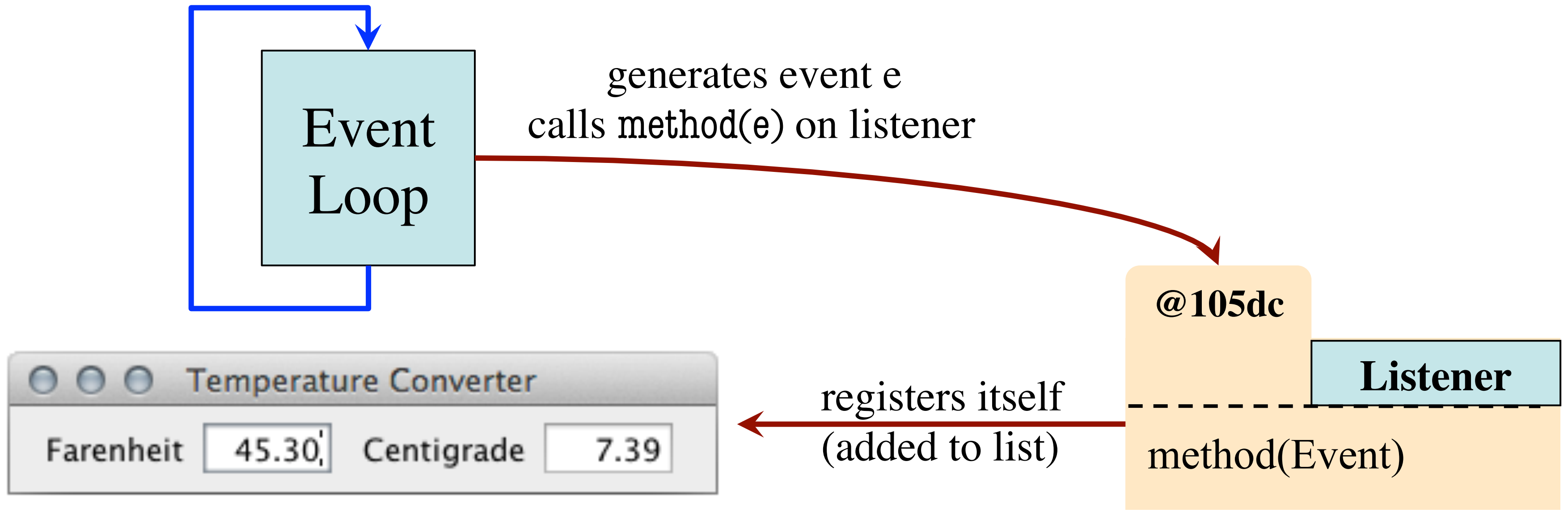
```
while ( JFrame is showing ) {
   Check for user input;
   Generate event for input;
   Find a Listener for this event;
   Call a method in this Listener;
}
```

Java provides this loop.
You do not write it.

Event-Driven Programming



Event-Driven Programming



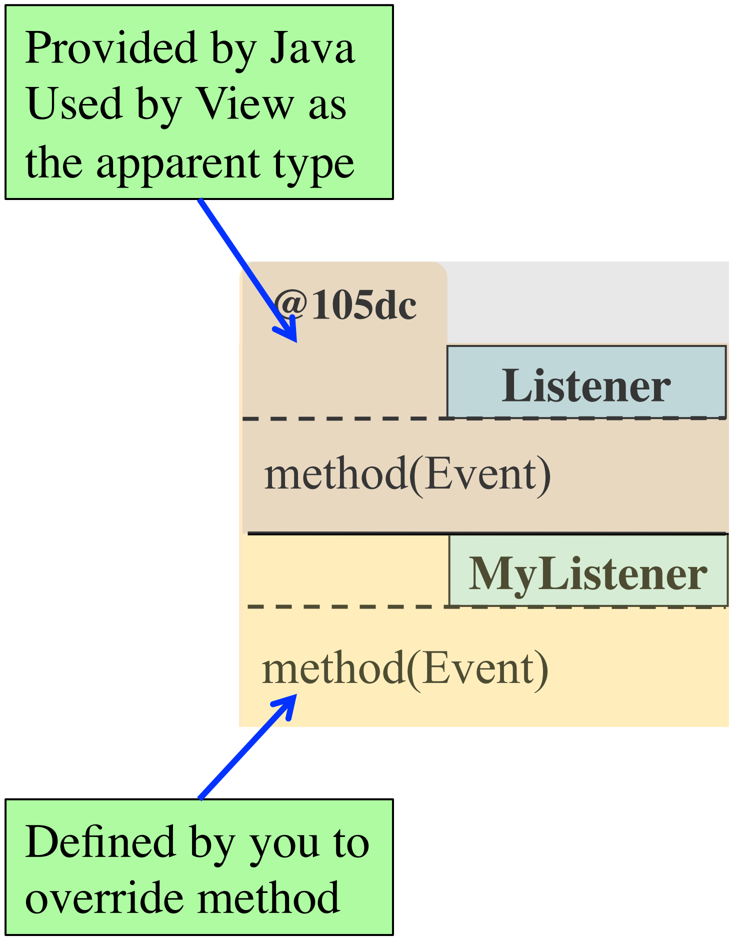
View

Listener

- JFrame has to know
  - Type of the Listener
  - Name of the method
- You did not write JFrame!

Solution: Apparent Types

- Java provides a Listener type
  - Has the method already in it
  - Subclass this as your own class
  - Override method for your usage
- View uses the Listener type
  - Allows it to call the method
  - Uses your version of method (bottom-up rule)
- Designed to be overridden…



Abstract Classes: Made to be Overridden

- Abstract method
  - Has the method header
  - But does not have body!
  - Example: Piece.java
- Why do this?
  - Will use Piece for the apparent type (variable)
  - But Piece will never be the real type of anything
- Artifact of static typing

```
public class Piece {
    …
    // Abstract
    public abstract void
                          act(JManBoard board);
}

public class JMan {
    …
    // IMPLEMENTATION
    public void act(JManBoard board) {
        …
    }
}
```

Listeners are actually Interfaces

- Like an abstract class
  - But **all methods** abstract!
  - And cannot have fields
- What is the difference?
  - Don't **extend** an interface
  - You **implement** one
- What the heck????
  - Major topic in CS 2110
  - Not needed for JMan
  - We did this for you

```
public interface A {
  public void doIt();  // Abstract
}

public class B implements A {
   public void doIt() {
        …
   }
}
```

Listeners and Events in Java

In packages:
- javax.swing.event
- java.awt.event

Events

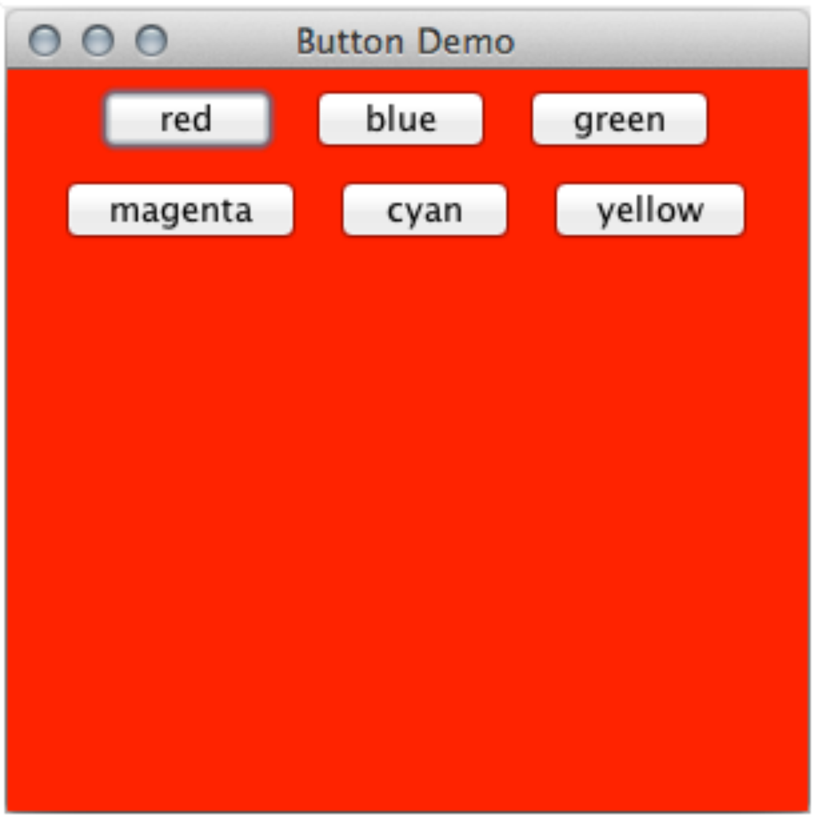
- ActionEvent
  - User clicks a button
  - User hits return in text field
- MouseEvent
  - User clicks the mouse
  - User moves the mouse
- KeyEvent
  - User presses a key
  - User releases a key

Listeners

- ActionListener
  - actionPerformed(ActionEvent)
- MouseListener
  - mouseClicked(MouseEvent)
  - mouseEntered(MouseEvent)
- MouseMotionListener
  - mouseDragged(MouseEvent)
- KeyListener
  - keyPressed(KeyEvent)

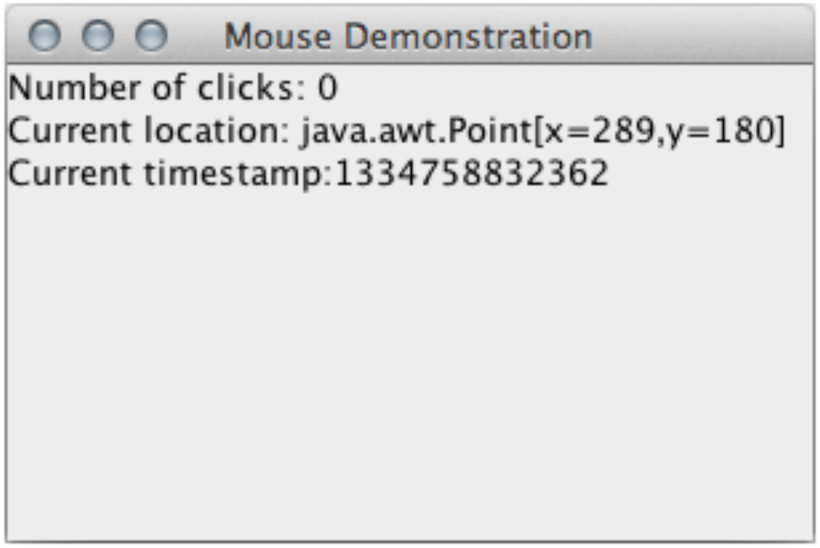
Example: Button Events

- Button generates ActionEvent
- Handle with ActionListener
  - actionPerformed(e)
  - Parameter contain button info
- Implement as separate class
  - A *controller* class
  - ButtonDemoView.java
  - ButtonDemoListener.java
- view.addActionListener(l)
  - Registers the listener
  - Done at start-up



Example: MouseEvents

- MouseListener: simple events
  - Ex: Mouse clicked
  - Stuff that is not updated at "animation frame rate"
- MouseMotionListener: High speed movement
  - Updated 20-30x second
  - Can slow down program!
- Demonstration:
  - MouseDemoView.java
  - MouseDemoListener.java
  - MotionDemoListener.java



```
Number of clicks: 0
Current location: java.awt.Point[x=289,y=180]
Current timestamp:1334758832362
```

Example: KeyEvents

- Only if input has focus
- Motivation:
  - Which text fields gets key?
  - One with the cursor!
  - This is setting focus
- Text fields do automatically
  - Others require `requestFocus()`
- Demonstration:
  - KeyDemoView.java
  - KeyDemoListener.java