Graphical User Interfaces

Programs With GUIs

- A Java program with a ***GUI***, or ***graphical user interface***, 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...

The User Interaction Problem

- Not just your program, but an end-user, can spontaneously change the "state" of any active user interface widget (e.g., click a button, check a box, move a slider, scroll a document, press a key, etc.)
- ***Problem***: How does your program know *when* the user has attempted to provide input to the program via a widget, and determine *which* widget has been manipulated?

The User

User interaction includes the keyboard—and any other input devices, e.g., a Kinect controller; so, it goes well beyond reading characters using a `SimpleReader`.

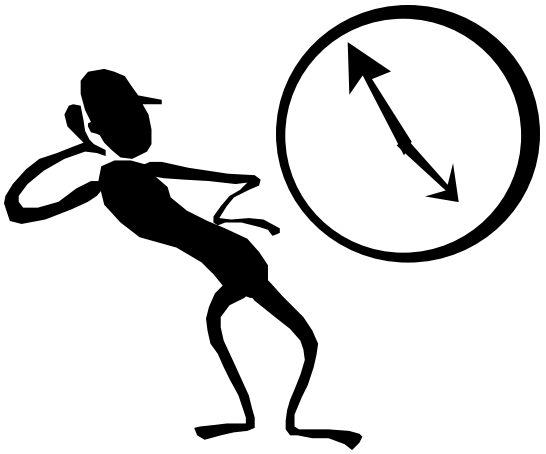
- Not just your r spontaneou user inte ace widget (e.g., click a button, check a box, move a slider, scroll a document, press a key, etc.)
- ***Problem***: How does your program know *when* the user has attempted to provide input to the program via a widget, and determine *which* widget has been manipulated?

Terminology

- The act of a user manipulating a widget is called an ***event*** for that widget
- The widget the user has manipulated is called the ***subject*** of the interaction
- The objects in your program that need to do something in response to the events for a particular subject are called ***observers*** (or ***listeners***) for that subject

Solution #1: Use Polling

subject



observer

- The main program (*the* only observer) continually ***polls*** each possible subject to ask whether any events have occurred
- This is considered cumbersome...

Polling Pseudo-code

```
while (true) {
  if (s0 has experienced an event) {
    if (event is e0) {
      respond to it
    } else if (event is e1) {
      respond to it
    } else ...
  } else if (s1 has experienced an event) {
    ...
  }
}
```

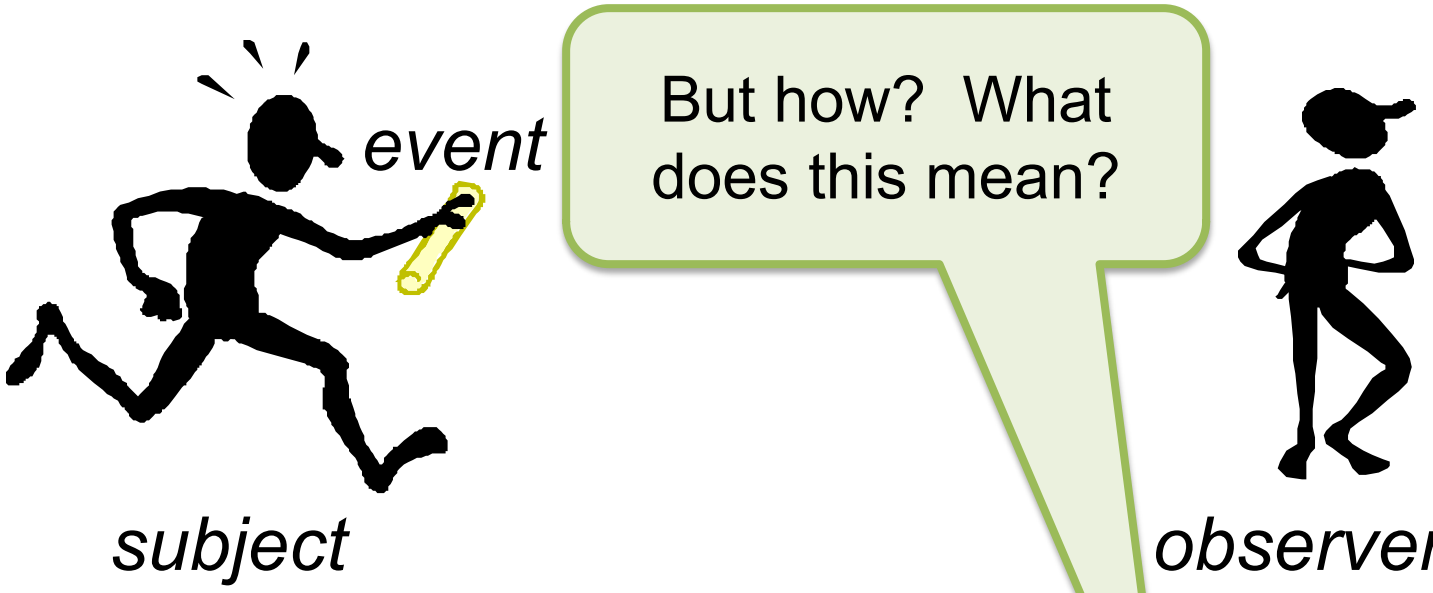
Solution #2: Use Callbacks

subject

observer

- Each observer (there may be many) registers its interest in a subject's events, and then waits until the subject ***calls it back*** to tell it that there has been an event

Solution #2: Use Callbacks



- Each observer (there may be many) registers its interest in a subject's events, and then waits until the subject ***calls it back*** to tell it that there has been an event

The Observer Pattern

- Each subject expects each observer (listener) to ***register*** itself with that subject if it is interested in the subject's events
- Each subject keeps track of its own ***set of interested observers***
- Whenever an event occurs, the subject invokes a specific ***callback method*** for *each* registered observer, passing an ***event*** argument that describes the event

The Observer Pattern

- Each subject expects each observer (listener) to ***register*** itself with that subject if it is interested in t[illegible] subject's events
- Each subject keeps[illegible] ***interested observ***
- Whenever an even[illegible] invokes a specific [illegible] *each* registered observer, passing an ***event*** argument that describes the event

Registering interest is done by calling a method of the subject; usually this is done once as part of set-up.

The Obse

- Each subject expe
(listener) to ***registe***
if it is interested in the subject's eve
- Each subject keeps track of its own ***set of interested observers***
- Whenever an event occurs, the subject invokes a specific ***callback method*** for *each* registered observer, passing an ***event*** argument that describes the event

The set of observers for a given subject can be kept in a `Set` variable, for example.

The Observer Pattern

- Each subject expe
(listener) to ***regist***
if it is interested in th
- Each subject keeps trac ts own ***set of interested observers***
- Whenever an event occurs, the subject invokes a specific ***callback method*** for *each* registered observer, passing an ***event*** argument that describes the event

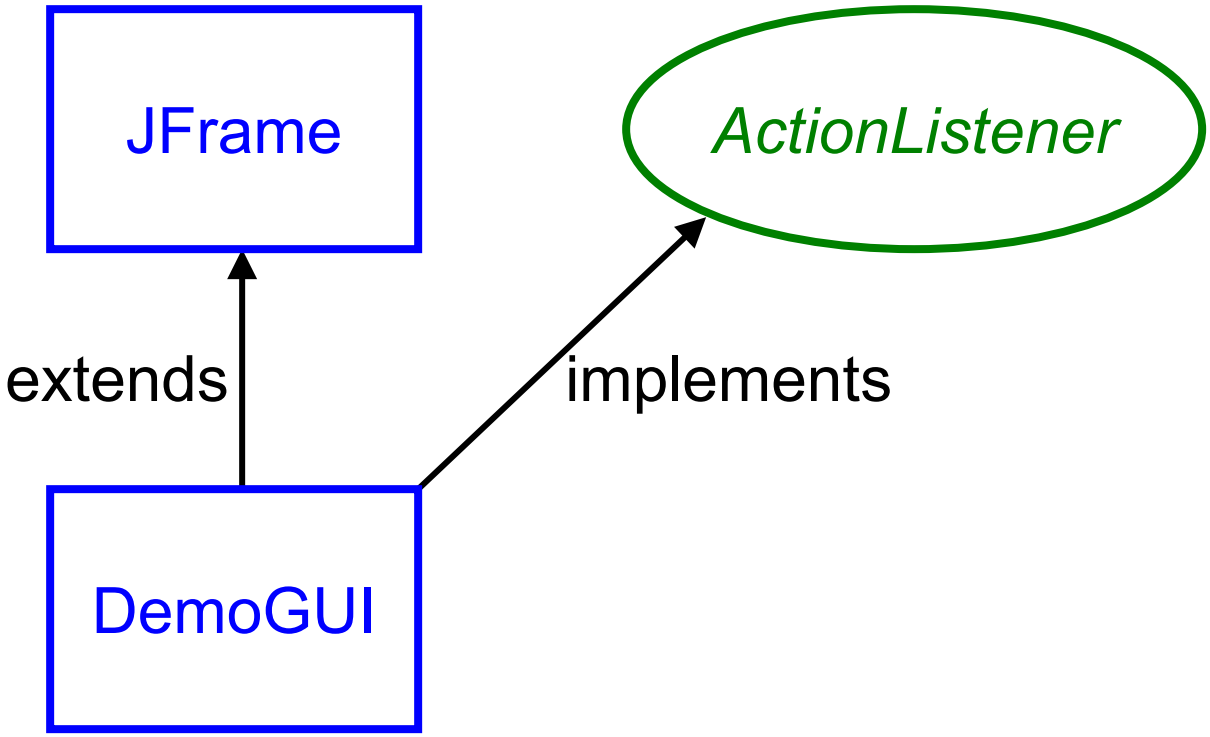
This method is described in an interface that any potential observer must implement.

The Observer Pattern

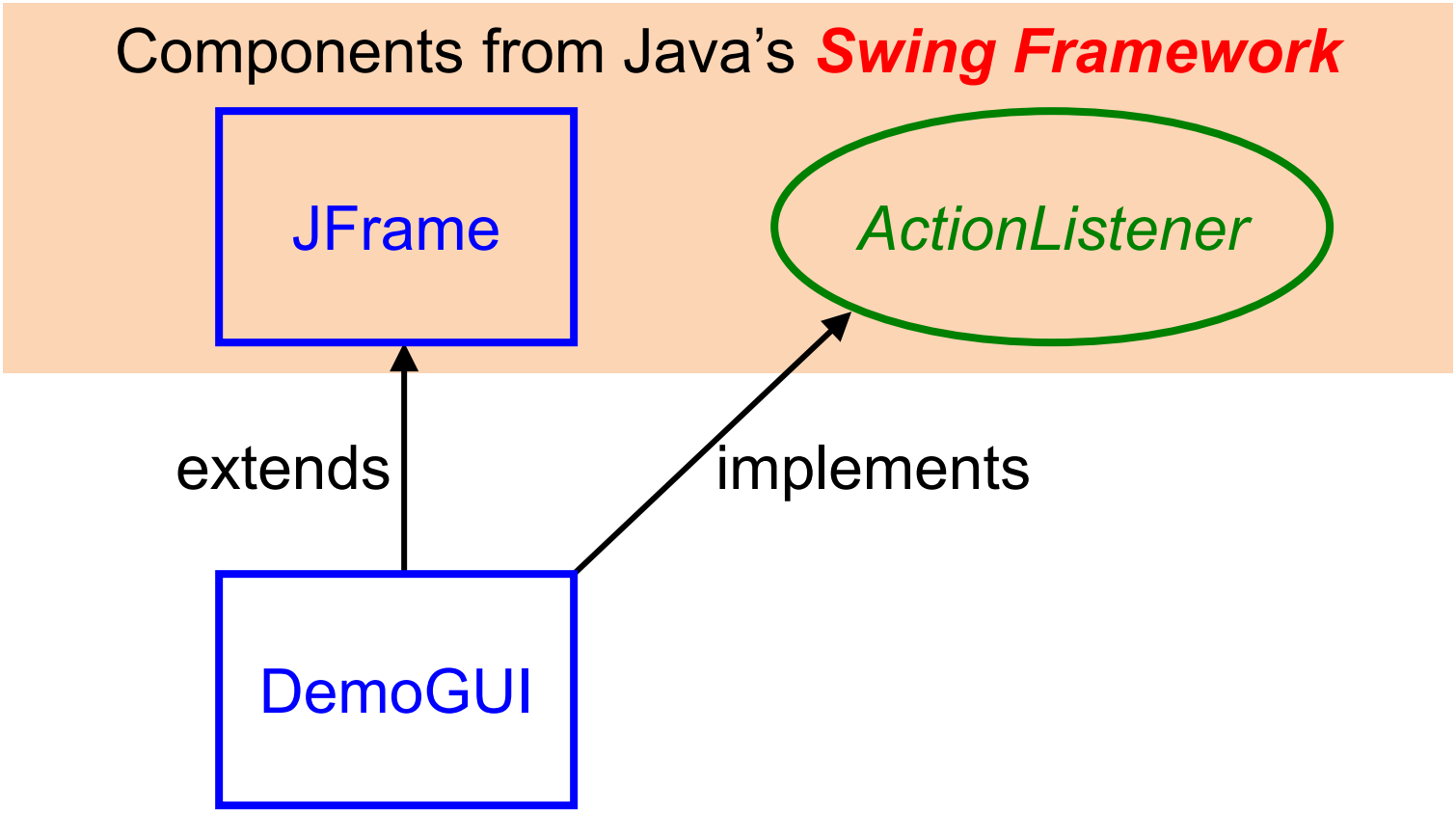
- Each subject
(listener) to
if it is interes
- Each subject
interested c
- Whenever an event occurs, the subject invokes a specific ***callback method*** for *each* registered observer, passing an ***event*** argument that describes the event

This is one of many ***object-oriented design patterns*** that address common OOP issues (often language deficiencies); most are considered **best practices**.

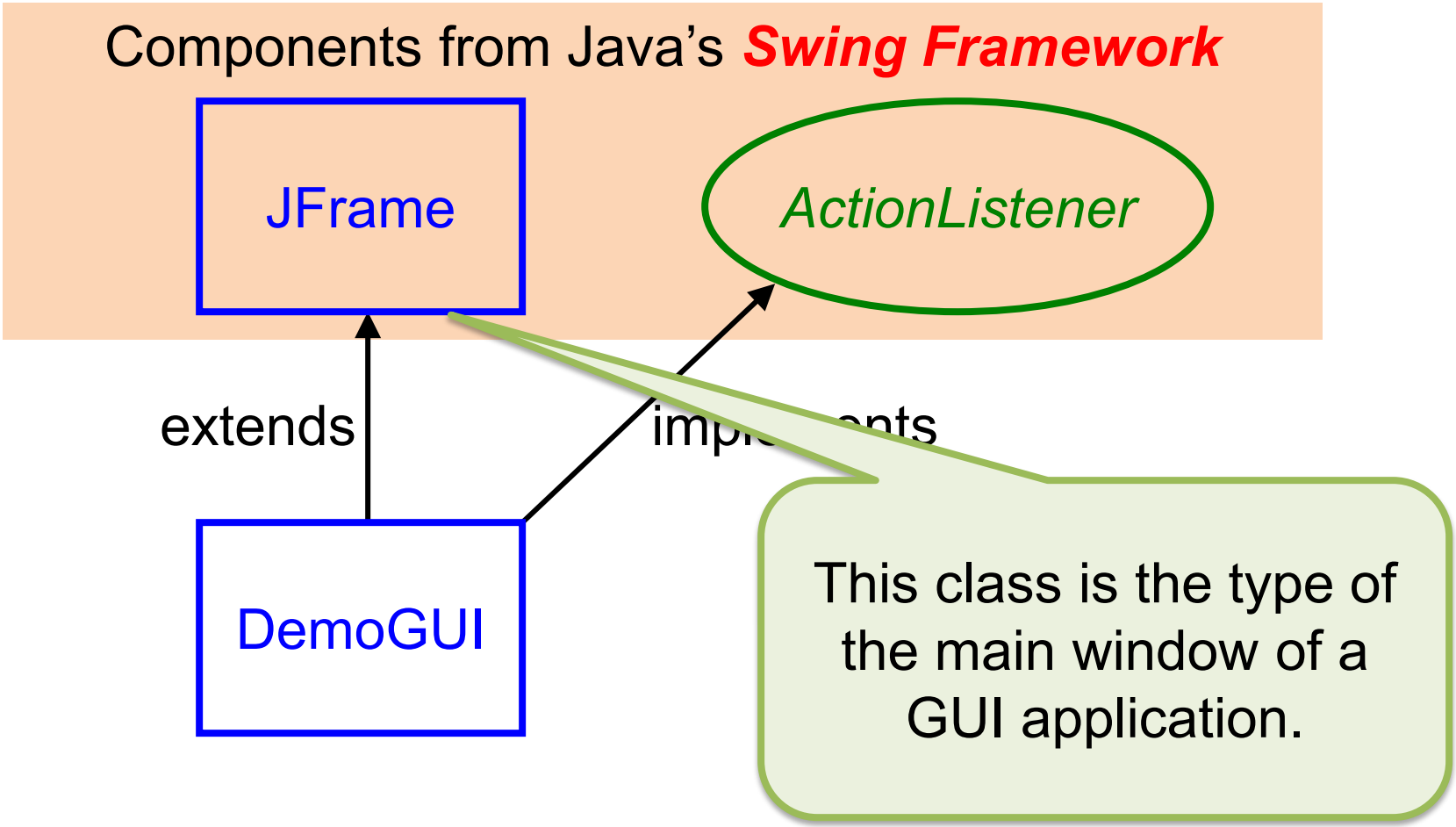
Example: Simple GUI Demo



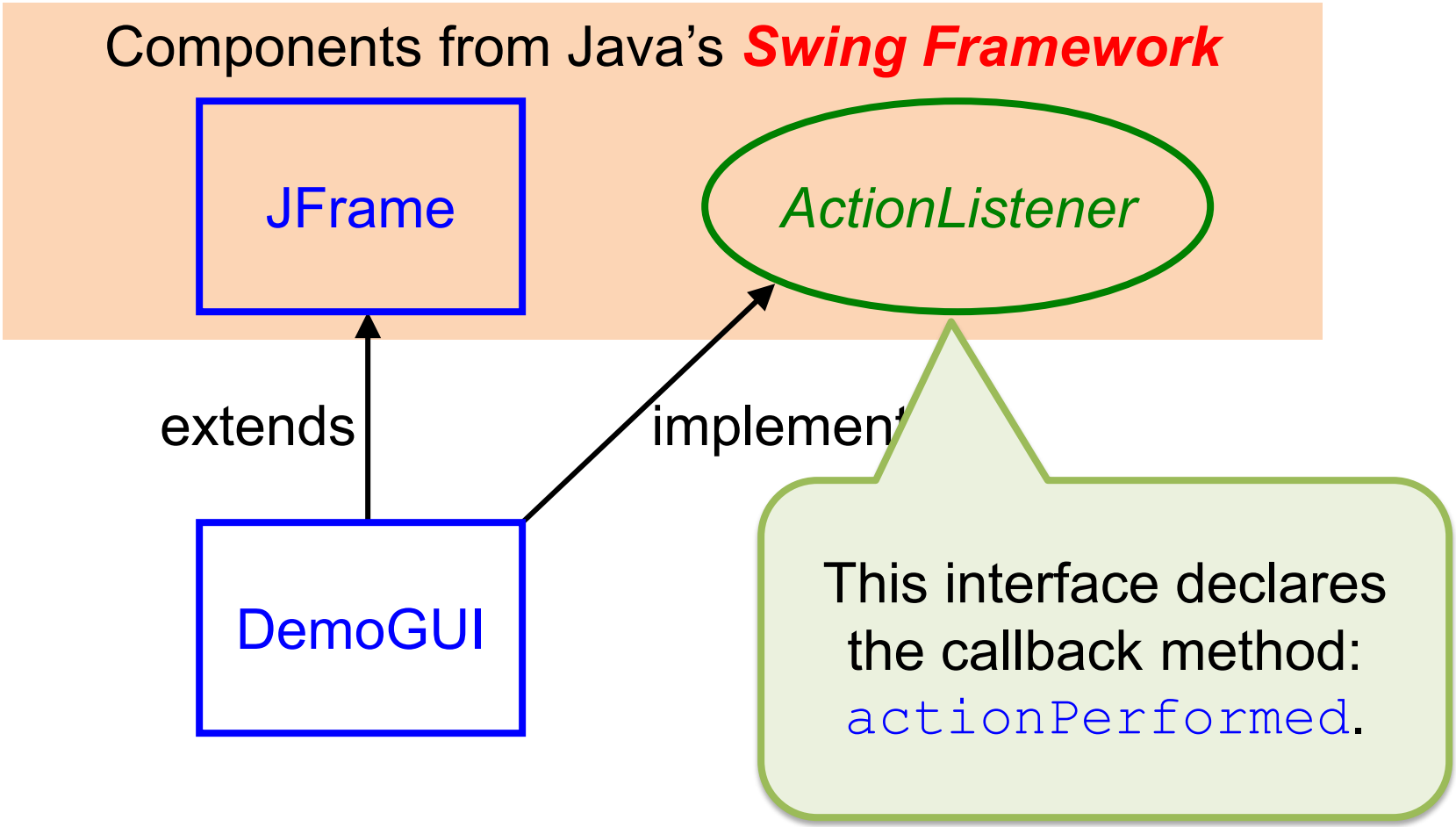
Example: Simple GUI Demo



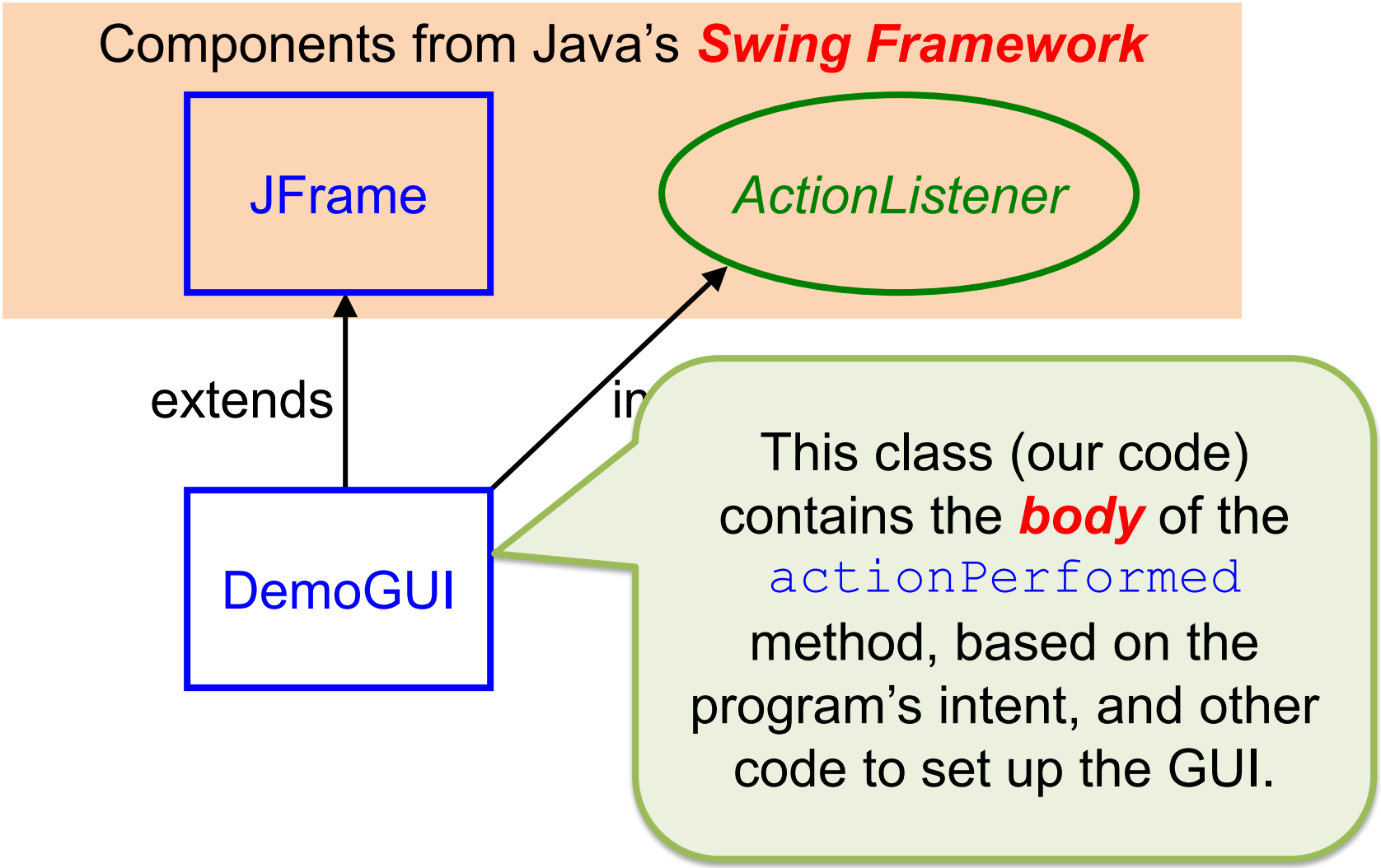
Example: Simple GUI Demo



Example: Simple GUI Demo



Example: Simple GUI Demo



Important Interfaces/Methods

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

interface ActionEvent {
   Object getSource();
   ...
}
```

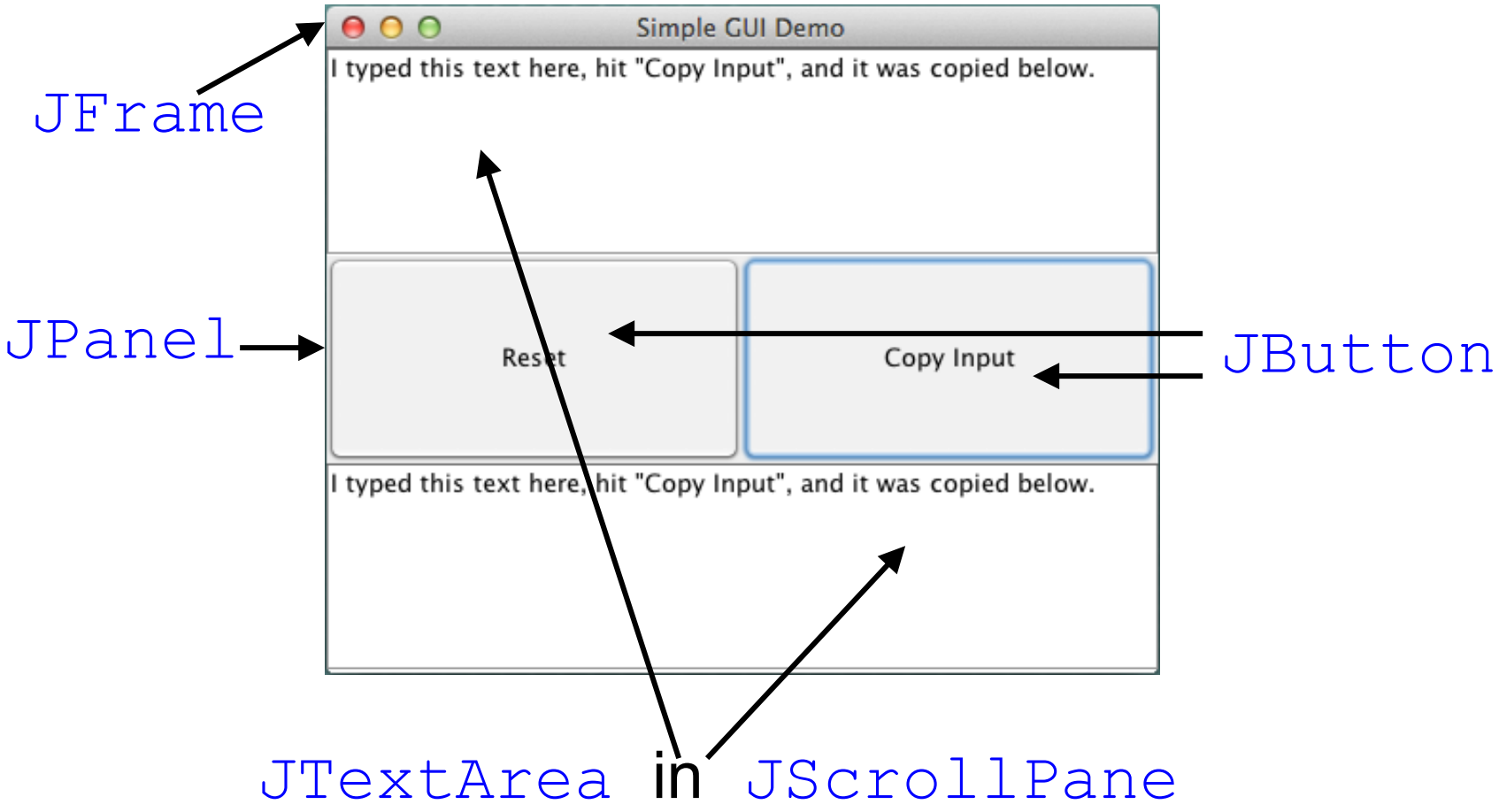
Methods

The class `Object` is special in Java: ***every class*** extends `Object`! We will return to this later...

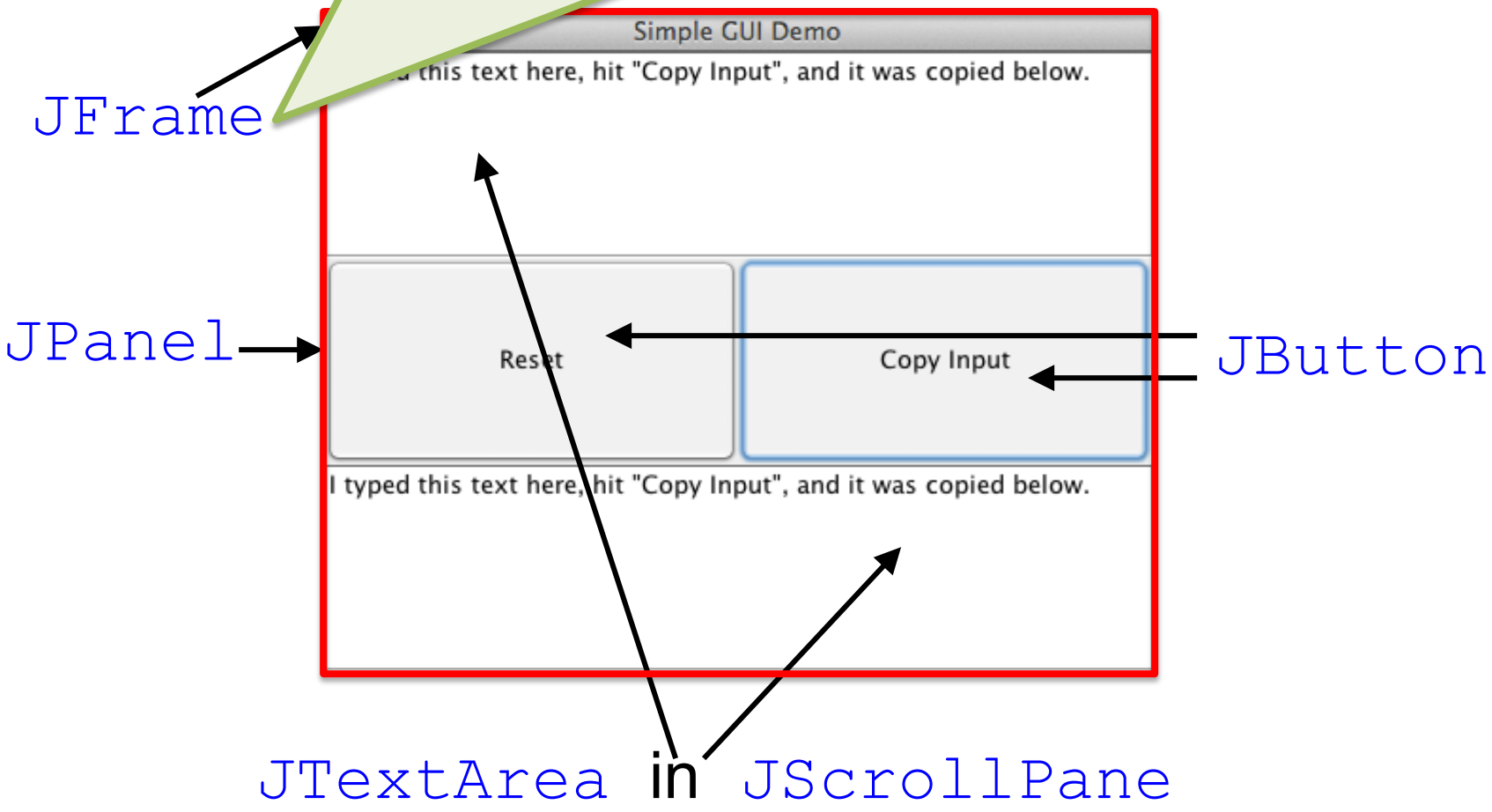
```
onEvent e);
}

interface ActionEvent {
   Object getSource();
   ...
}
```

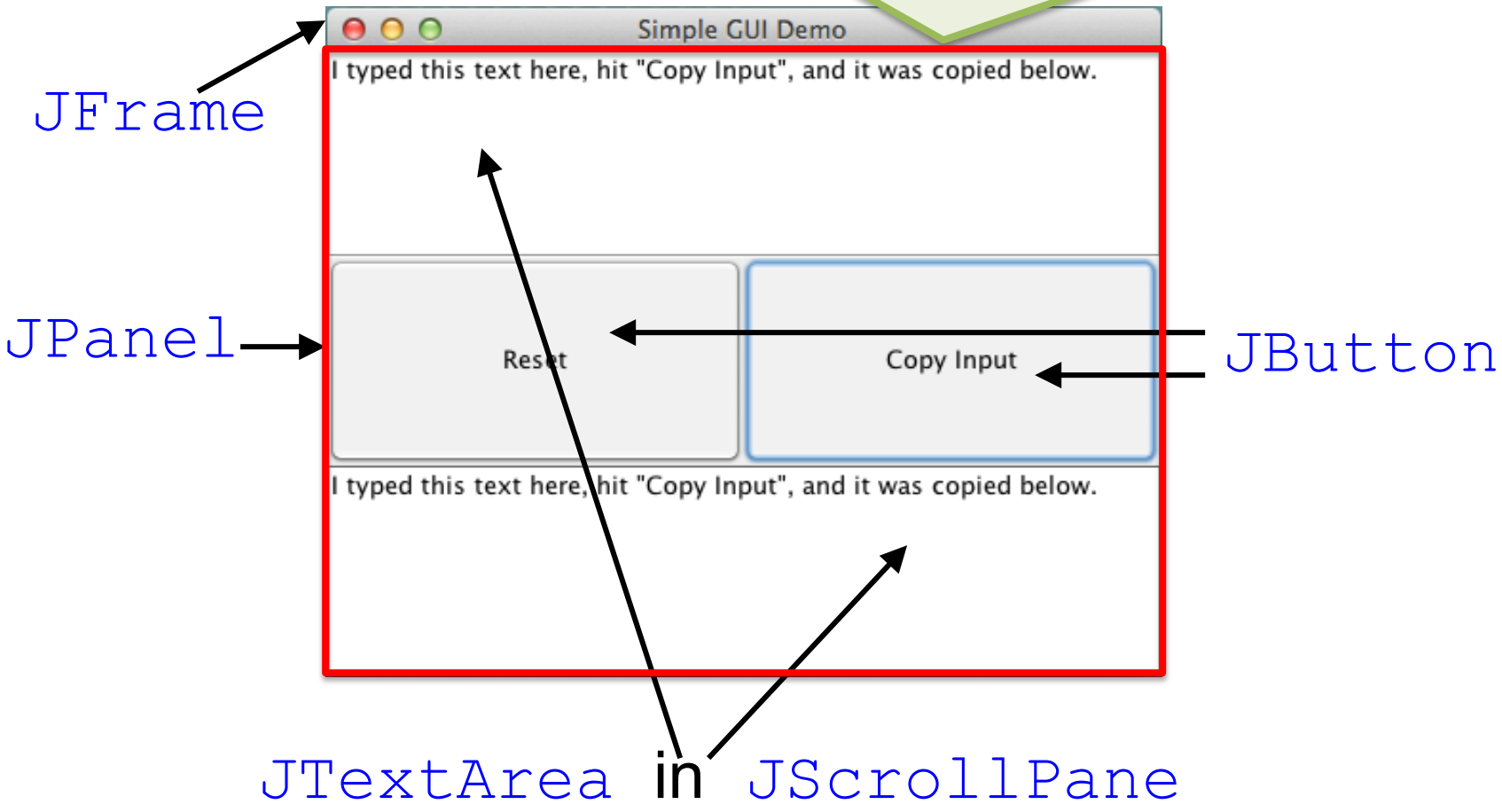
Fundamentals: DemoGUI

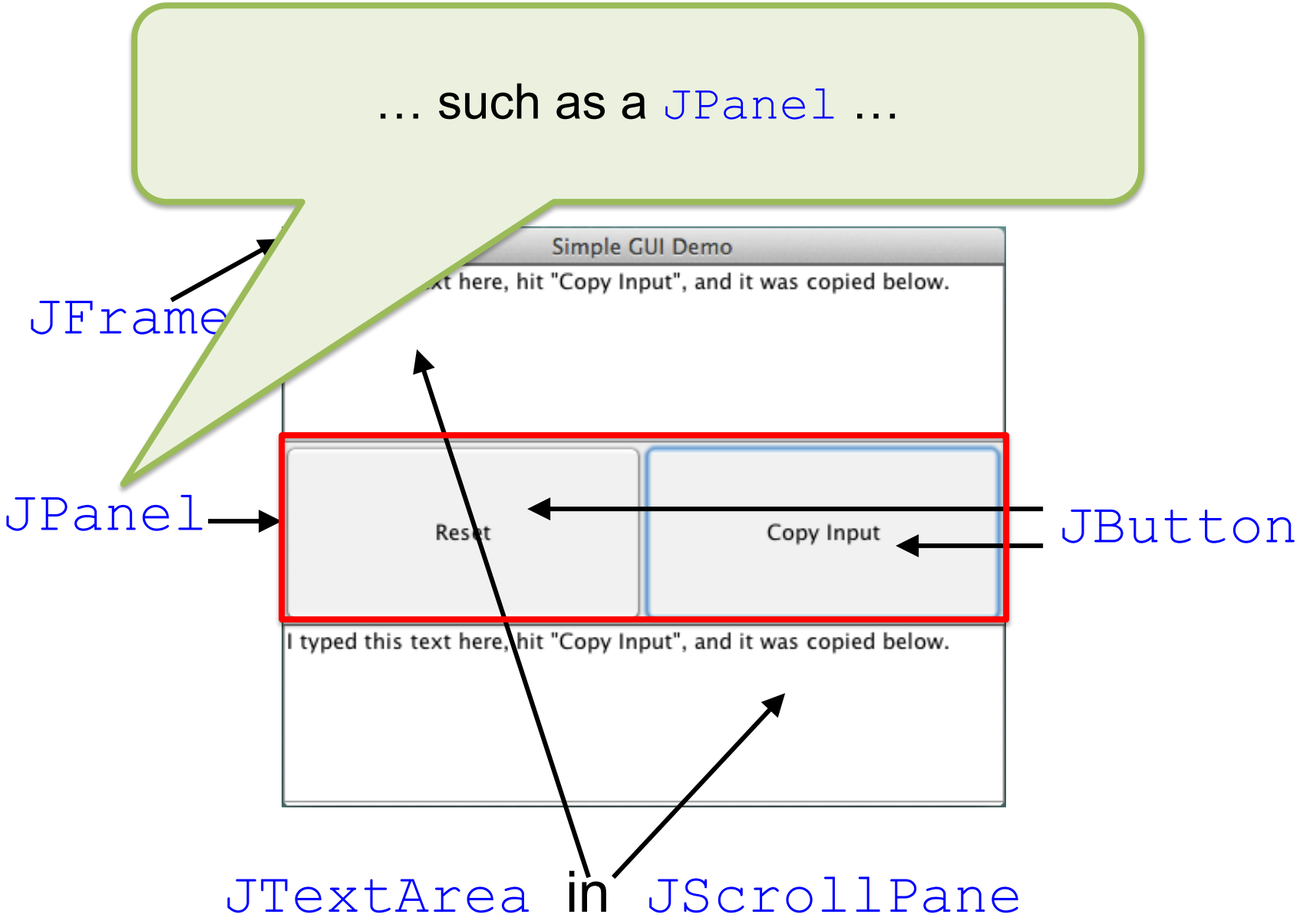


This is the underlying type of the main window of a GUI application using Swing.

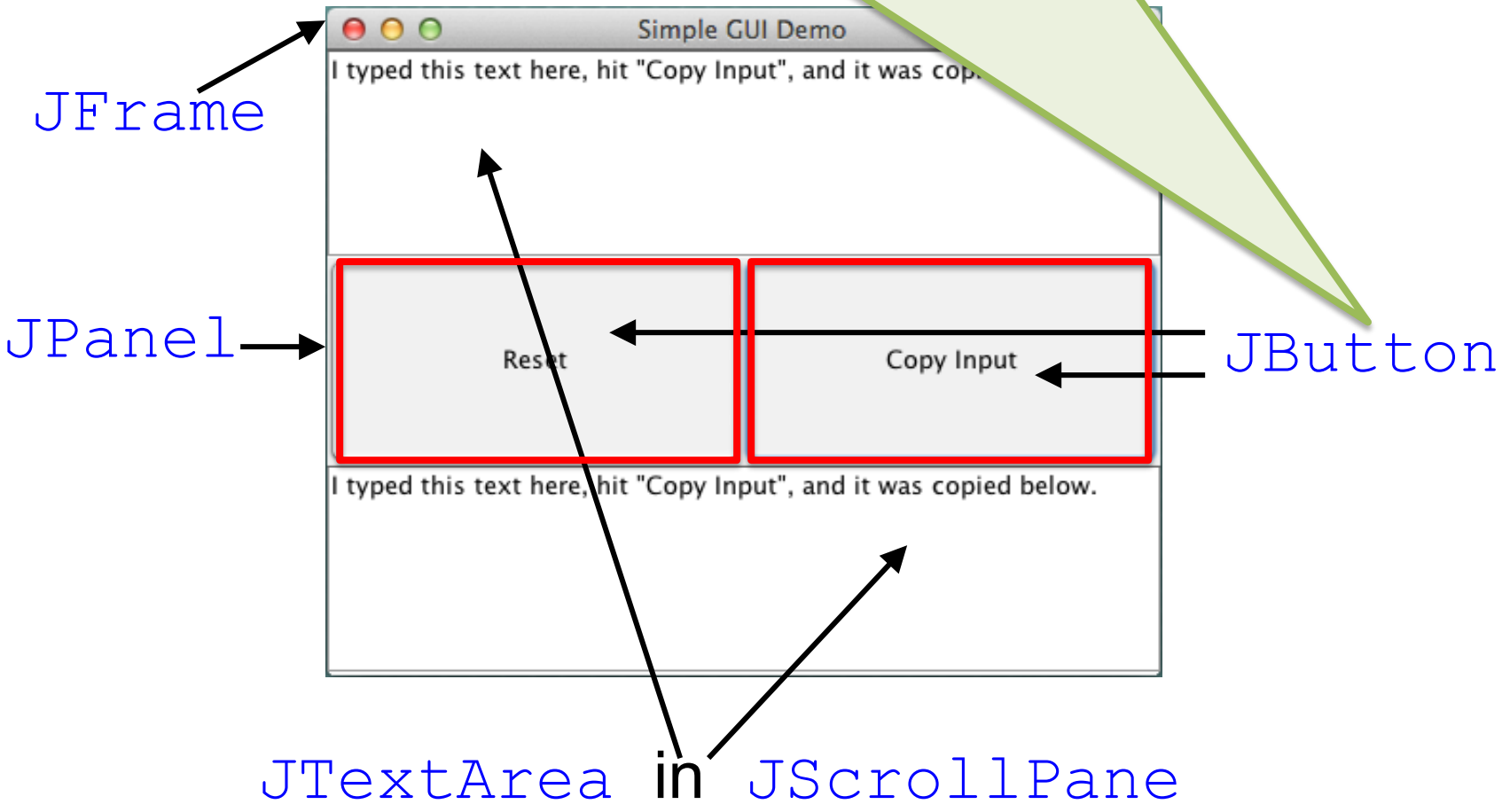


Nested inside a `JFrame`'s ***content pane***, you can put any number of things …

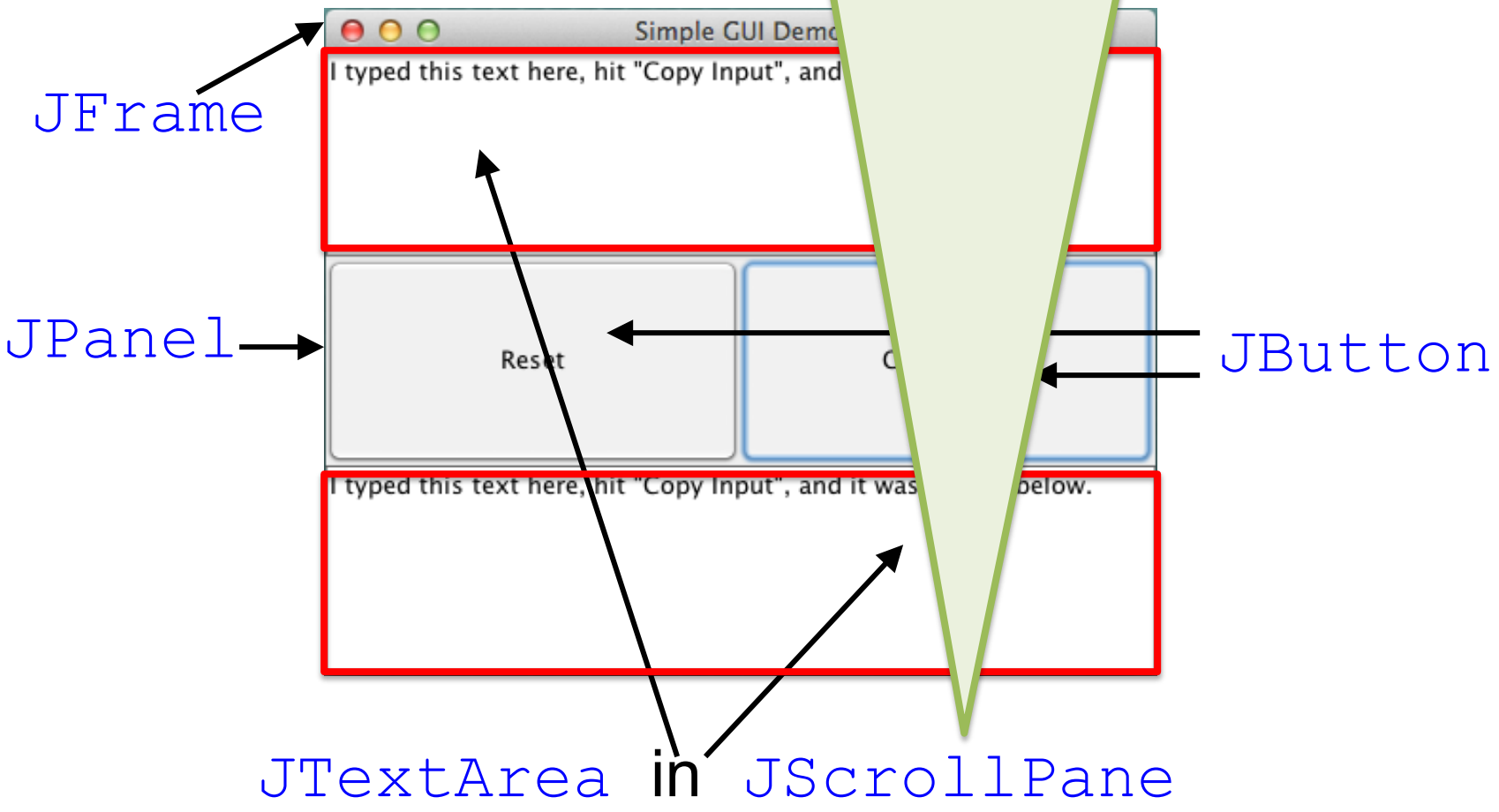



… such as a JPanel …
JFrame
JPanel
JButton
JTextArea in JScrollPane
Simple GUI Demo
here, hit "Copy Input", and it was copied below.
Reset
Copy Input
I typed this text here, hit "Copy Input", and it was copied below.

… and nested inside a JPanel, you can put any number of, e.g., JButtons.



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



It's Demo Time

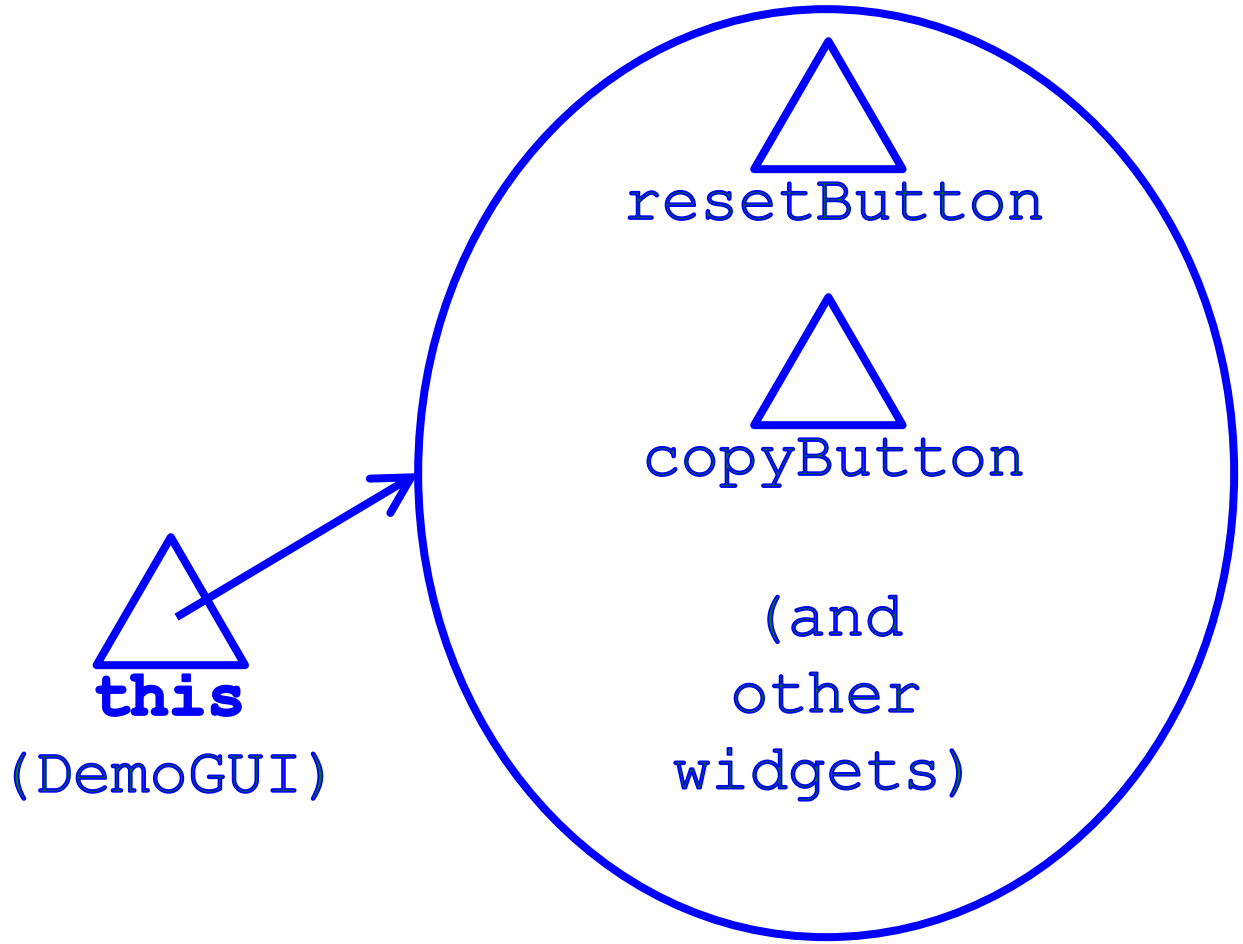
- The `DemoGUI1` project contains a very simple GUI application using Swing
- You can get it at:

  http://web.cse.ohio-state.edu/software/common/DemoGUI1.zip

Instance Variables

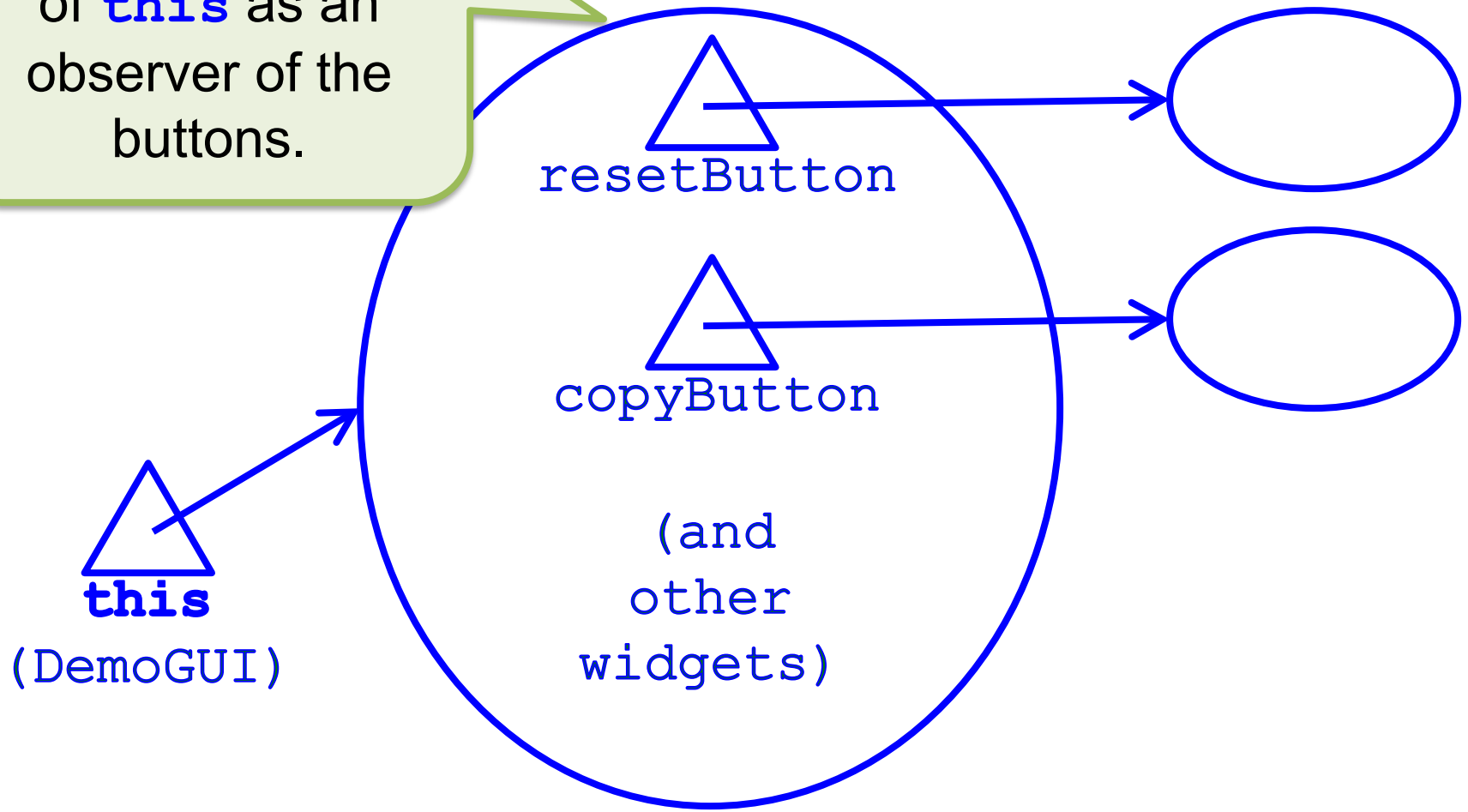
- Variables can be declared:
  - in method bodies: ***local variables***
  - in method headers: ***formal parameters***
  - in classes: ***fields*** or ***instance variables***
- Examples of instance variables:
  - resetButton, copyButton, inputText, outputText, input, output
- Instance variables are essentially *global* variables that are shared by and can be accessed from all instance methods in the class

Set Up by `DemoGUI` Constructor



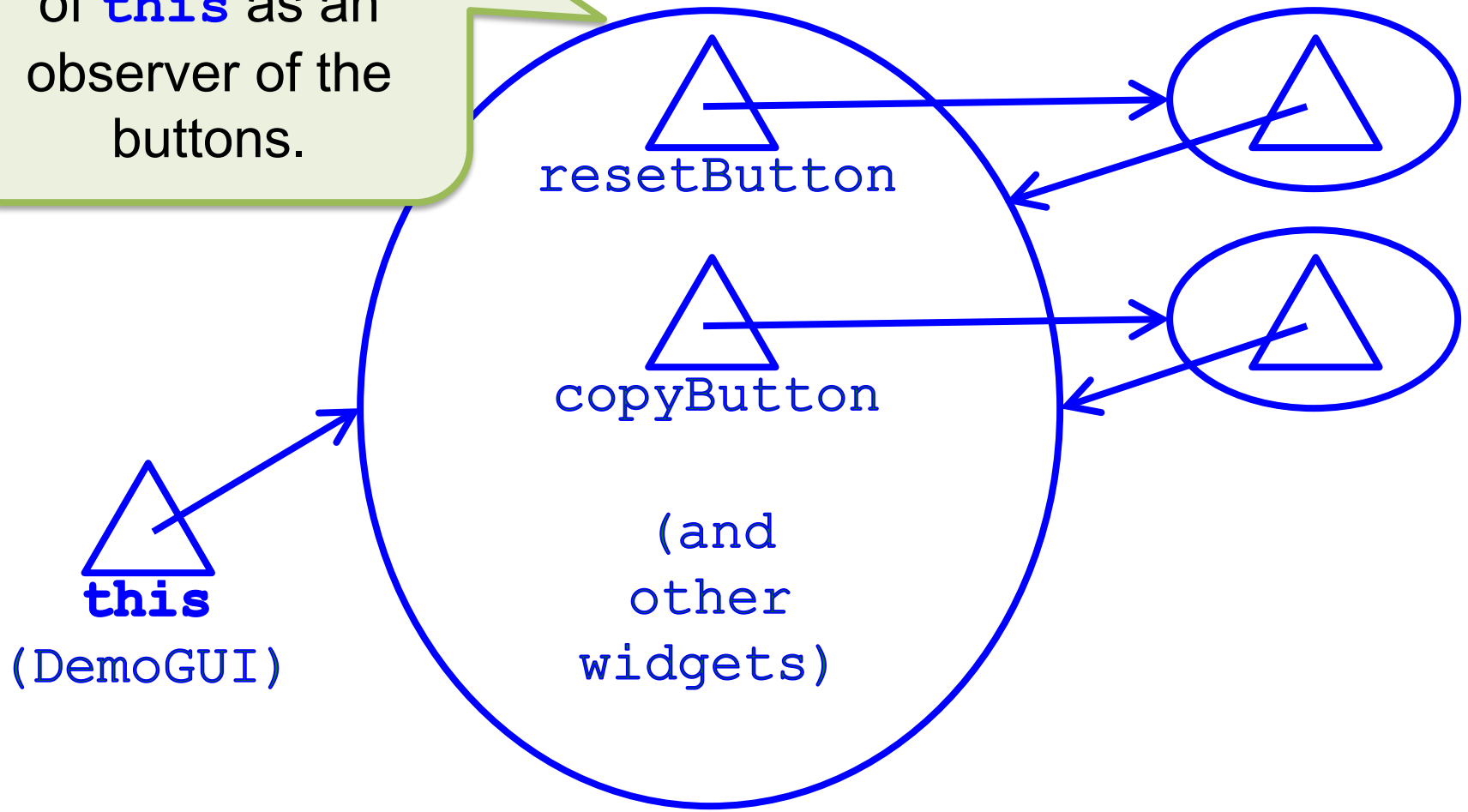
DemoGUI Constructor

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



DemoGUI Constructor

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



Now, Who's In Charge?

- Note: when `DemoGUI` is executed:
  - `DemoGUI.main` starts execution
    - Constructor for `DemoGUI` is called by `main`
    - Constructor for `DemoGUI` returns to `main`
  - `DemoGUI.main` finishes execution
- After that, what code is executing?

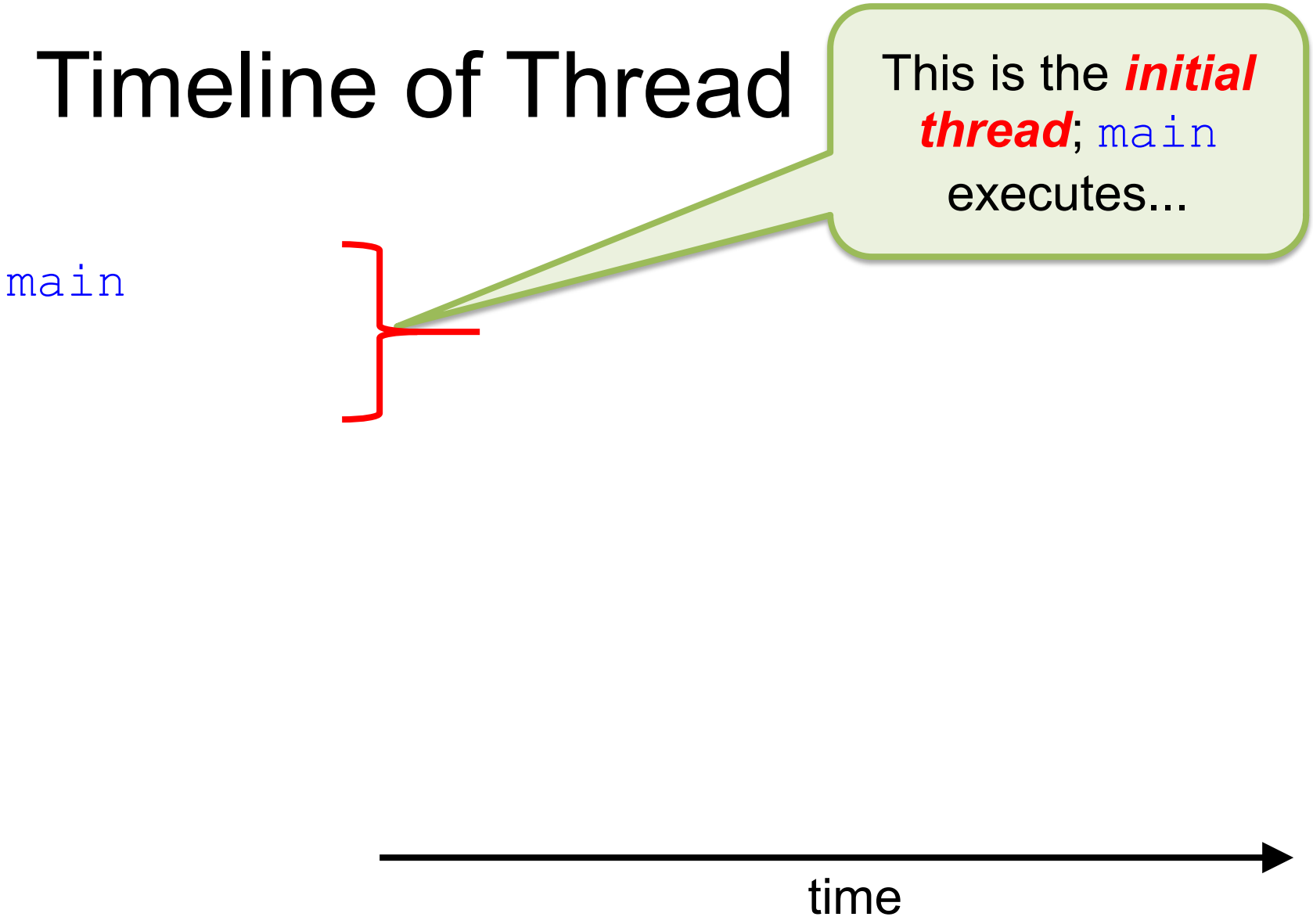
Threads

- 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`

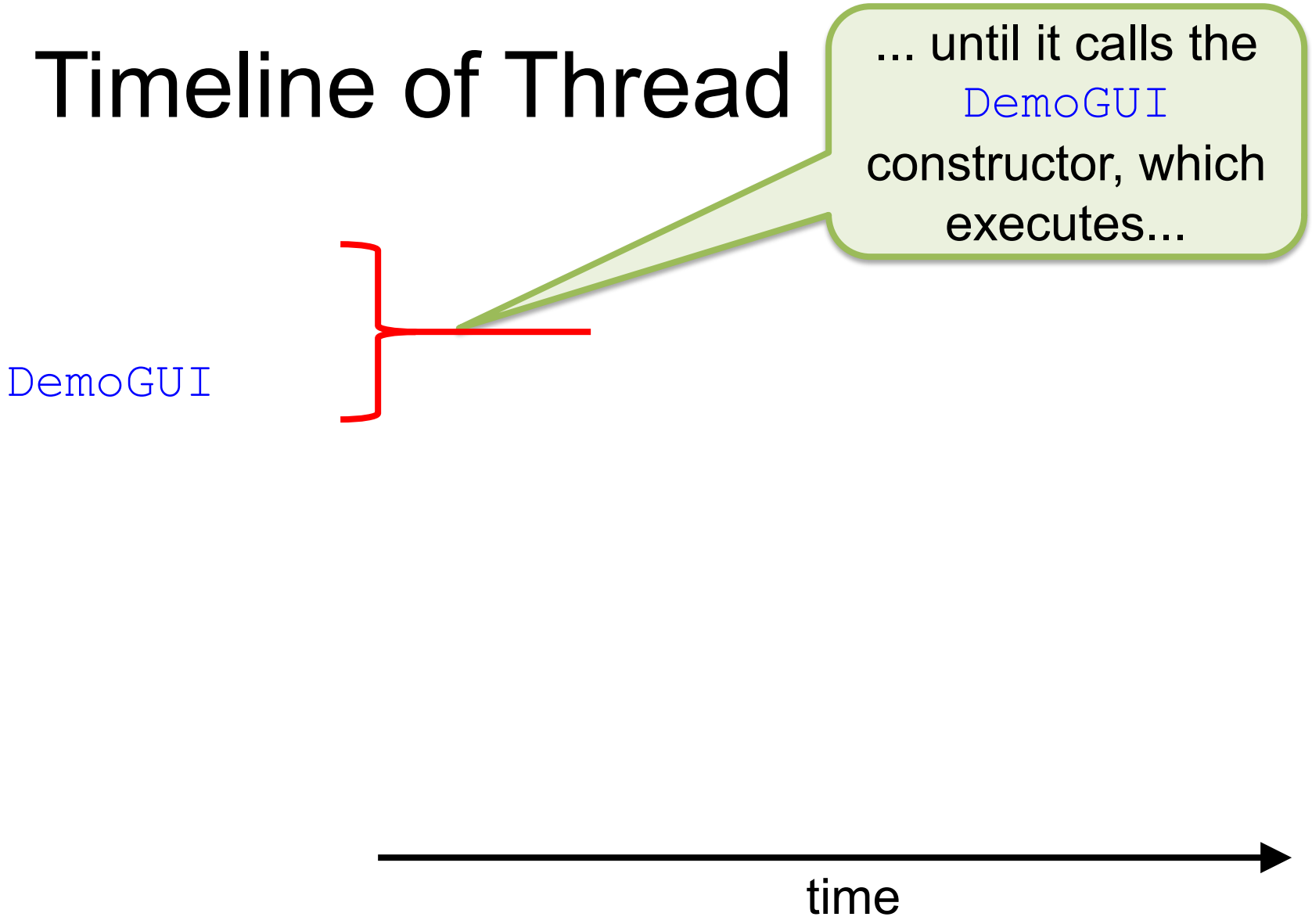
Timeline of Thread Execution

`main`

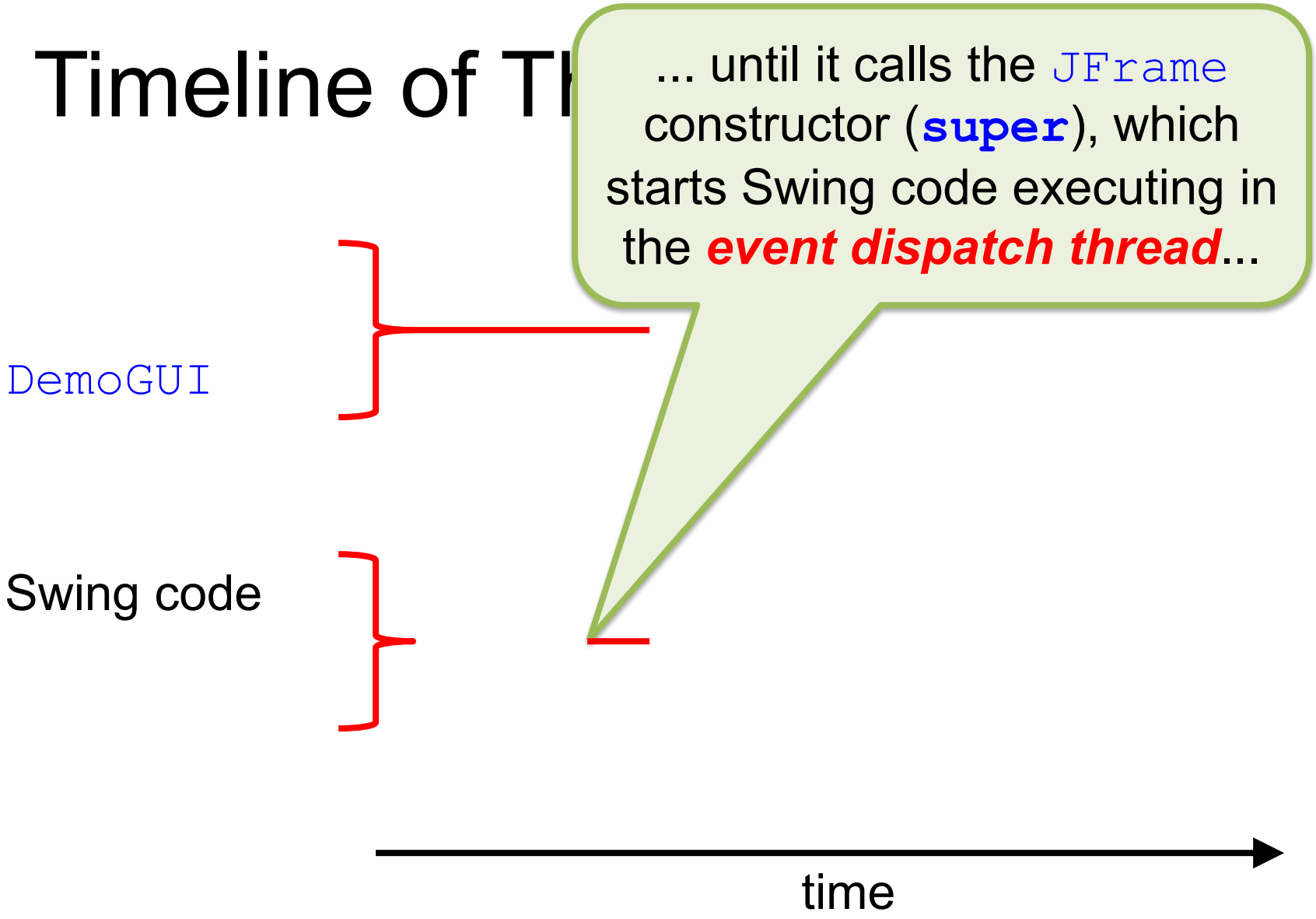
Timeline of Thread



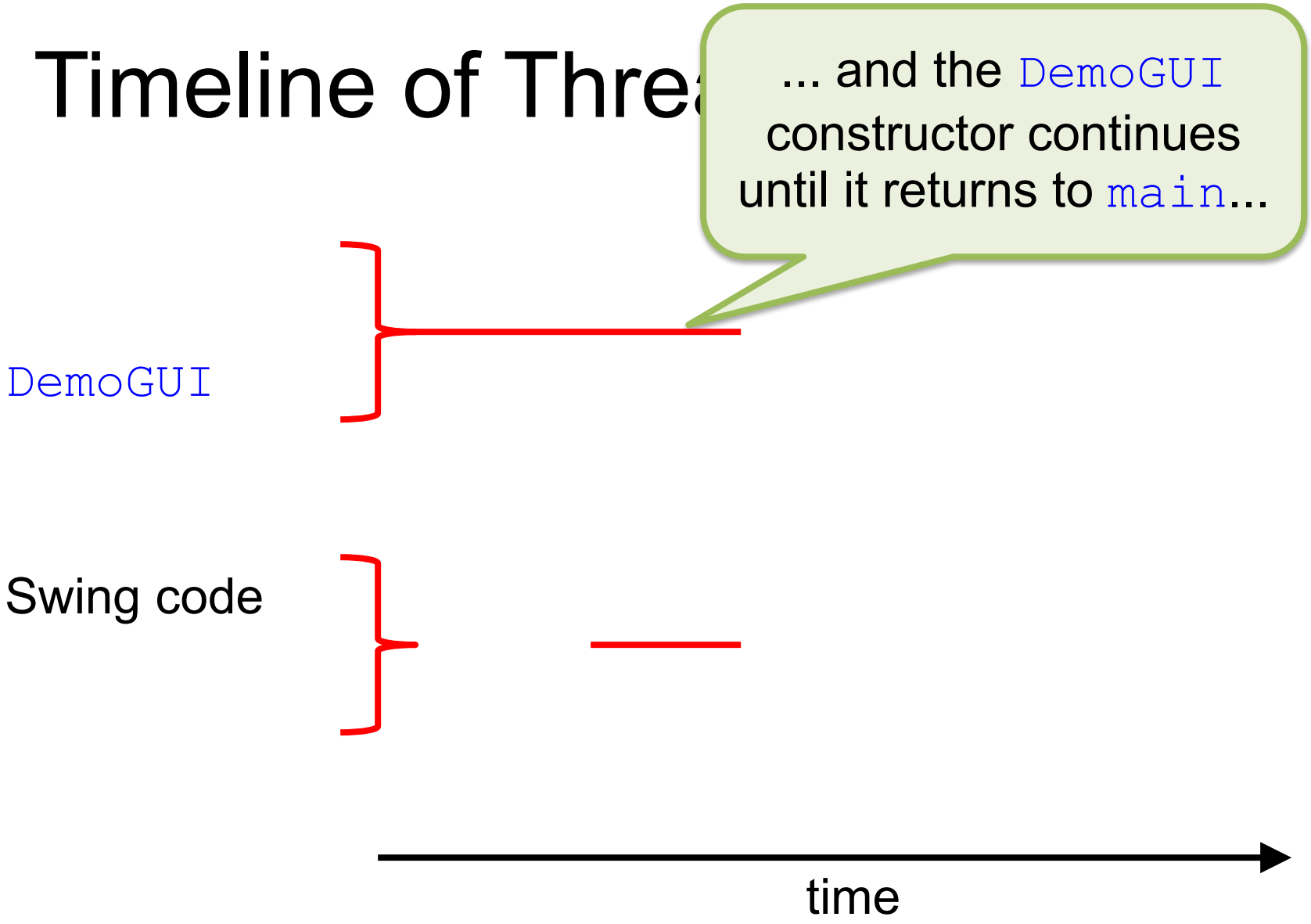
Timeline of Thread



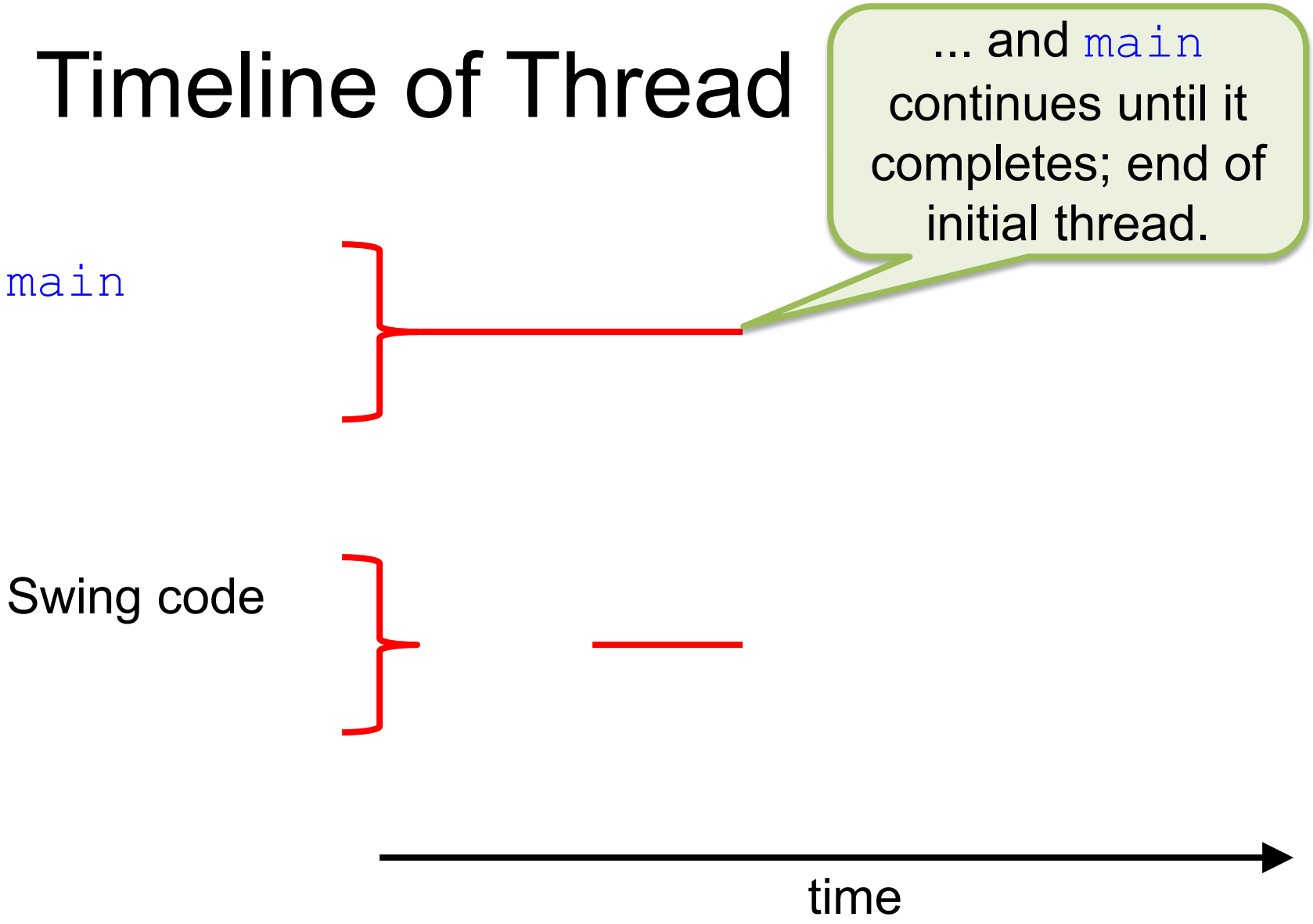
Timeline of Th



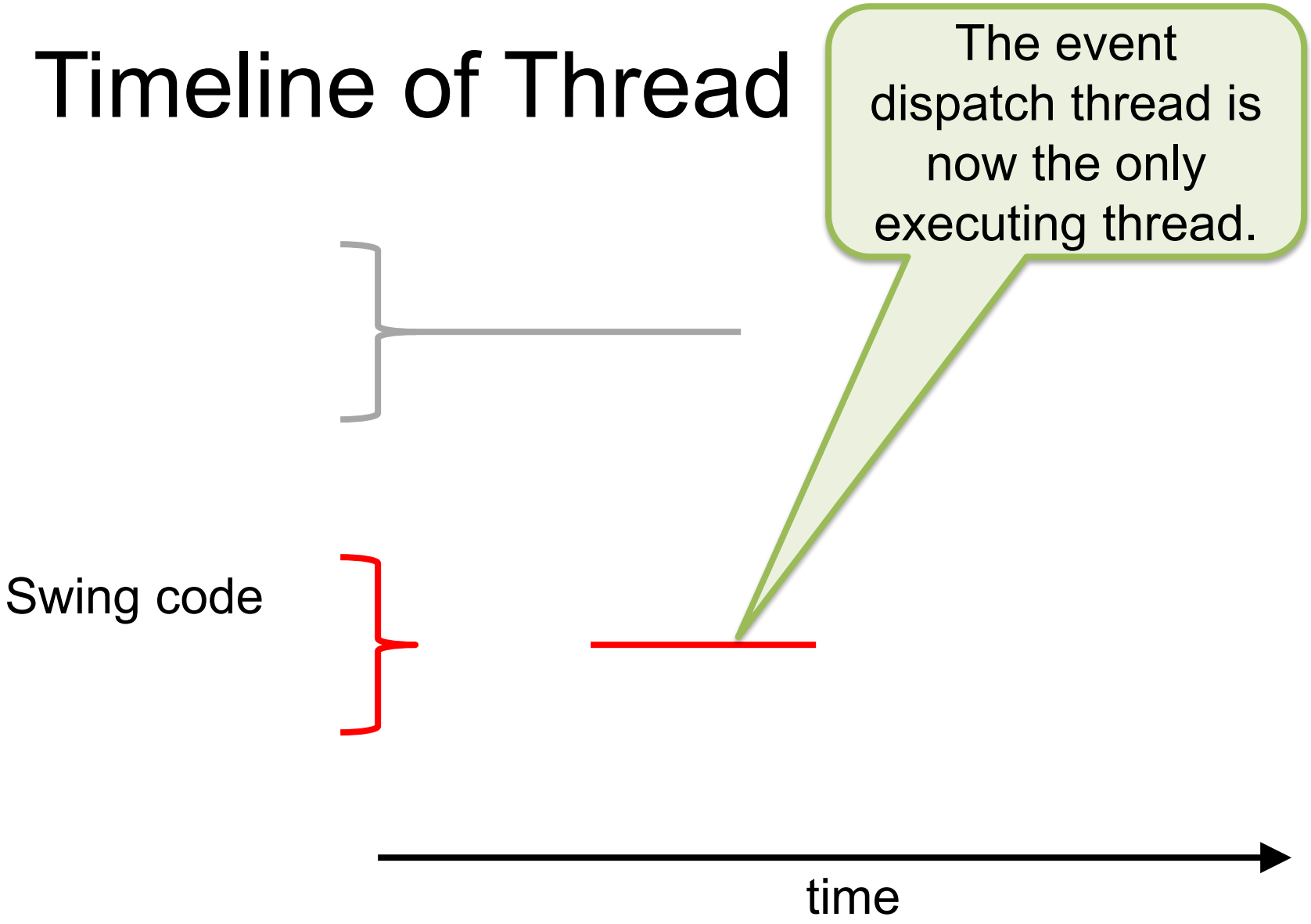
Timeline of Thre



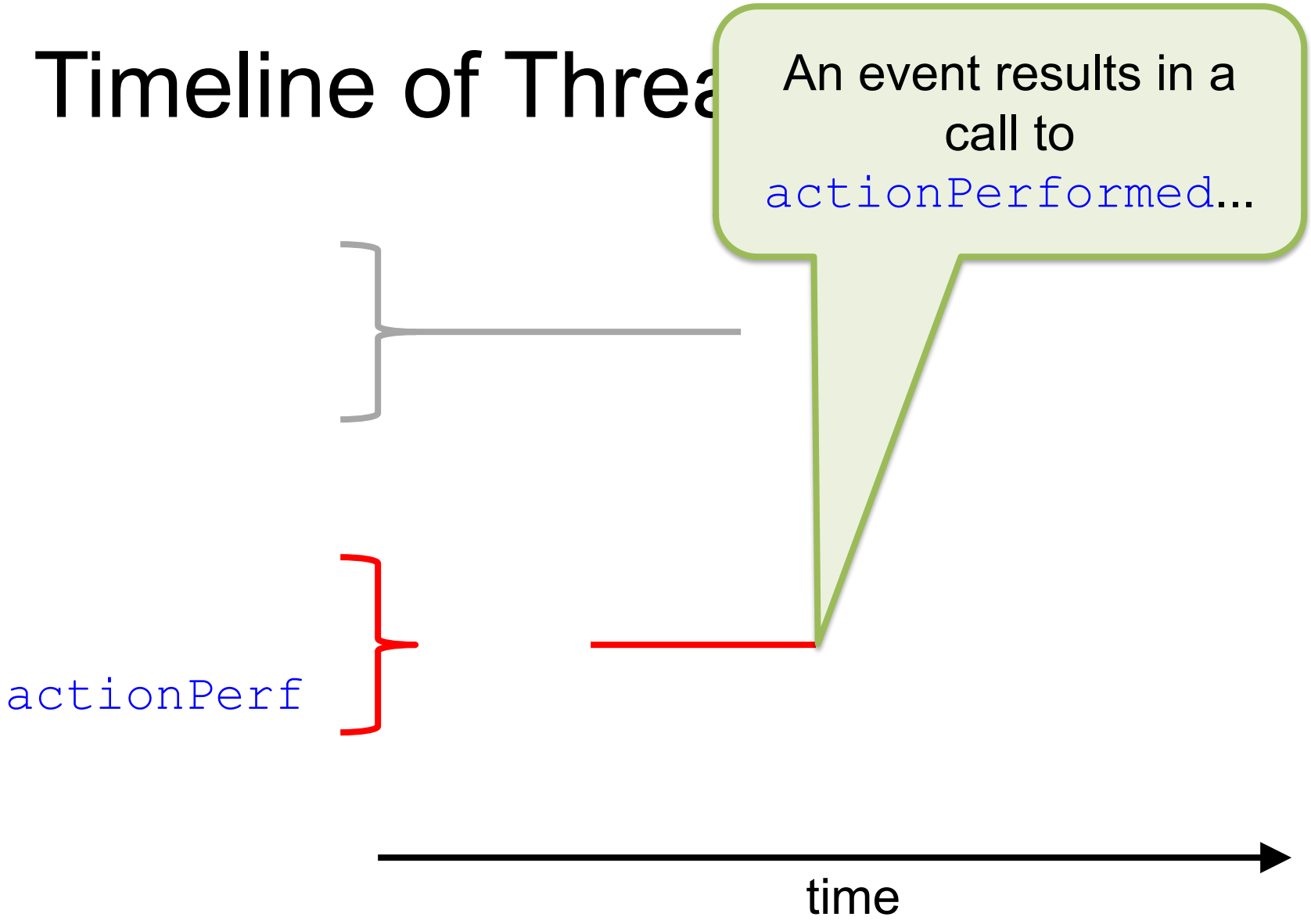
Timeline of Thread



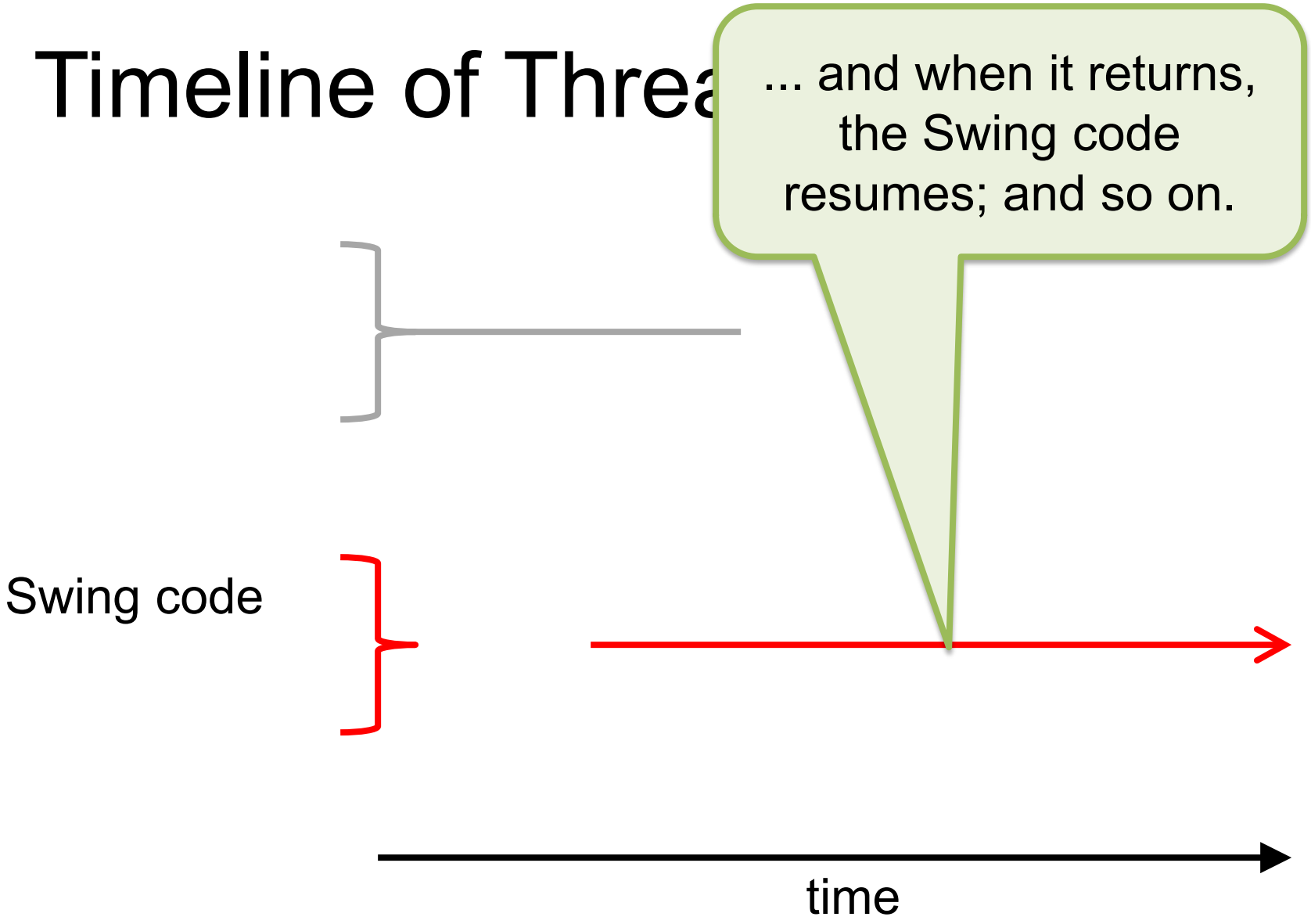
Timeline of Thread



Timeline of Threa



Timeline of Threa



Layout Managers

- A ***layout manager*** allows you to arrange widgets without providing specific location coordinates
  - `GridLayout` (simplest?; used in `DemoGUI`)
  - `FlowLayout` (default for `JPanel`)
  - `BorderLayout` (default for `JFrame`)
  - ...

Java GUI Packages

- Some important packages in the Java libraries for GUI components:
  - `java.awt`
  - `java.awt.event`
  - `javax.swing`
  - `...`

Java Swing Widgets

- Some important classes in `javax.swing`:
  - `JFrame`
  - `JPanel`
  - `JButton`
  - `JScrollPane`
  - `JTextArea`
  - `JCheckBox`
  - `JComboBox`
  - `...`

Resources

- Java Tutorials (and beyond...)
  - http://docs.oracle.com/javase/tutorial/uiswing/index.html
- A Visual Guide to Layout Managers
  - http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html