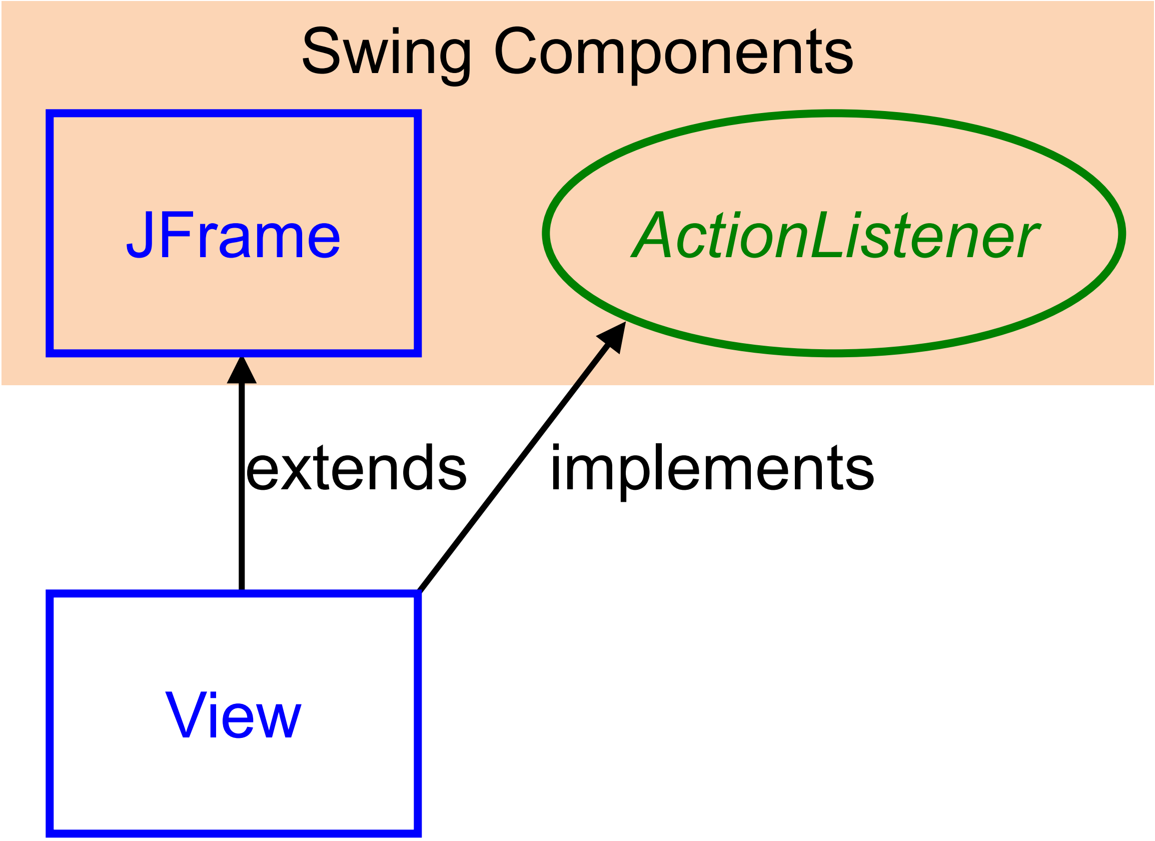
Model-View-Controller

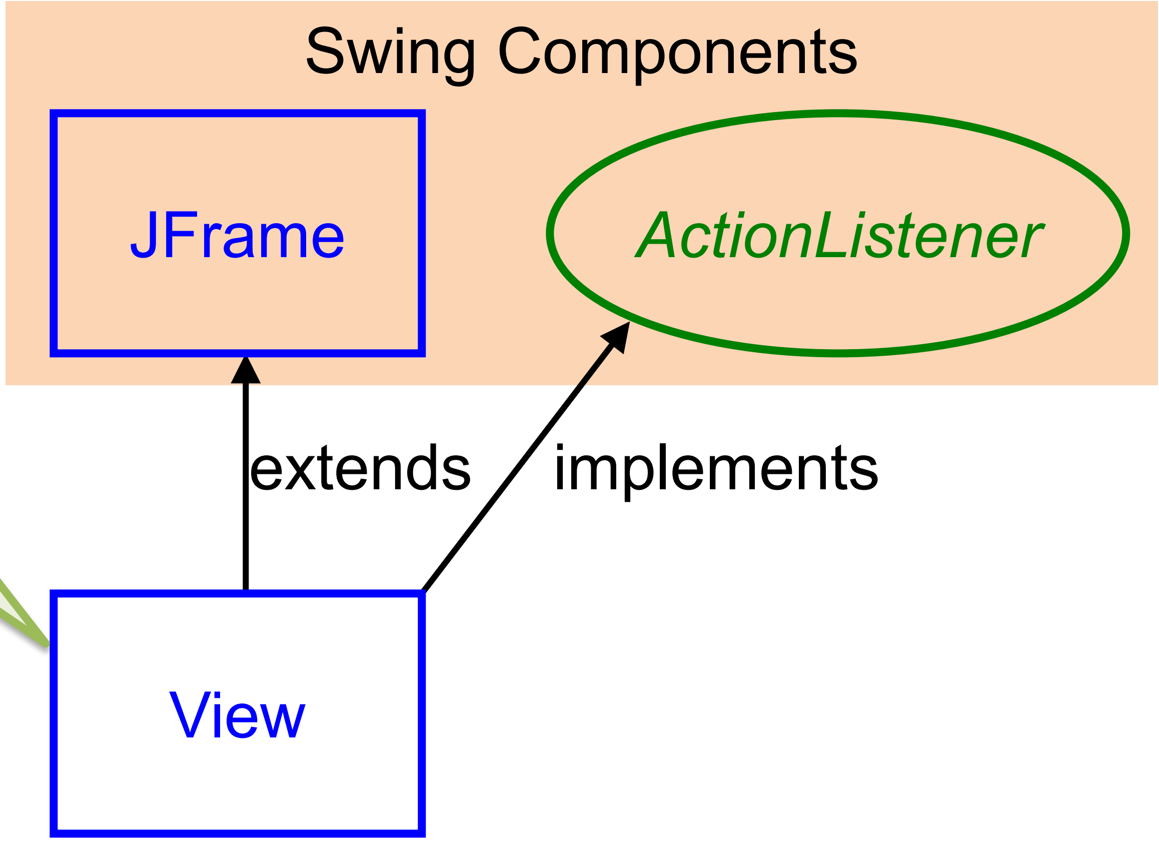
MVC Design Pattern

- The dominant approach to organizing software with GUIs is the ***model-view-controller design pattern***
- There are several versions of this design 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

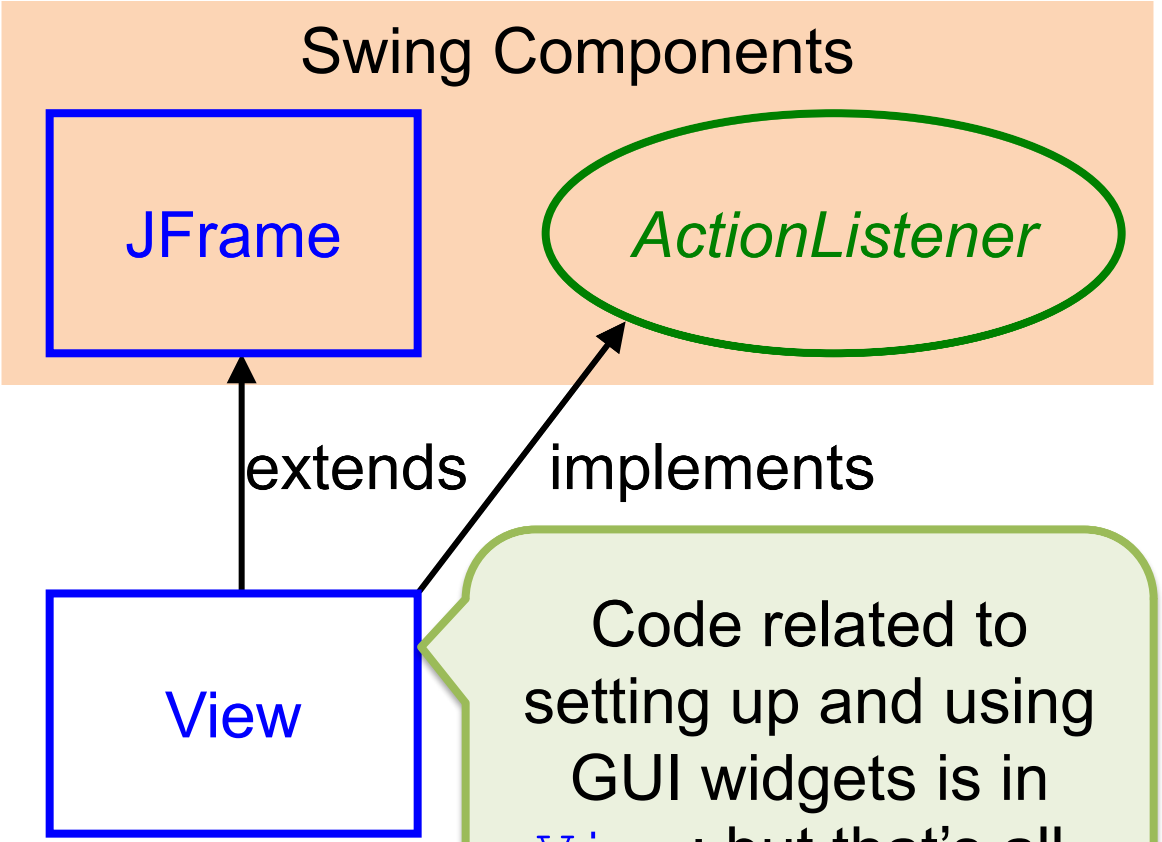


Example: Simple MVC GUI Demo



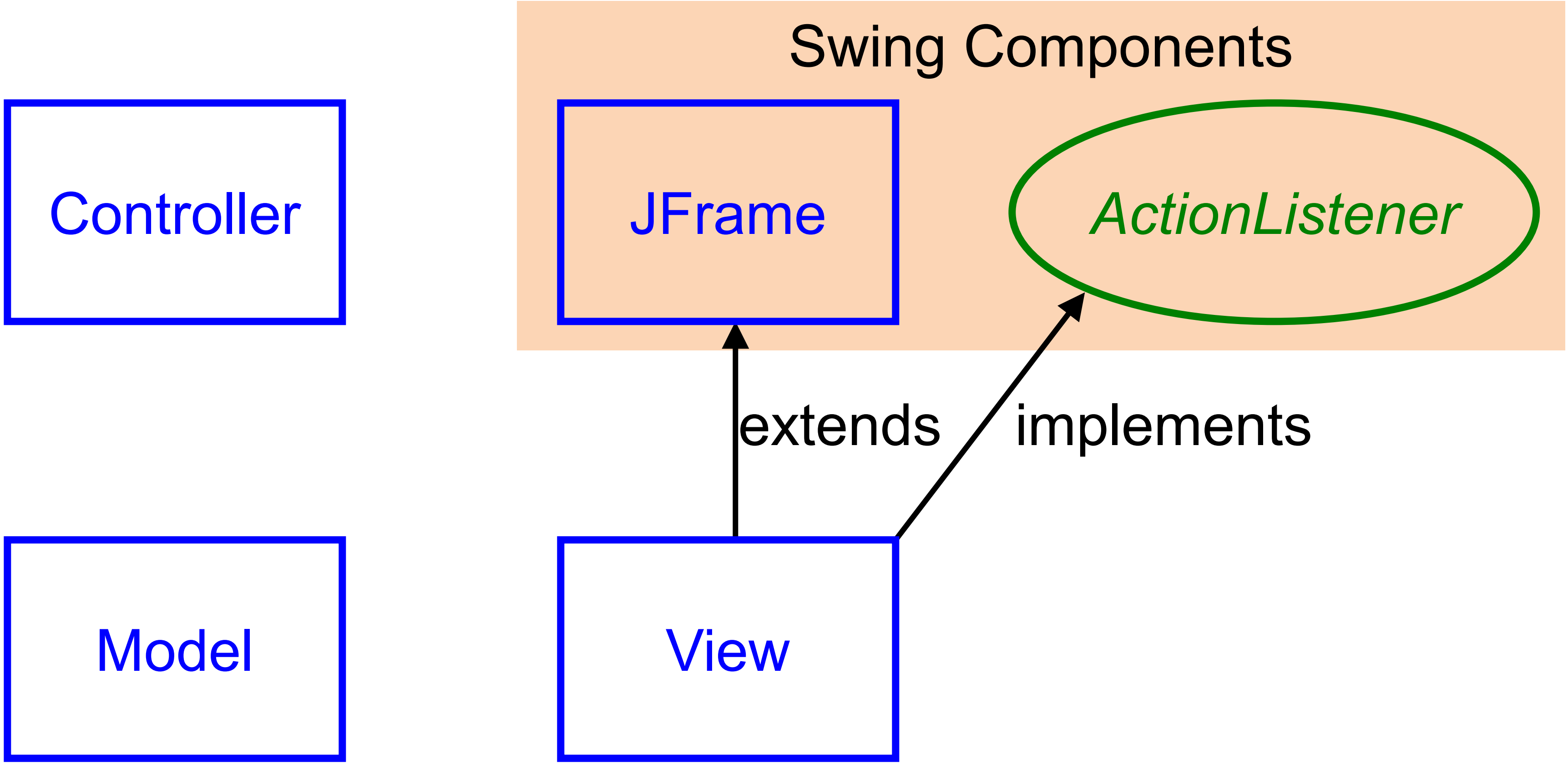
So far, it’s just like the previous GUI demo, except this is called `View`.

Example: Simple MVC GUI Demo

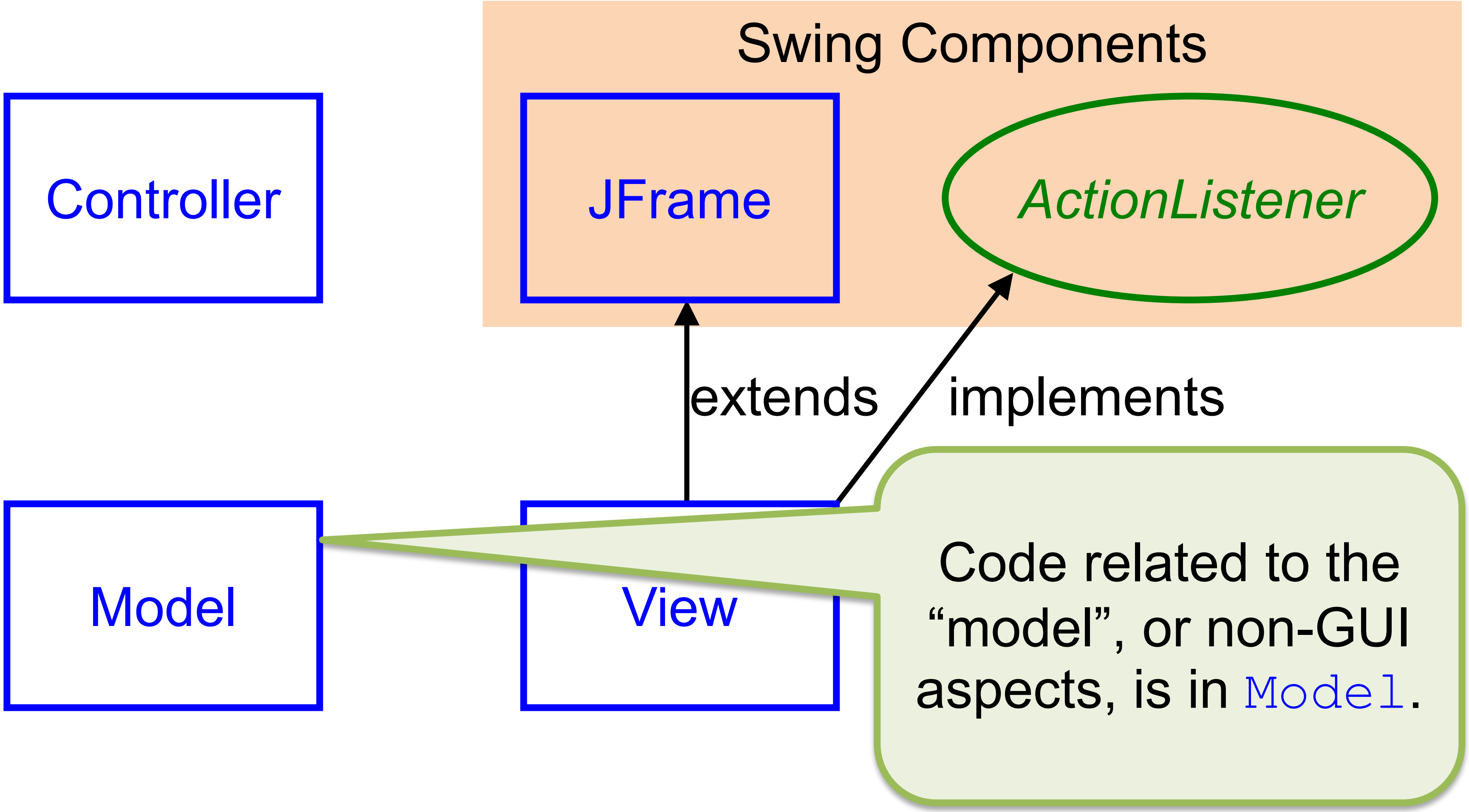


Code related to setting up and using GUI widgets is in `View`; but that's all.

Example: Simple MVC GUI Demo

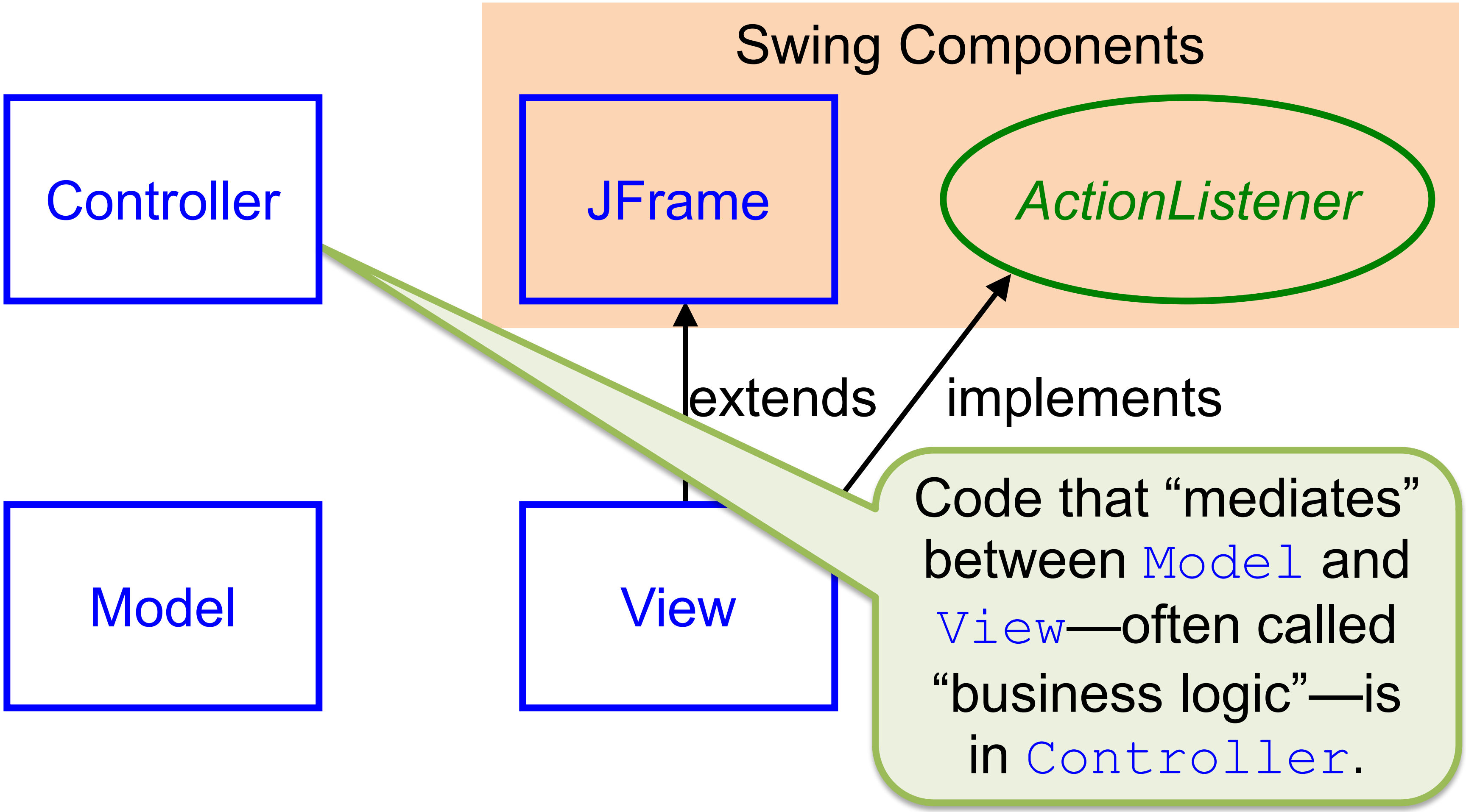


Example: Simple MVC GUI Demo



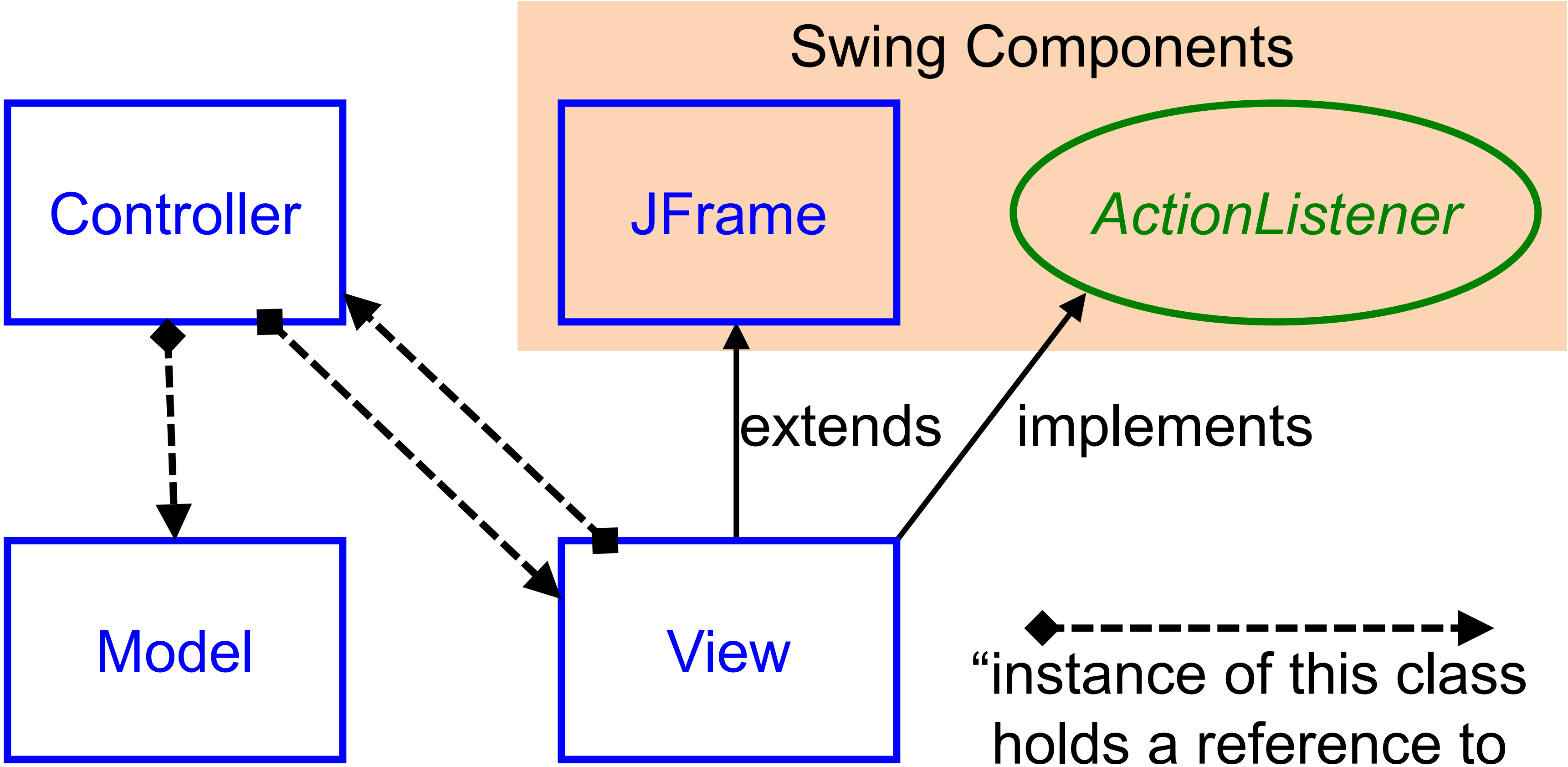
Code related to the "model", or non-GUI aspects, is in `Model`.

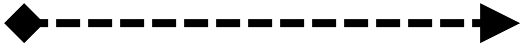
Example: Simple MVC GUI Demo



Code that “mediates” between `Model` and `View`—often called “business logic”—is in `Controller`.

Example: Simple MVC GUI Demo





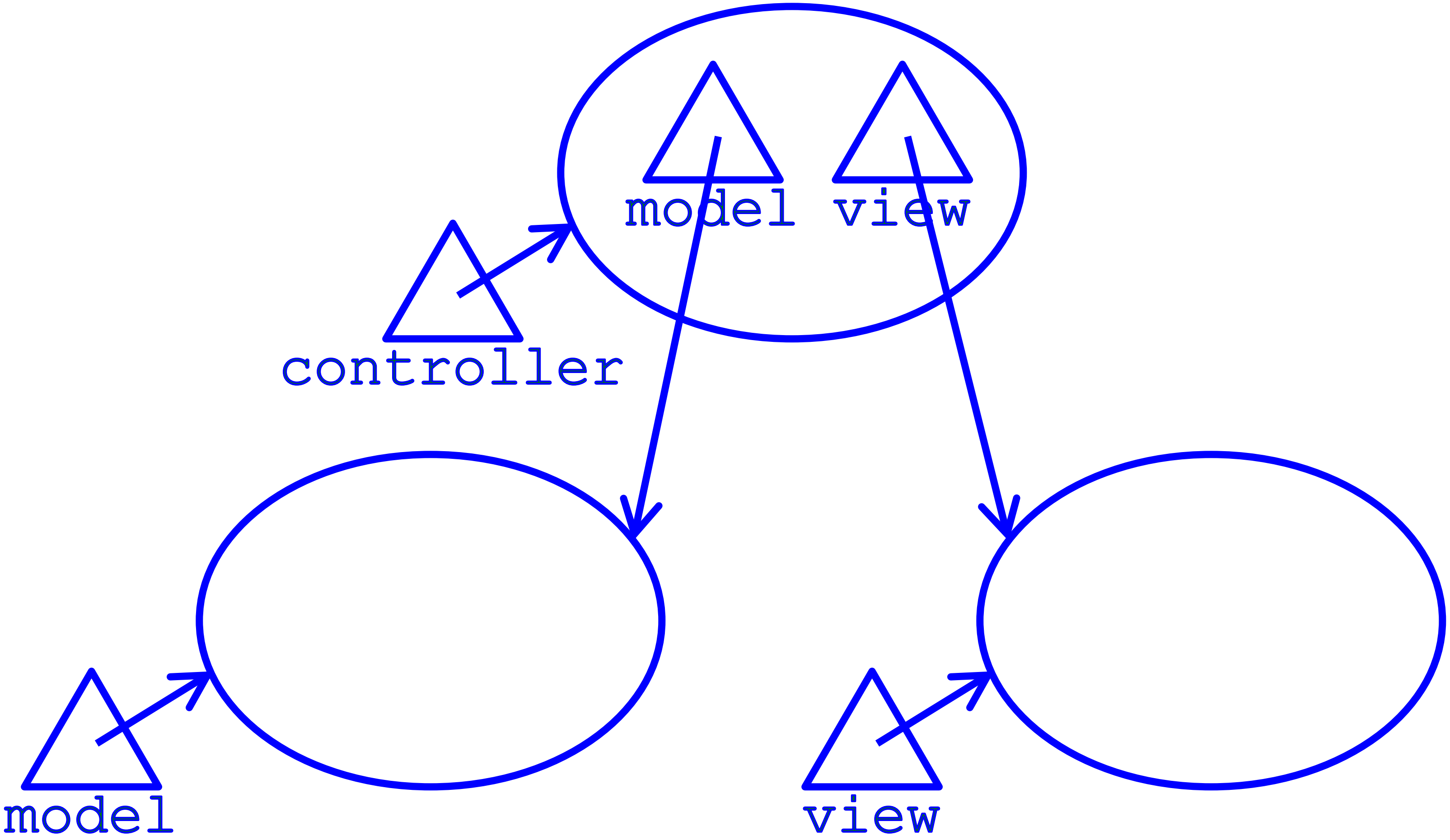
"instance of this class holds a reference to instance of that class"

It’s Demo Time

- The `DemoGUI2` project contains a very simple MVC GUI application using Swing
- You can get it at:

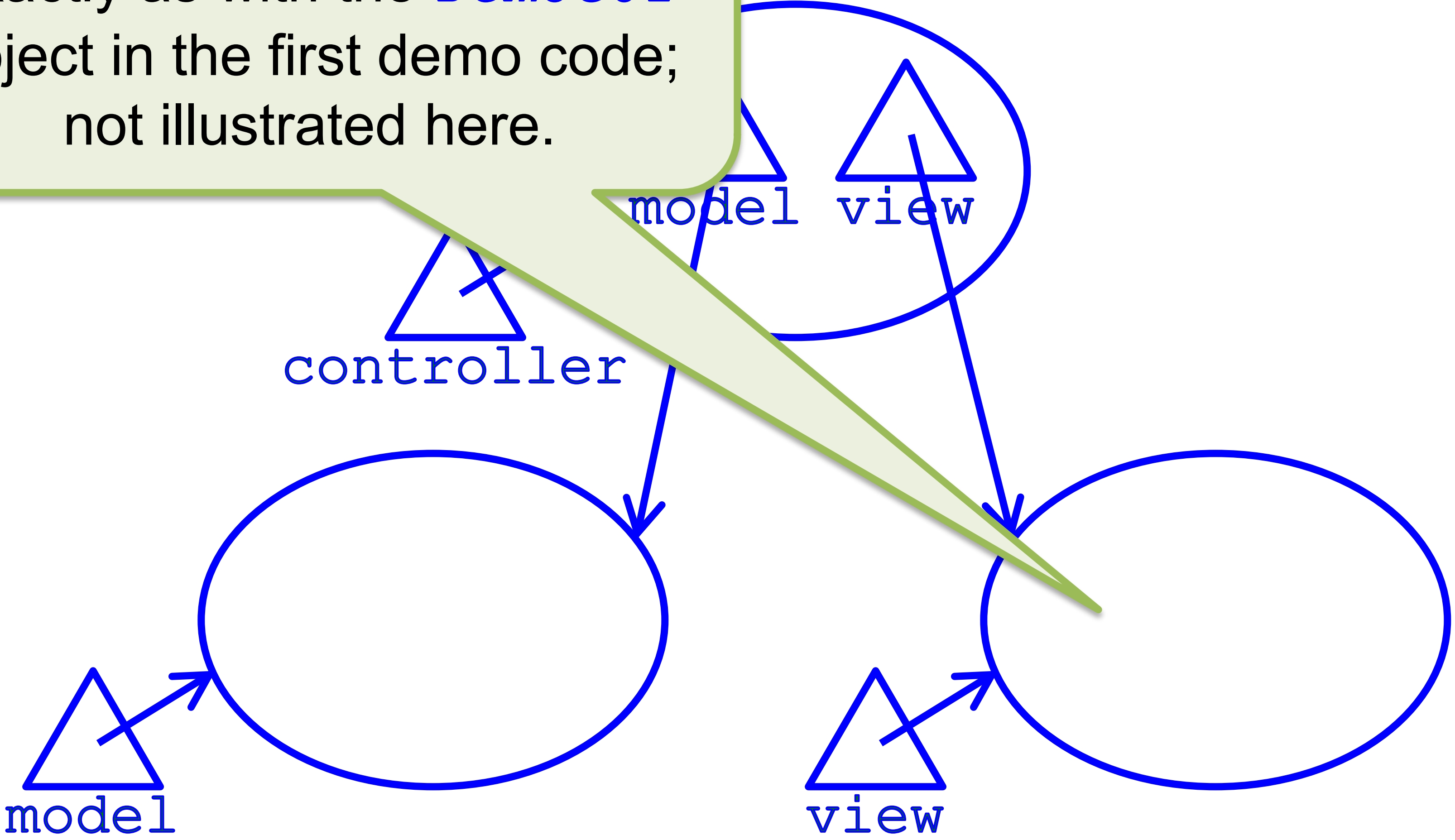
  http://web.cse.ohio-state.edu/software/common/DemoGUI2.zip

Set-up by `main`: After Constructors

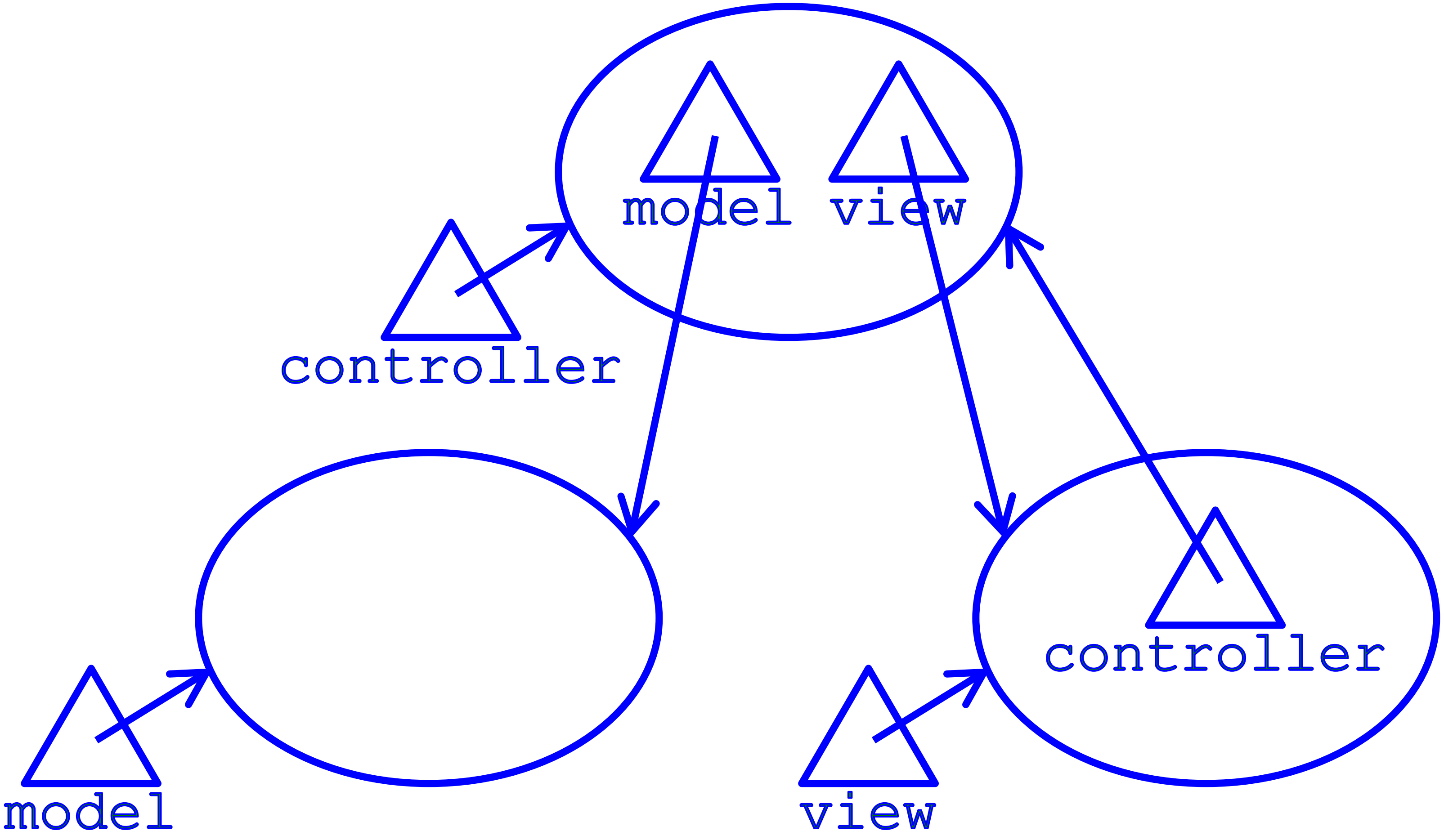


fter Constructors

Constructor sets up observers exactly as with the `DemoGUI` object in the first demo code; not illustrated here.



After `view.registerObserver`



Now, Who's In Charge?

- Once `main` completes in the initial thread, the event dispatch thread executes both the Swing code that monitors user interactions and all callbacks it makes to `actionPerformed` methods