Lecture 22

# GUI Listeners

# Announcements for This Lecture

## Prelim II

- Grades were a bit low
  - 69 mean, 72 median
  - Historically a 76
  - Culprit is recursion
- But good grade indicator
  - A (mastery) 75+
  - B (compentency) 52+
  - C (awareness) 33+
- Will make final a bit easier

## Assignments

- A6 due Tonight at Midnight
  - Hopefully you are done!
  - To be graded this weekend
- Assignment A7 up Tomorrow
  - Last assignment of semester
  - Sizeable project; longer than the previous ones
  - Will give you until Saturday after last day of classes

# Main Challenges in GUI Applications

## Layout

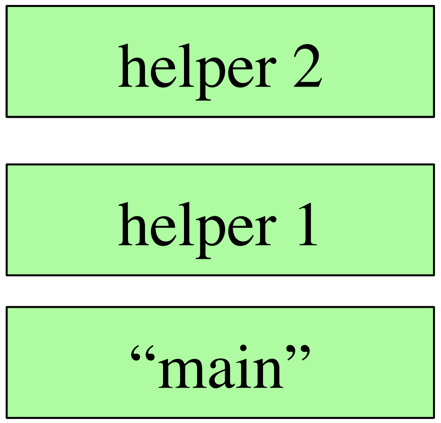
- Arranging items the screen
  - Java has many components
  - But where do they go?
- **Challenge**: Resizing
  - Want components to "behave nicely" as you resize
  - Change size of components
  - Change padding in between
- LayoutManagers do both

## 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
- **Topic of today's lecture**

## Traditional Programming

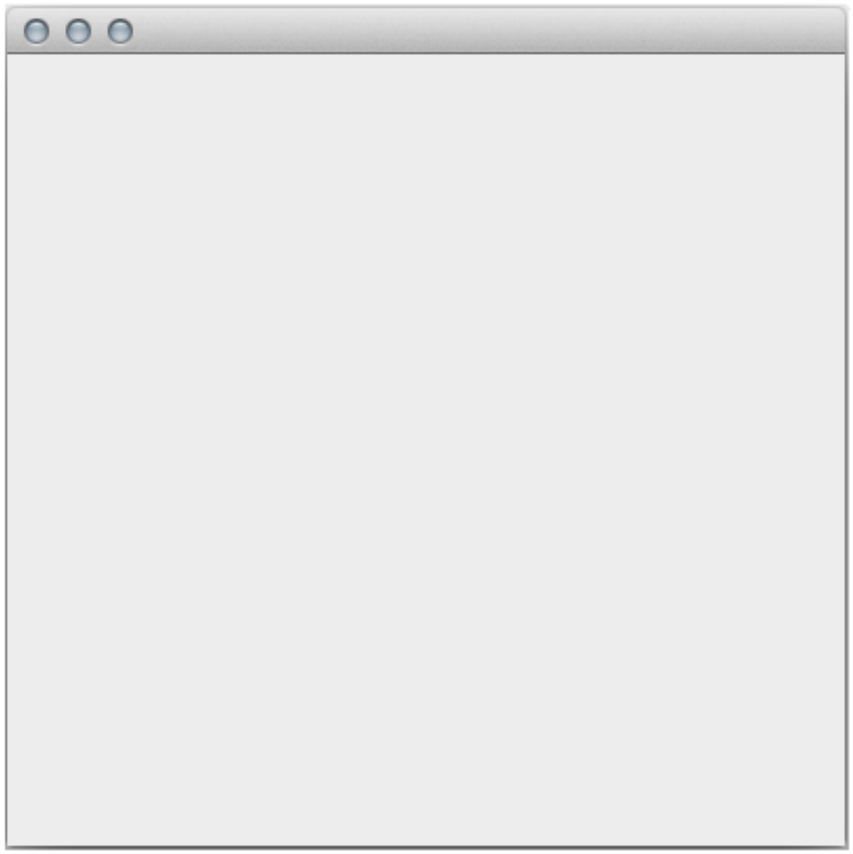
- Have a “main” method
  - Call in Interactions pane
  - Call in JUnit test
  - …somewhere else?
- Other methods are helper methods to “main” one
- Big reason for DrJava
  - Usually only one “main”
  - Interactions pane allows all methods to be “main”



Program ends when “main” is done

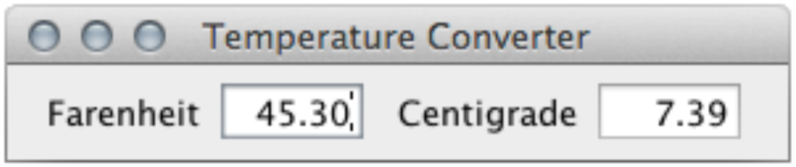
# JFrame is Different

- Get demo code for today
- 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?



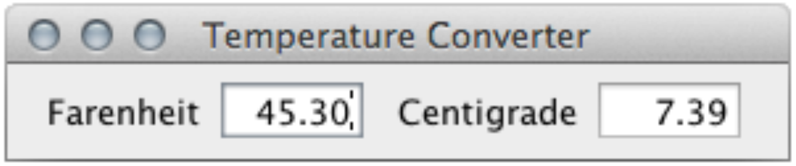
Starts

```
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
  - ImageProcessor in A6
  - 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

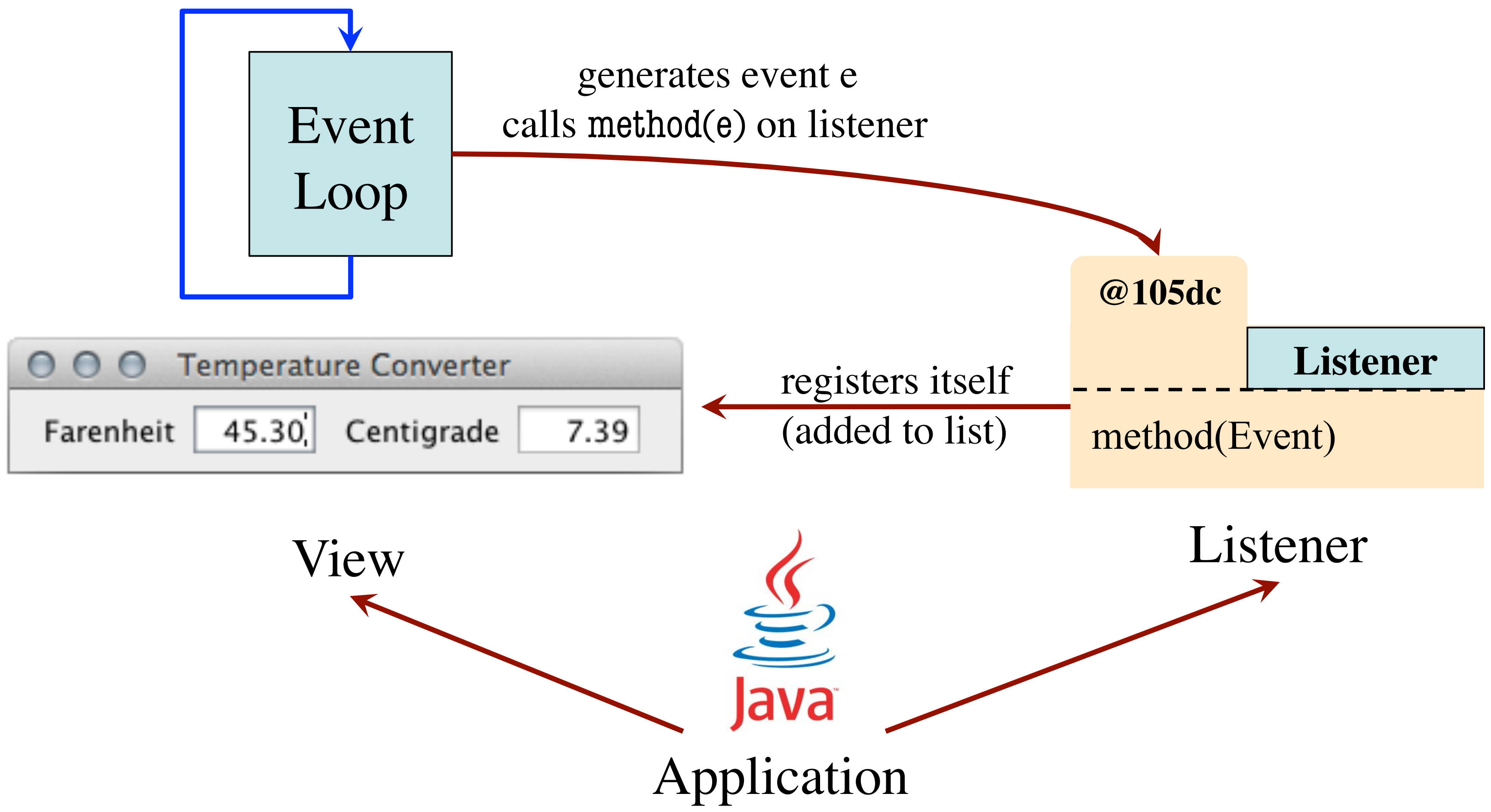


Starts

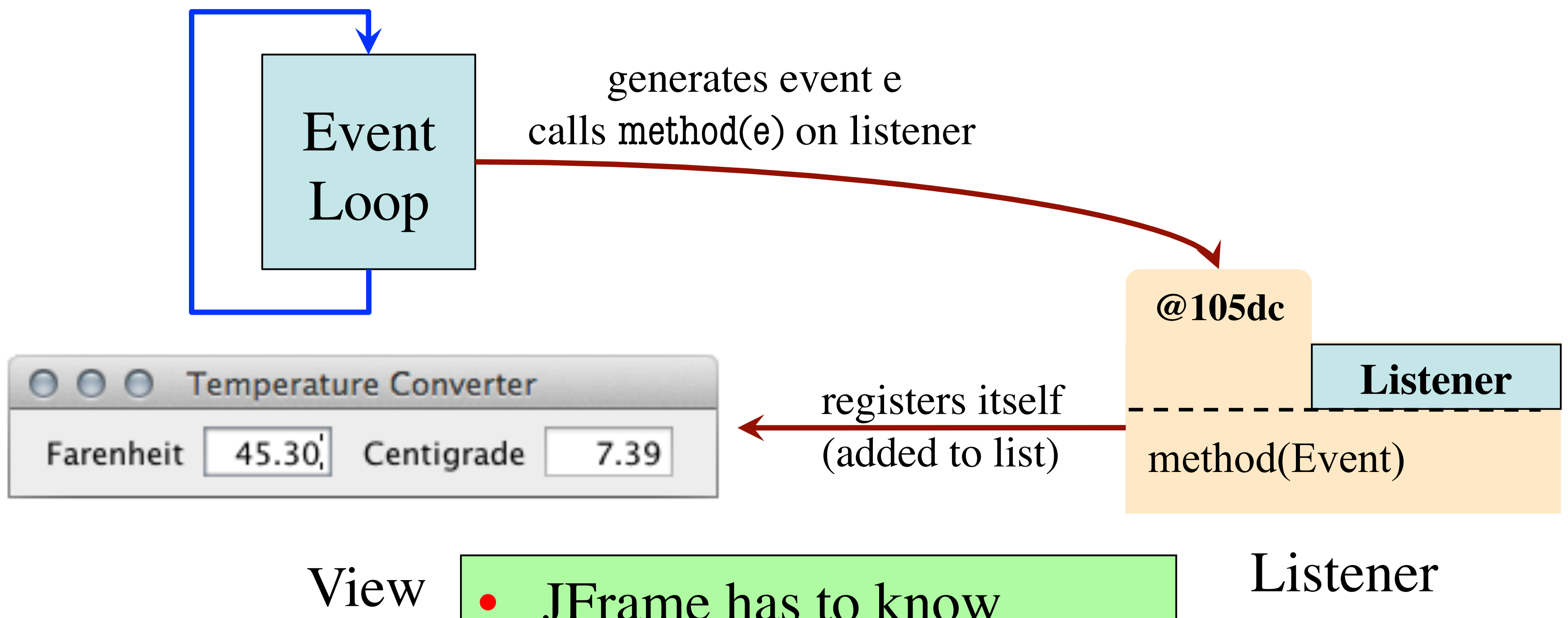
```
while ( JFrame is showing ) {
  Check for user input;
  Generate event for input;
  Find a Listener for this event;
  Call a method in this Listener;
}
```

Java provides this loop.
You do not write it.

# Event-Driven Programming



# Event-Driven Programming

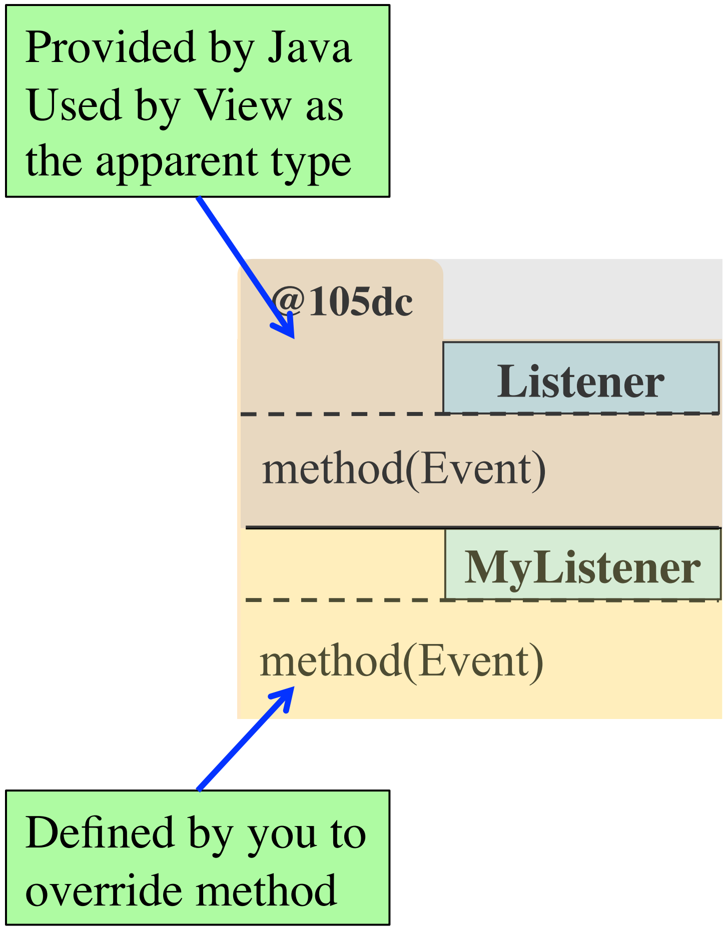


- JFrame has to know
  - Type of the Listener
  - Name of the method
- You did not write JFrame!

## 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…

Sounds like an abstract class!


Provided by Java
Used by View as the apparent type
@105dc
Listener
method(Event)
MyListener
method(Event)
Defined by you to override method

## Well, Almost

- Listeners are interfaces
  - Like an abstract class
  - But **all methods** abstract!
- What is the difference?
  - Don't extend an interface
  - You implement one
- What the heck????
  - Part of lecture next week
  - Major topic in 2110

```
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

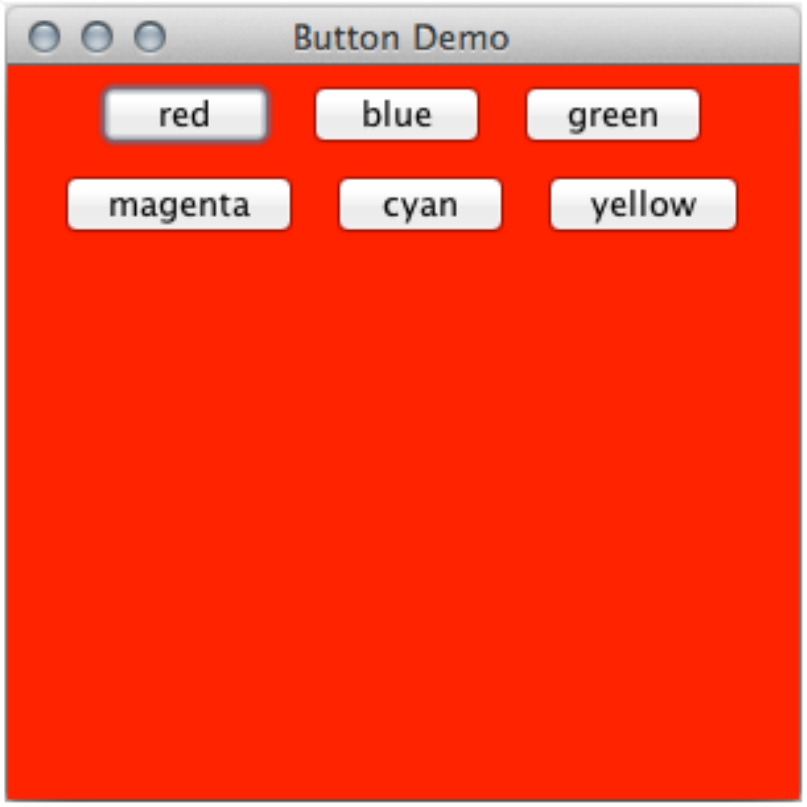
- ActionEvent
  - User clicks a button
  - User hits return in text field
- MouseEvent
  - User clicks the mouse
  - User moves the mouse
- KeyEvent
  - User presses a key
  - User releases a key

## Listeners

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

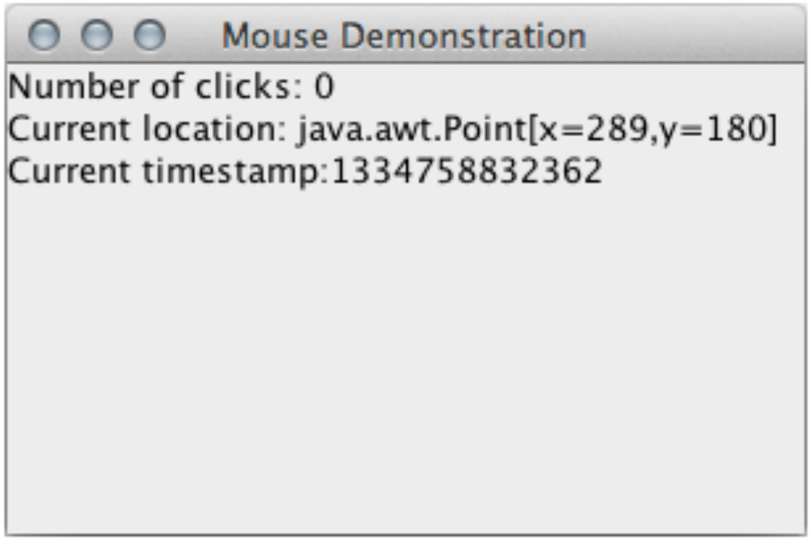
# Example: Button Events

- 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

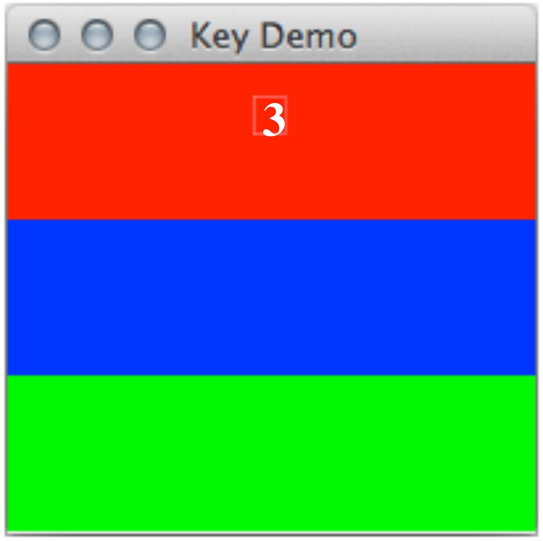
- MouseListener: simple events
  - Ex: Mouse clicked
  - Stuff that is not updated at "animation frame rate"
- MouseMotionListener: High speed movement
  - Updated 20-30x second
  - Can slow down program!
- Demonstration:
  - MouseDemoView.java
  - MouseDemoListener.java
  - MotionDemoListener.java



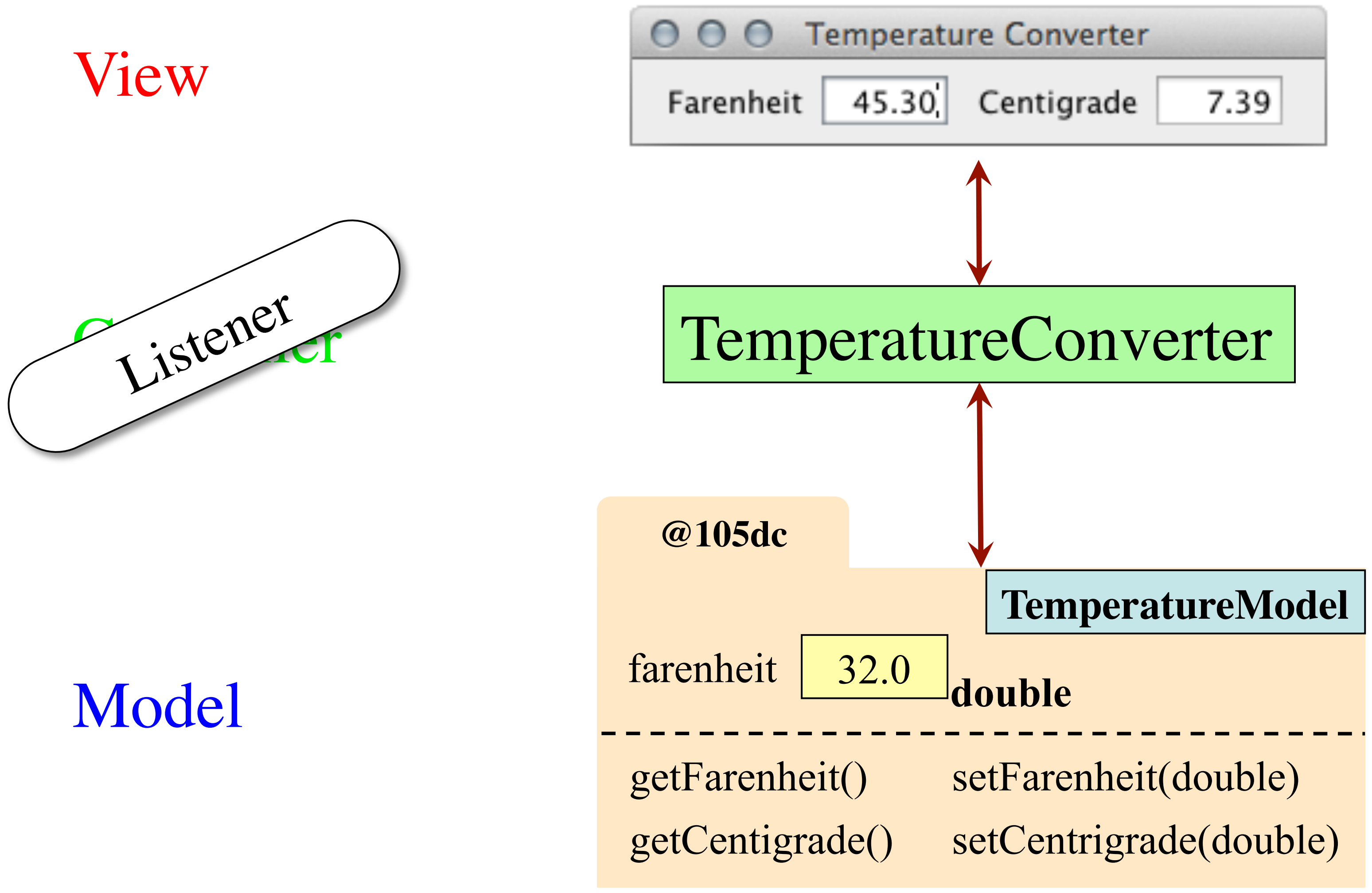
```
Number of clicks: 0
Current location: java.awt.Point[x=289,y=180]
Current timestamp:1334758832362
```

## 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



# TemperatureConverter Revisited


View
Listener
Model
Temperature Converter
Farenheit
45.30
Centigrade
7.39
TemperatureConverter
@105dc
TemperatureModel
farenheit
32.0
double
getFarenheit()
setFarenheit(double)
getCentigrade()
setCentrigrade(double)