

GUIs with Java Swing

School of Computing
Clemson University
Clemson, SC 29634 USA

CPSC 2150: Wednesday, April 12, 2023

Slide Sources:

- 1 Previous/Current CPSC 2150 Instructors
- 2 The lecture slides have also benefitted from instructors at Ohio State: Paolo Bucci, Wayne Heym, Paul Sivilotti, and Bruce Weide.

Last Class

- 1 Midterm Review
- 2 Midterm Exam #2

Homework Assignment

- 1 Programming assignment 4 is posted on your lab Canvas page.
- 2 Programming assignment 5 will be out Friday!

Programs with GUIs

- ▶ A Java program with a graphical user interface (GUI) is pretty routine in most respects
 - ▶ It declares and manipulates the values of some variables of various types, albeit new ones intended for use in developing GUIs (e.g., buttons, scrollbars, drawing panels, etc.)
- ▶ There is just one (big) issue
 - ▶ How to handle the user interaction problem?
 - ▶ The user can interact with any user interface object at any time
 - ▶ How can we react to different events?
- ▶ We saw how we can use the **Observer Pattern** to solve this problem
- ▶ Today we talk about how the Java Swing library is used to create programs with GUIs

Java Swing

- ▶ Java Swing is a library used to create programs with GUIs
- ▶ Inside that library, we have some important interfaces and classes
 - ▶ `JFrame`
 - ▶ `ActionListener`
 - ▶ `ActionEvent`
- ▶ JavaFX is another library
 - ▶ Concepts are very similar
 - ▶ JavaFX is more powerful
 - ▶ But also much more complicated
 - ▶ Has all the functionalities in Java Swing
 - ▶ Java Swing is much easier to learn first

JFrame Class

- ▶ The `JFrame` class represents the screen that will be displayed on the window
 - ▶ Has functionality to re-size, minimize and close
 - ▶ Has a larger area for us to add widgets in
 - ▶ Has a title bar on the top of the window
- ▶ This class provides a lot of functionality
- ▶ Every time we want to make a new screen in our program, we'll add a new class that **extends** the `JFrame` class
 - ▶ Inherits the basic functionality we need

ActionListener Interface

- ▶ This is the **interface** for our **Observer Pattern**
- ▶ It provides one method: `actionPerformed(ActionEvent e)`
 - ▶ This is our *callback* method
- ▶ Our screen will **implement** the **ActionListener** interface
 - ▶ Provides a method for `actionPerformed` that checks the event and responds appropriately
 - ▶ Does not have all the code to respond to the event, just determine which event it was and call the appropriate method on the controller.
- ▶ Then when an event occurs to one of our widgets, it knows to call the `actionPerformed` method on all of it's observers
 - ▶ This means we need our screen to register as an observer

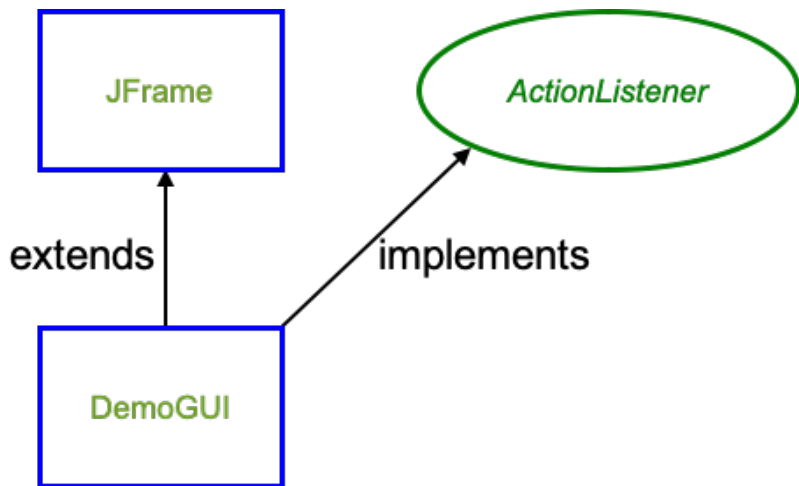
```
public interface ActionListener extends EventListener {  
    public void actionPerformed(ActionEvent e);  
}
```

ActionEvent Class

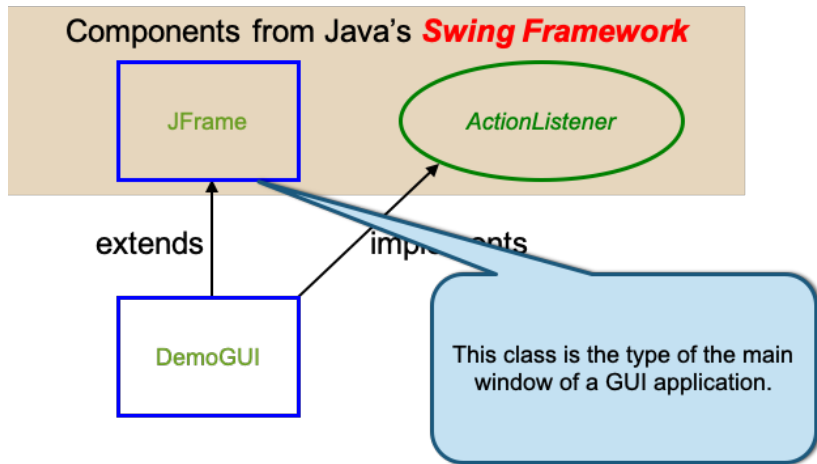
- ▶ This `class` provides a way of passing on the subject and event to our observer
- ▶ Provides a method called `getSource` that returns an object
 - ▶ That will tell us what widget the user interacted with
 - ▶ Remember every widget will call the same `actionPerformed` event on the observer
 - ▶ So we can check the source of the `ActionEvent` that's passed in to determine how to react

```
public class(ActionEvent) extends AWTEvent {  
    ...  
    public Object getSource() {...}  
    ...  
}
```

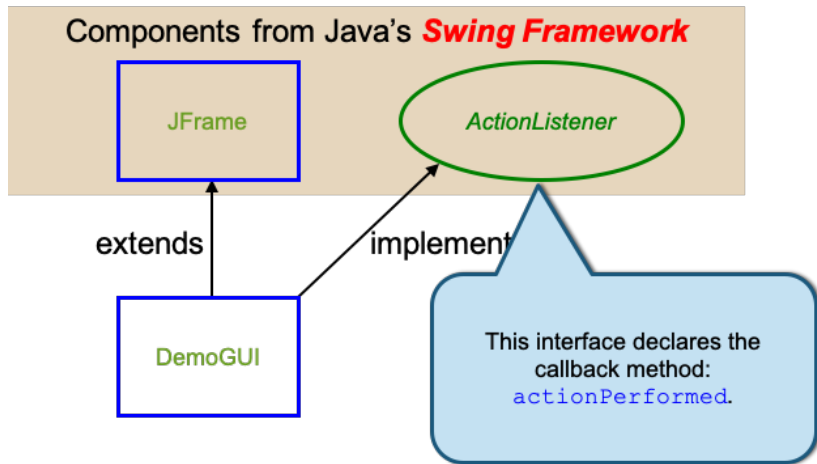
Example: Simple GUI Demo (No MVC)



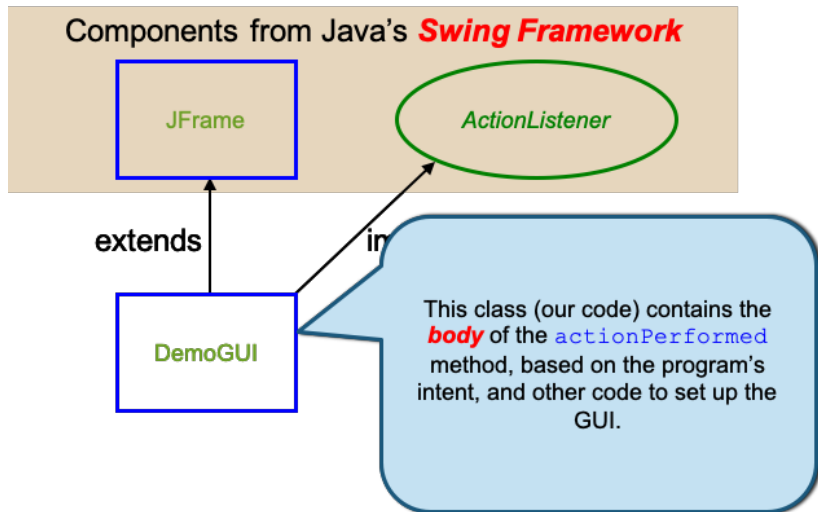
Example: Simple GUI Demo (No MVC)



Example: Simple GUI Demo (No MVC)



Example: Simple GUI Demo (No MVC)



Java Swing Widgets

- ▶ Some important classes in `javax.swing`:
 - ▶ `JFrame`
 - ▶ `JPanel`
 - ▶ `JButton`
 - ▶ `JScrollPane`
 - ▶ `JTextArea`
 - ▶ `JCheckBox`
 - ▶ `JComboBox`
 - ▶ ...
- ▶ These are the main classes related to GUI
- ▶ For more information on these (and other useful classes), see:
<https://docs.oracle.com/en/java/javase/11/docs/api/java.desktop/javax/swing/package-summary.html>

Java Widgets

- ▶ Java widgets have many important methods to help us control how the user interface looks
 - ▶ Sizing of the widget
 - ▶ Often can't set an exact size, which makes the screen more dynamic
 - ▶ Can set minimum and maximum sizes
 - ▶ Can set a preferred size
 - ▶ Text on the widget
 - ▶ Color
 - ▶ Border
 - ▶ ...
- ▶ These methods are thoroughly documented

Adding Widgets to Our Screen

- ▶ To add a widget to our screen, we declare a variable of that widget type:
 - ▶ `JButton submitButton = new JButton("Submit");`
- ▶ Then we add it to our screen using the `add` method provided by `JFrame`:
 - ▶ `this.add(submitButton);`
- ▶ Where it appears on our screen will be determined by our `LayoutManager` and any panels or panes we add.
- ▶ We also need to register as an observer, so we will be notified when someone uses the widget
 - ▶ `submitButton.addActionListener(this);`
- ▶ All of this happens in our constructor for our screen class

Instance Variables

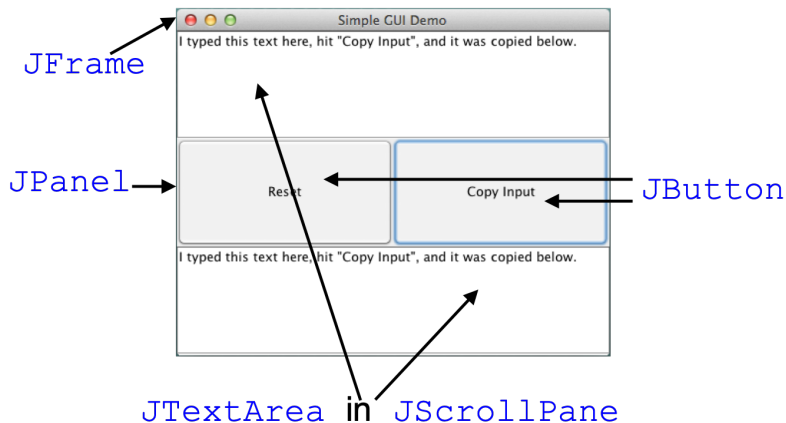
- ▶ Variables can be declared:
 - ▶ in method bodies: **local variables**
 - ▶ in method headers: **formal parameters**
 - ▶ in classes: **fields** or **instance variables**
- ▶ **Examples of instance variables¹:**
 - ▶ `myResetButton`
 - ▶ `myCopyButton`
 - ▶ `myInputText`
 - ▶ `myOutputText`
 - ▶ `myInput`
 - ▶ `myOutput`
 - ▶ ...
- ▶ Instance variables are essentially *global variables* that are shared by and can be accessed from all instance methods in the class

¹**Note:** I prefix every instance variable (proper camel case) with `my`. Makes it easy to distinguish between instance and non-instance variables

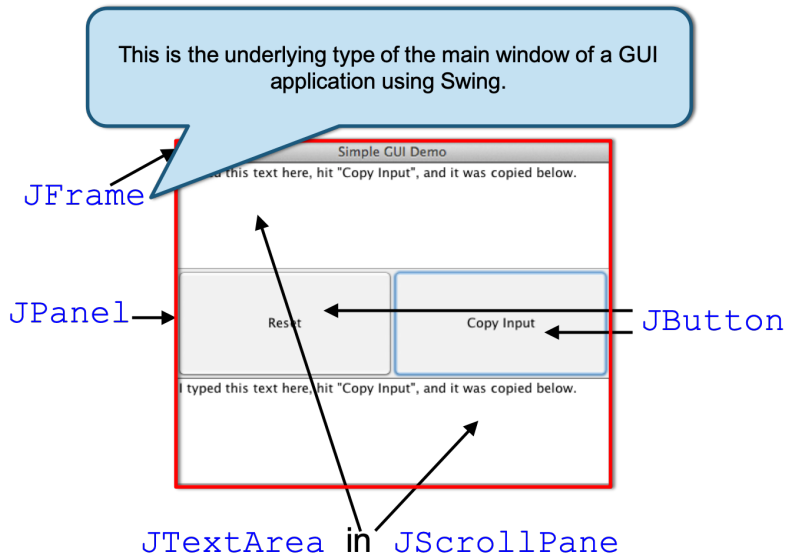
Layout Managers

- ▶ A *layout manager* allows you to arrange widgets without providing specific location coordinates
- ▶ In Java, they **implement** the **LayoutManager** (or **LayoutManager2**) interface
 - ▶ **GridLayout** (Simplest(?) Used in **DemoGUI**)
 - ▶ Define a number of rows and columns
 - ▶ As you add a widget, it fills the screen row by row
 - ▶ **FlowLayout** (Default for **JPanel**)
 - ▶ **BorderLayout** (Default for **JFrame**)
 - ▶ ...
- ▶ **See:** <https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>

Fundamentals: DemoGUI

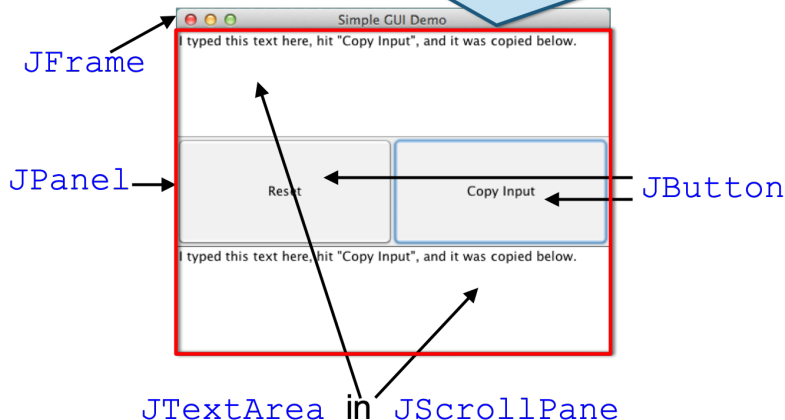


Fundamentals: DemoGUI

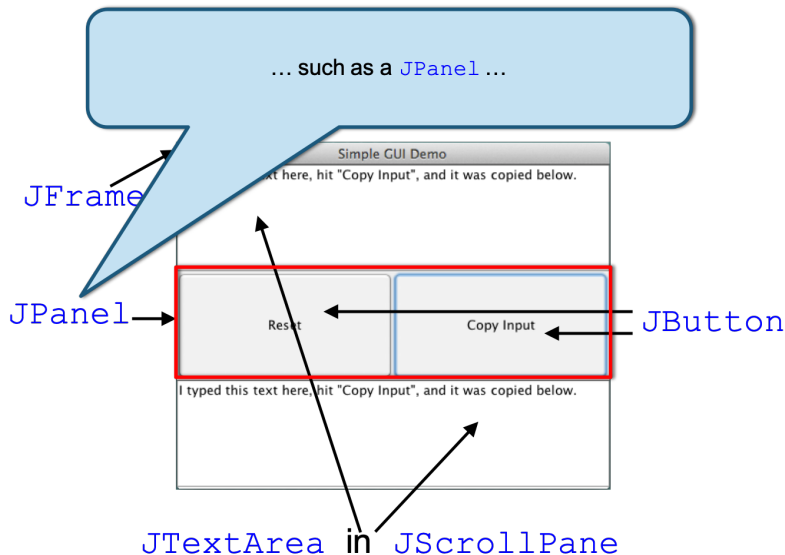


Fundamentals: DemoGUI

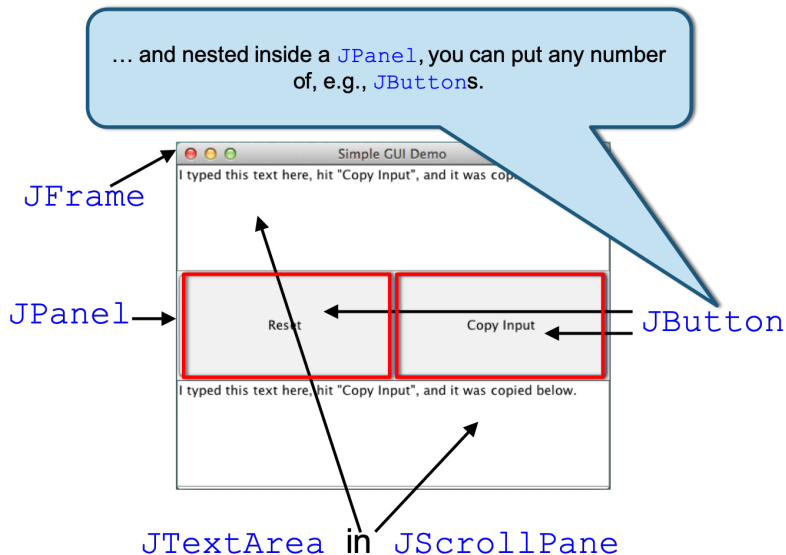
Nested inside a `JFrame`'s **content pane**, you can put any number of things ...



Fundamentals: DemoGUI

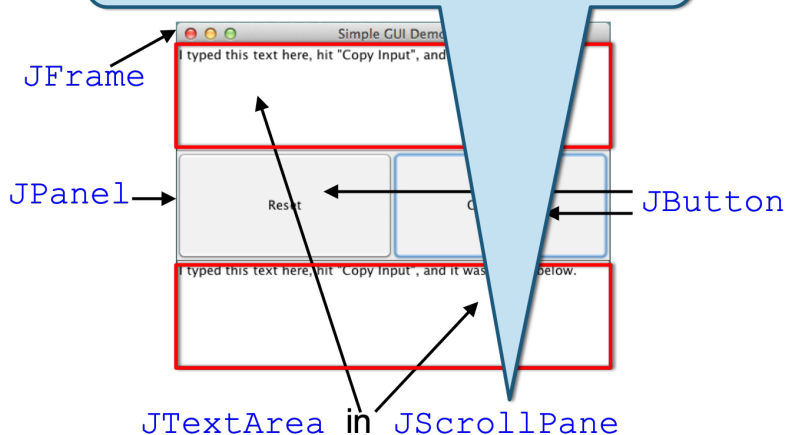


Fundamentals: DemoGUI

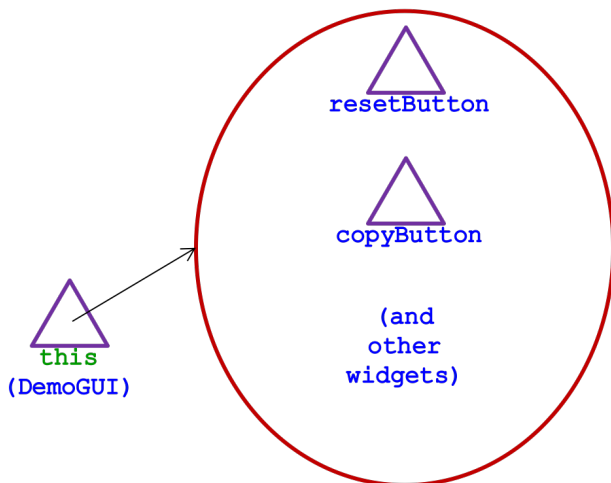


Fundamentals: DemoGUI

You can also put in a `JScrollPane` with, e.g., a `JTextArea`.

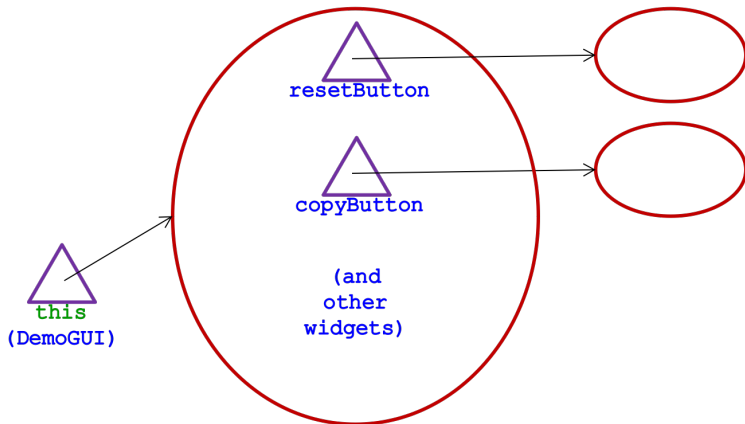


Illustrative Example: Setup by DemoGUI



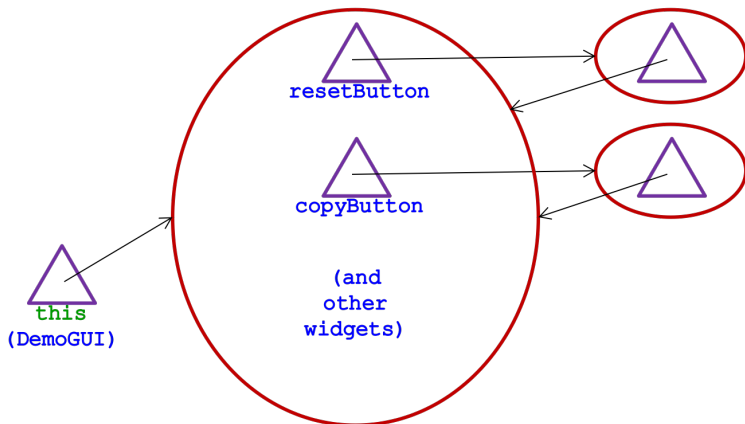
Set Up by DemoGUI Constructor

Illustrative Example: Setup by DemoGUI



Before registration of `this` as an observer of the buttons.

Illustrative Example: Setup by DemoGUI



After registration of `this` as an observer of the buttons.

Constructor for View Class

Tasks To Be Performed

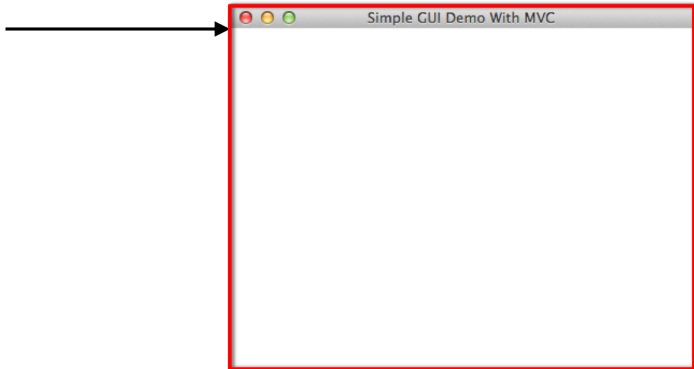
- ▶ A simple constructor for the **view** class has four main jobs:
 - 1 Create the `JFrame` being extended
 - 2 Set up the GUI widgets to be used and “lay out” these widgets in the main application window
 - 3 Set up the observers by registering (in our examples) the object being constructed
 - ▶ i.e., `this`, with each of the GUI widgets that might have events of interest to the application
 - 4 Start the main application window

1. Create the JFrame

- ▶ Use `super` to call the parent constructor:
`super("Simple GUI Demo With MVC");`

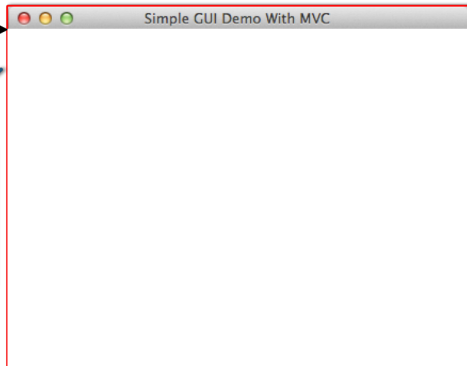
External (GUI) Effect

this



External (GUI) Effect

this



Nothing illustrated in these slides actually becomes visible to the user until later!

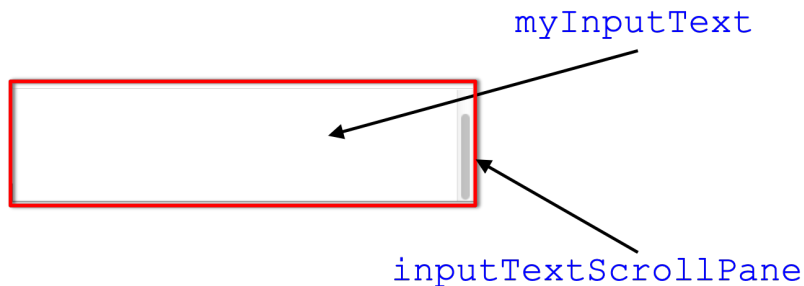
2. Setup GUI Widgets (Text) ...

- ▶ Input Text and Scroll Pane²:

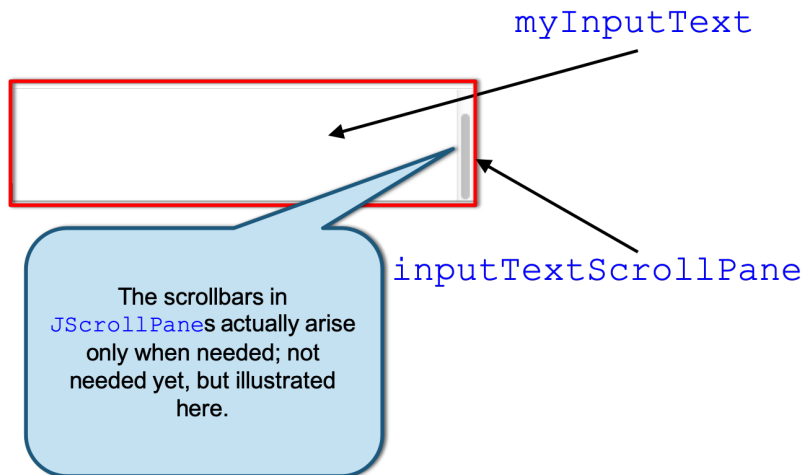
```
myInputText = new JTextArea(" ",  
    LINES_IN_TEXT_AREAS ,  
    LINE_LENGTHS_IN_TEXT_AREAS);  
...  
JScrollPane inputTextScrollPane =  
    new JScrollPane(myInputText);
```

²`LINES_IN_TEXT_AREAS` and `LINE_LENGTHS_IN_TEXT_AREAS` are constants has been defined to be 5 lines and each with a length of 20 characters long

External (GUI) Effect



External (GUI) Effect



2. Setup GUI Widgets (Buttons) ...

- ▶ Copy Button:

```
myCopyButton =  
    new JButton("Copy Input");
```

External (GUI) Effect

myCopyButton



2. ...and Lay Out GUI Widgets

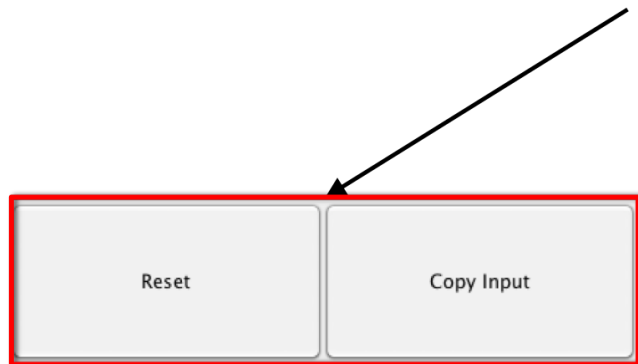
► Button Panel³:

```
JPanel buttonPanel =  
    new JPanel(  
        new GridLayout(  
            ROWS_IN_BUTTON_PANEL_GRID ,  
            COLUMNS_IN_BUTTON_PANEL_GRID));  
  
...  
buttonPanel.add(myResetButton);  
buttonPanel.add(myCopyButton);
```

³ROWS_IN_BUTTON_PANEL_GRID and COLUMNS_IN_BUTTON_PANEL_GRID are constants has been defined to be 1 row and 2 columns

External (GUI) Effect

buttonPanel



2. ...and Lay Out GUI Widgets

- Lay out for this `JFrame`⁴:

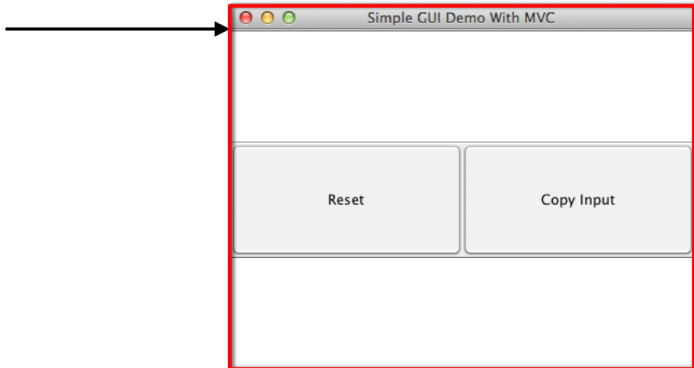
```
this.setLayout(  
    new GridLayout(  
        ROWS_IN_THIS_GRID ,  
        COLUMNS_IN_THIS_GRID));  
  
...  
this.add(inputTextScrollPane);  
this.add(buttonPanel);  
this.add(outputTextScrollPane);
```

- **Remember:** Two buttons are in `buttonPanel`

⁴`ROWS_IN_THIS_GRID` and `COLUMNS_IN_THIS_GRID` are constants has been defined to be 3 rows and 1 column

External (GUI) Effect

this

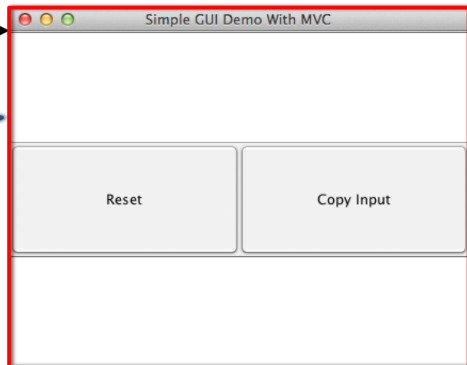


External (GUI) Effect

this



Remember:
none of this is visible
to the user yet.

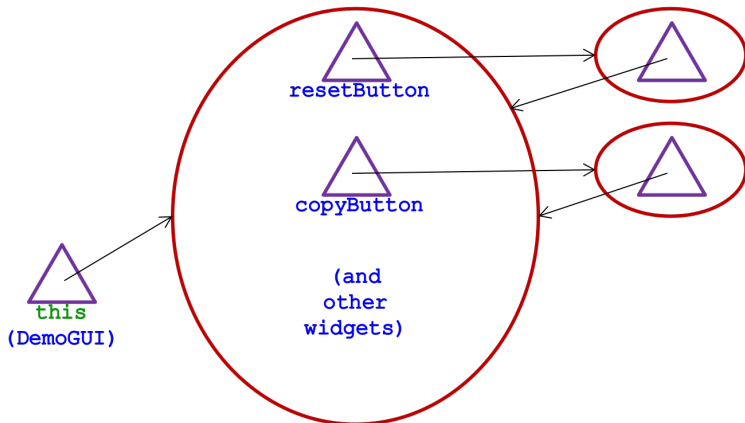


3. Set Up the Observers

- ▶ Registering this `JFrame` as an observer of the buttons:

```
myResetButton.addActionListener(this);  
myCopyButton.addActionListener(this);
```

Internal (non-GUI) Effect



After registration of `this` as an observer of the buttons.

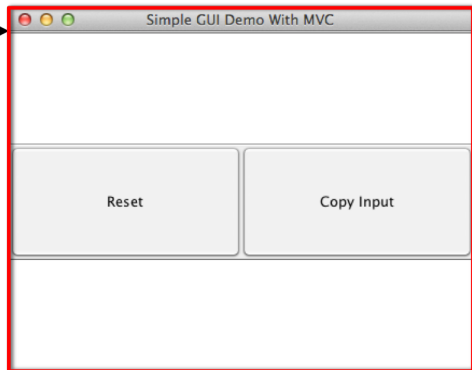
4. Start the Main Window

- ▶ Add more configuration settings before set the window as visible:

```
this.pack();  
this.setDefaultCloseOperation(  
    JFrame.EXIT_ON_CLOSE);  
this.setVisible(true);
```

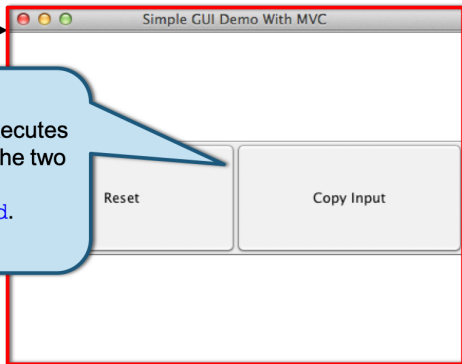
External (GUI) Effect: Now Visible

this



External (GUI) Effect: Now Visible

`this`



The only code you wrote that executes now is the callback method for the two buttons:

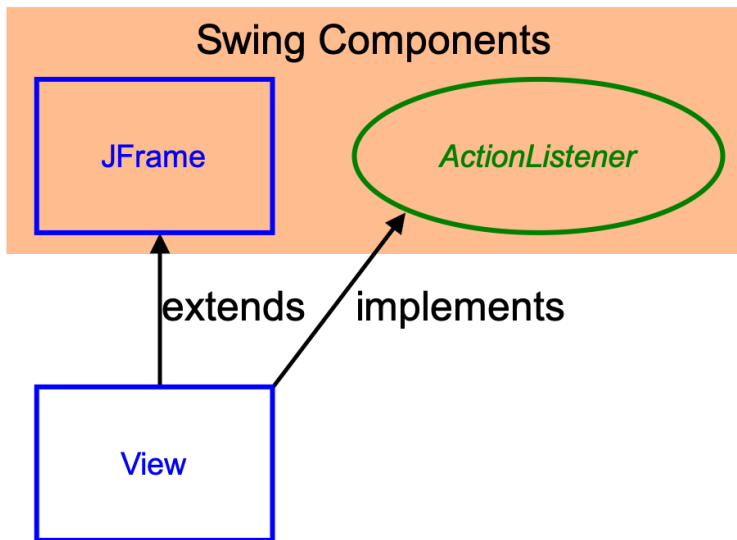
`this.actionPerformed.`

Model-View-Controller

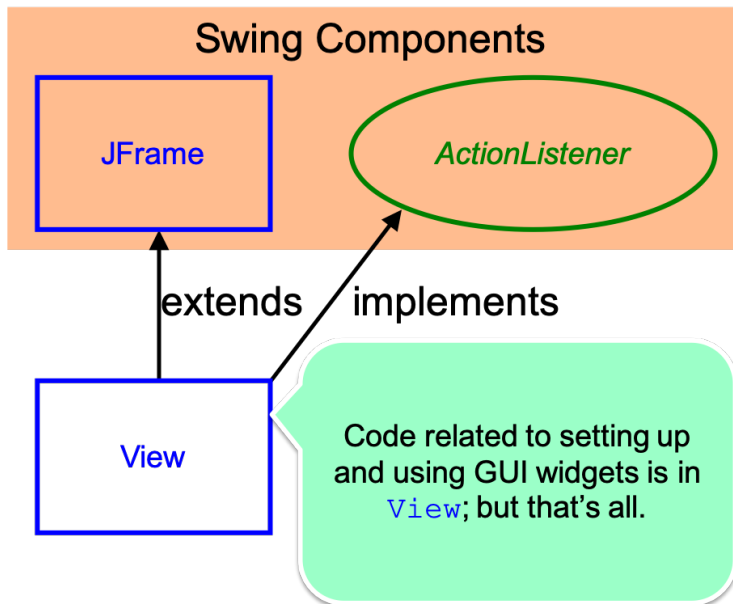
Recall: MVC Pattern

- ▶ The dominant approach to organizing software with GUIs is the **model-view-controller architectural pattern**
- ▶ There are more meaningful examples of this pattern
 - ▶ We illustrate one that is very clean
 - ▶ There should be interfaces for the model, view, and controller classes, but they are left out here *only* to keep the sample code smaller

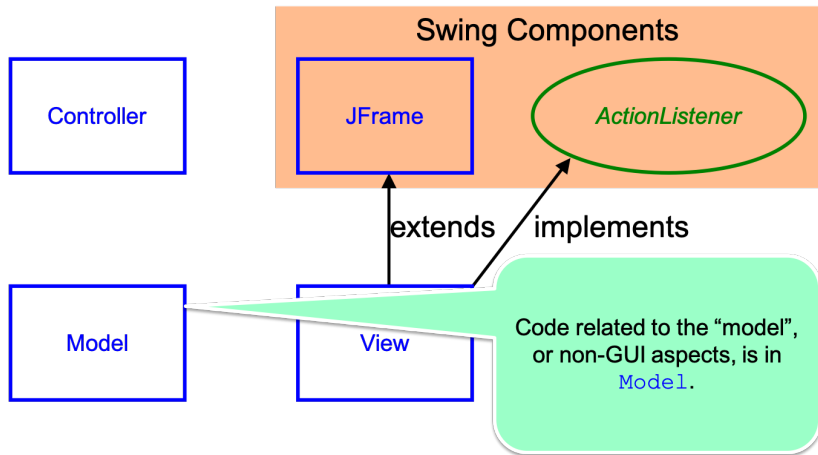
Example: Simple MVC GUI Demo



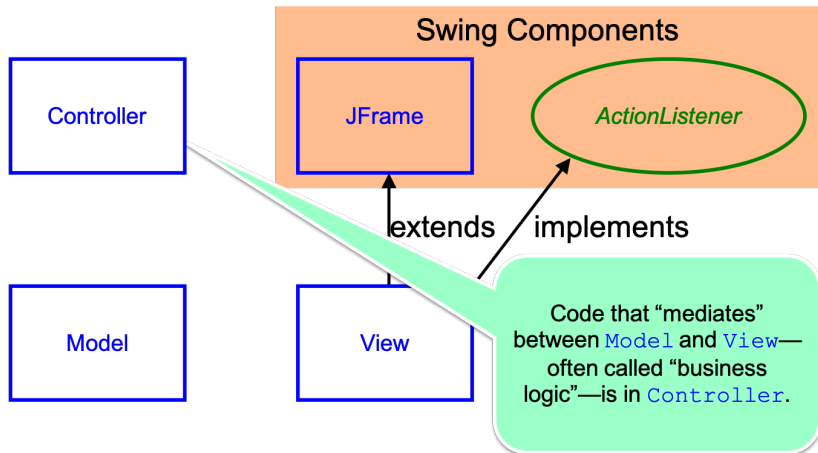
Example: Simple MVC GUI Demo



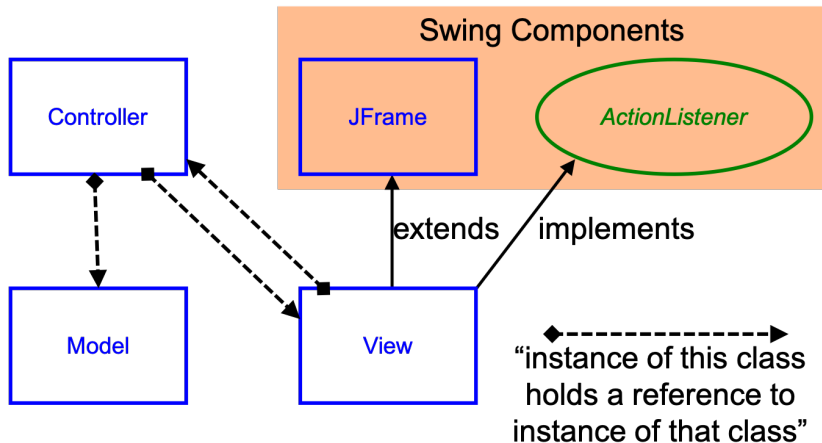
Example: Simple MVC GUI Demo



Example: Simple MVC GUI Demo



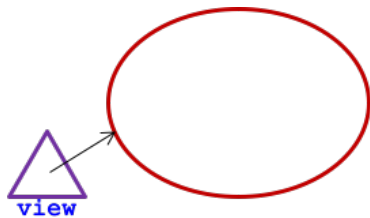
Example: Simple MVC GUI Demo



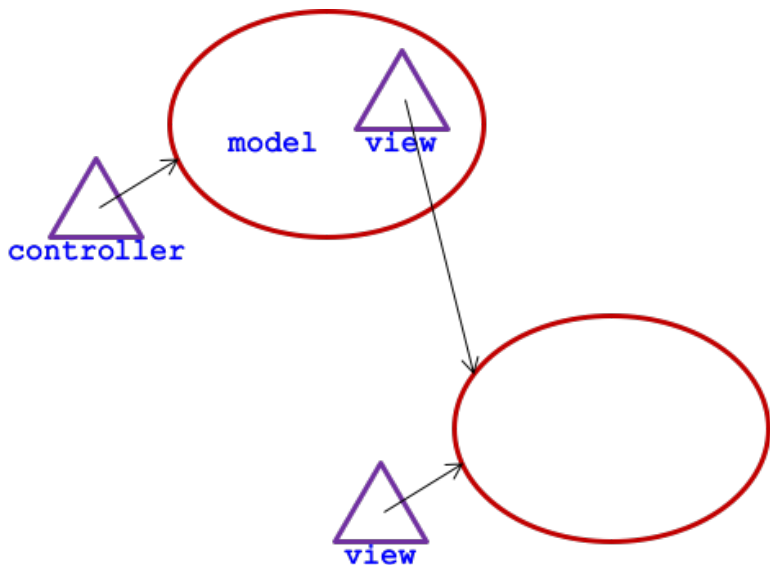
Our main method

- ▶ So how does this actually run?
- ▶ We still need a main driver class and method
 - ▶ The `main` method is the entry point of the program
- ▶ `main` method must:
 - ▶ Create our view instance
 - ▶ Create our controller instance, passing in the view to the constructor
 - ▶ Register our controller as an observer of our view
 - ▶ So view can call methods of the controller class to react to events
- ▶ `main` method might:
 - ▶ Create our model instance as well
 - ▶ Often done by the controller since it knows what model object(s) it will need

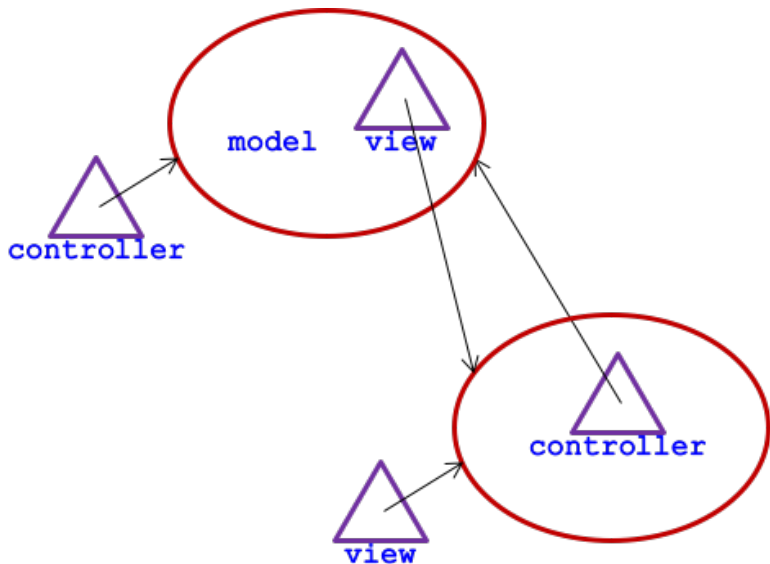
Set-up by `main`: Create view



Set-up by `main`: Create controller



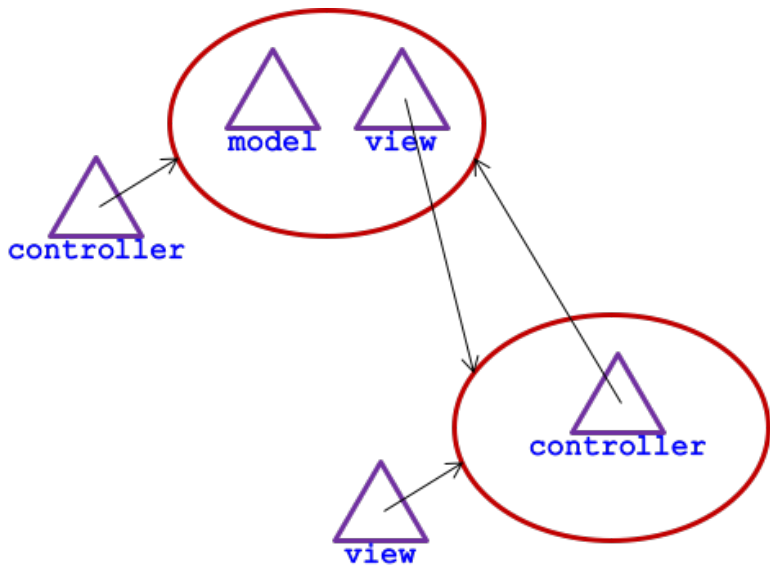
After `view.registerObserver`



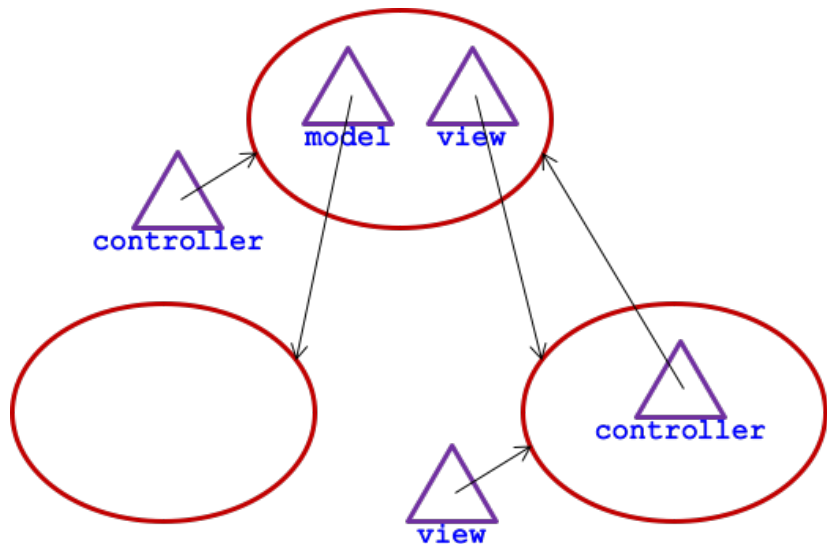
Now, Who's In Charge?

- ▶ **Note:** when `main` is executed:
 - ▶ Calls all constructors
 - ▶ Registers the controller as an observer of view
 - ▶ Then returns to `main`, and there is no more code
- ▶ After that, what code is executing?
 - ▶ Depends on the program
 - ▶ A GUI based program will then wait for user interactions (**View** handles with the **Observer Pattern**)
 - ▶ Other code may pass control to the controller object

After Controller declares Model object



After calling Model constructor



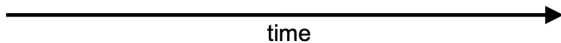
Threads

Threads

- ▶ A standard Java program executes in a **thread**
 - ▶ i.e., a single path of sequential code executing one step at a time
- ▶ A GUI program with Swing uses at least *two* threads rather than one:
 - ▶ The **initial thread** executes `main` (until it completes)
 - ▶ An **event dispatch thread** executes everything else, including `actionPerformed`

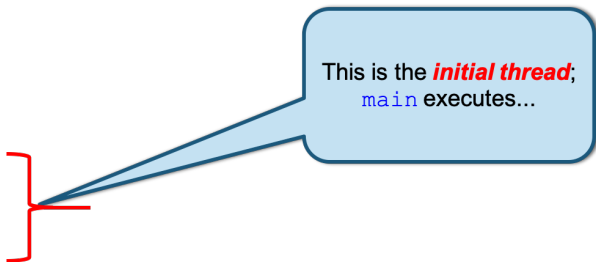
Timeline of Thread Execution

main



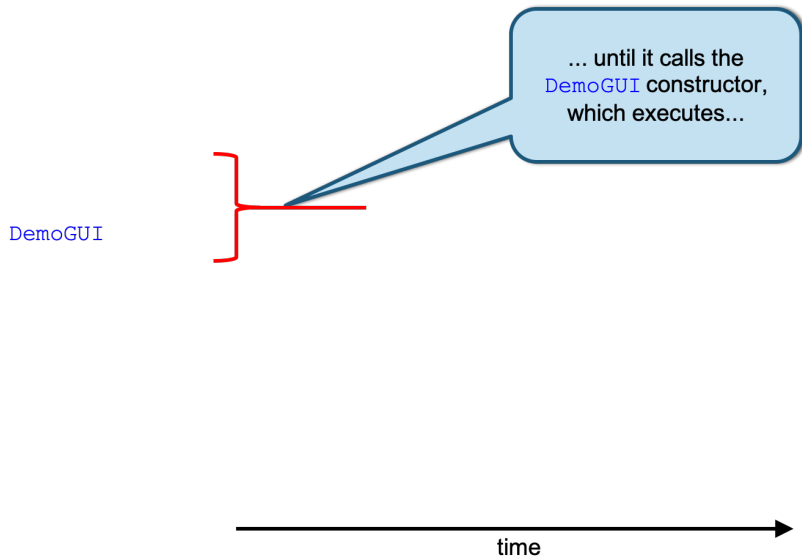
Timeline of Thread Execution

main

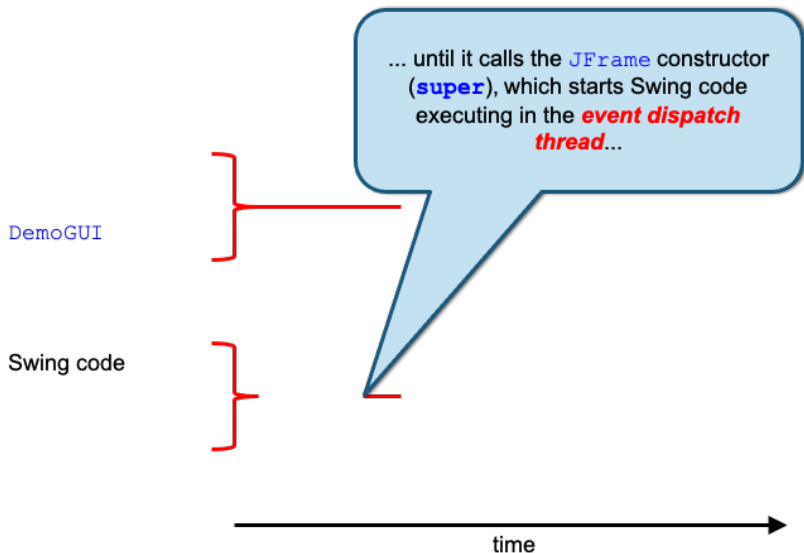


time

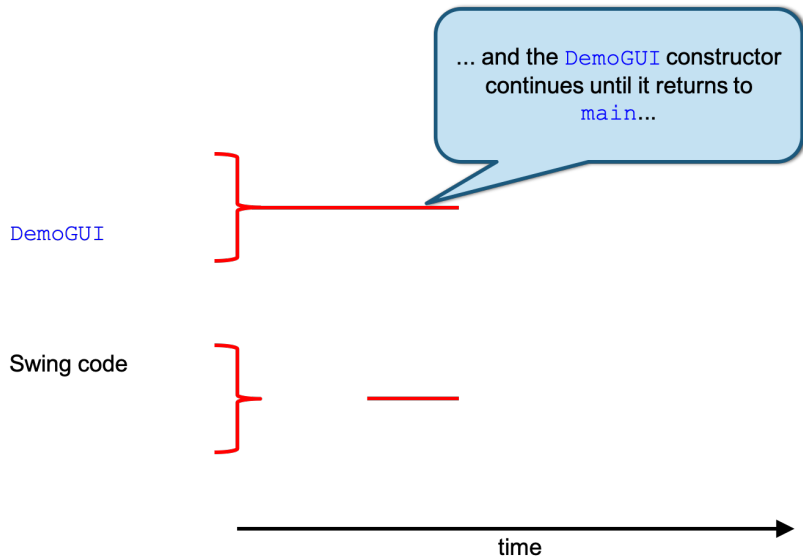
Timeline of Thread Execution



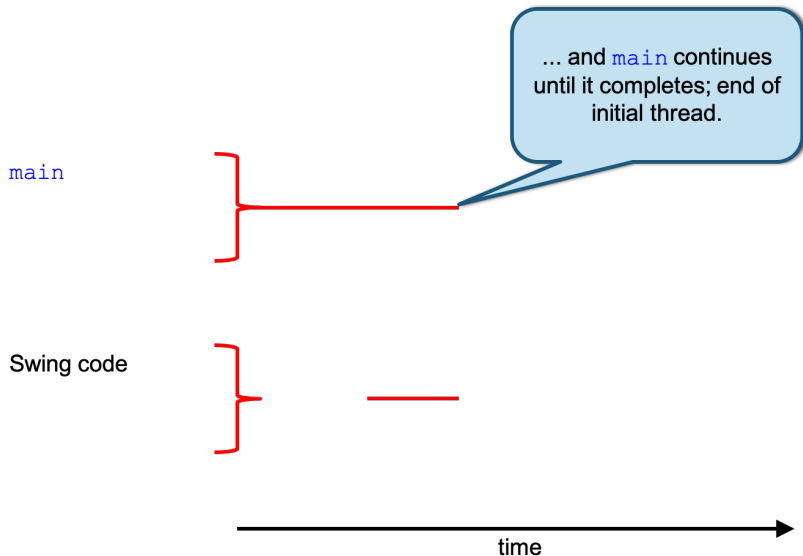
Timeline of Thread Execution



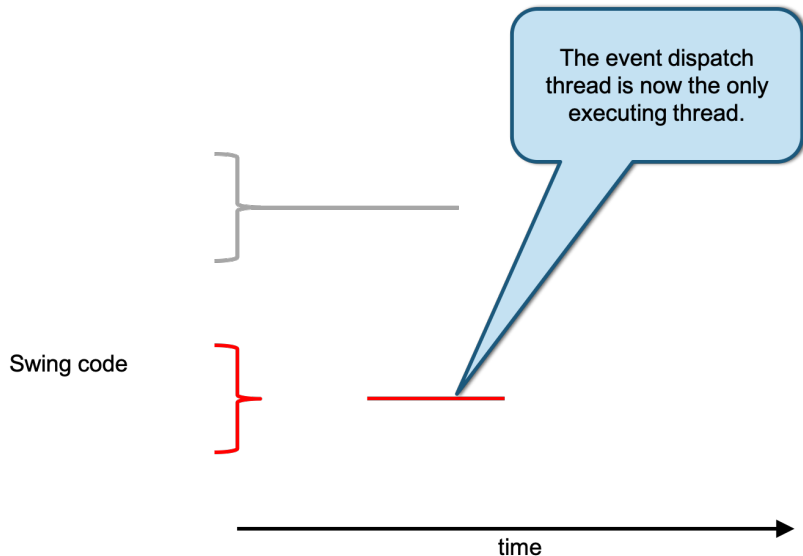
Timeline of Thread Execution



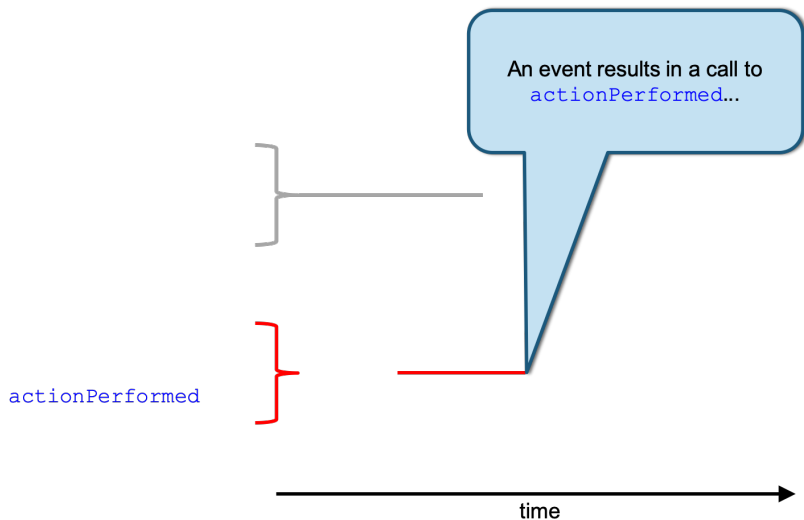
Timeline of Thread Execution



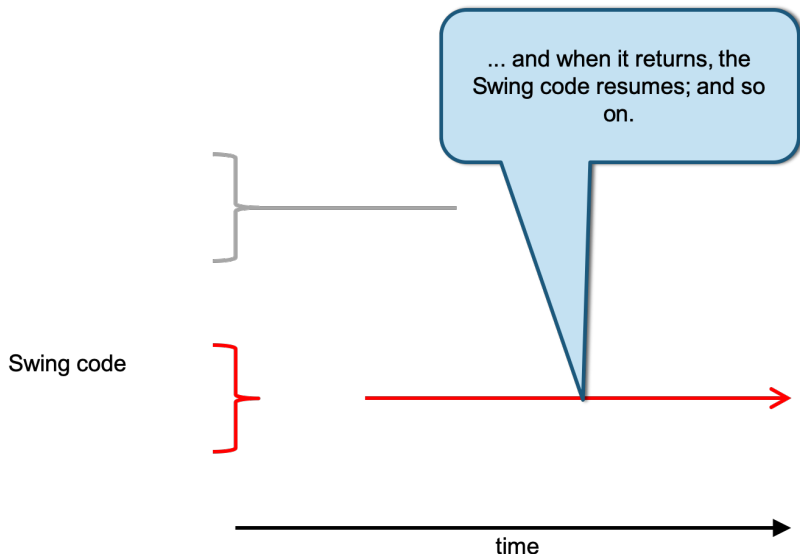
Timeline of Thread Execution



Timeline of Thread Execution



Timeline of Thread Execution



Resources

- ▶ Java Tutorials (and beyond...)
 - ▶ <https://docs.oracle.com/javase/tutorial/uiswing/index.html>
- ▶ A Visual Guide to Layout Managers
 - ▶ <https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>
- ▶ Observer Pattern
 - ▶ https://en.wikipedia.org/wiki/Observer_pattern

Homework Assignment

- 1 Programming assignment 4 is posted on your lab Canvas page.
- 2 Programming assignment 5 will be out Friday!

Summary

- 1 Programs with GUIs
- 2 Java Swing
 - ▶ [JFrame](#)
 - ▶ [ActionListener](#)
 - ▶ [ActionEvent](#)
- 3 Java Swing Widgets
 - ▶ Adding Widgets to Our Screen
 - ▶ Instance Variables
 - ▶ Layout Managers
- 4 Fundamentals: [DemoGUI](#)
- 5 Constructor for View Class
- 6 Recap: MVC Pattern
- 7 Threads