EECS 1720
Building Interactive Systems

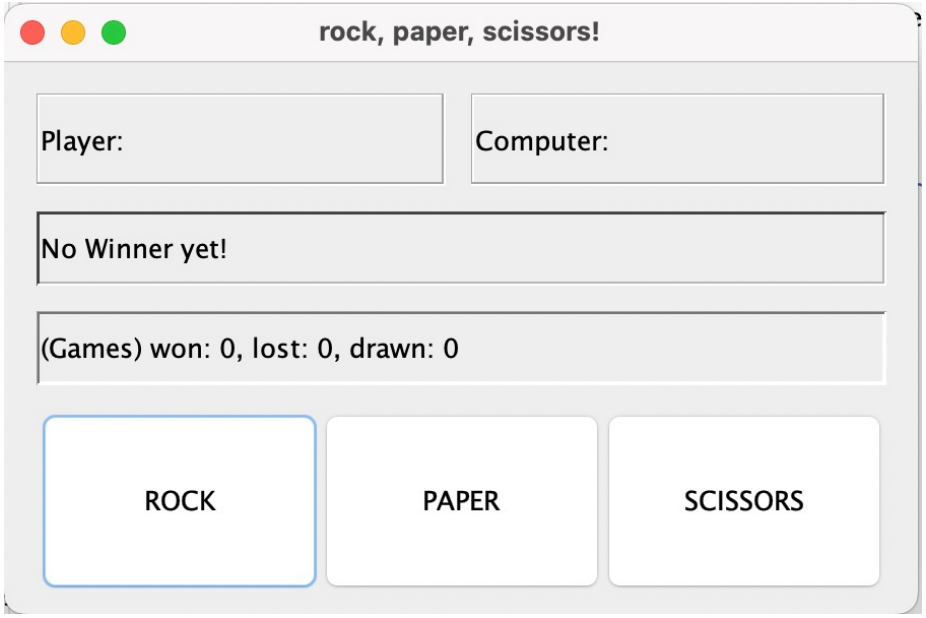
Lecture 18 :: Model-View-Controller (MVC)

Walkthrough Example GUI/Events (Scissors, Paper, Rock)

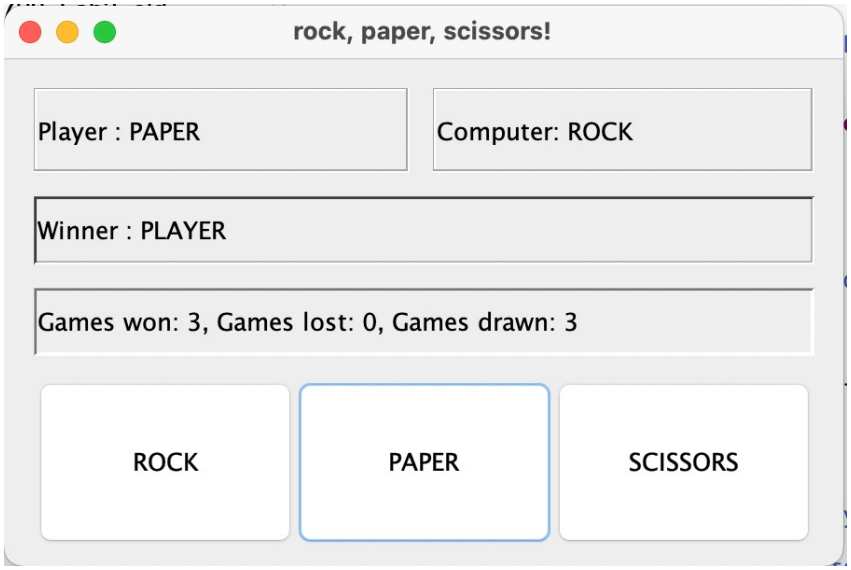
YORK UNIVERSITÉ UNIVERSITY

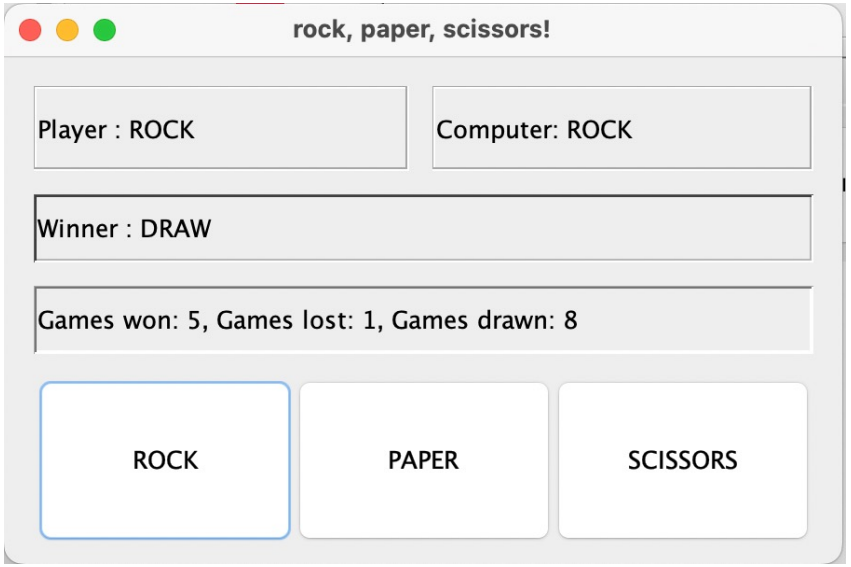
Scissors, Paper, Rock ??

- Simple classic game
  - Rock beats Scissors (rock crushes scissors)
  - Paper beats Rock (paper covers rock)
  - Scissors beat Paper (scissors cut paper)
- Buttons + ActionEvents



Scissors, Paper, Rock ??



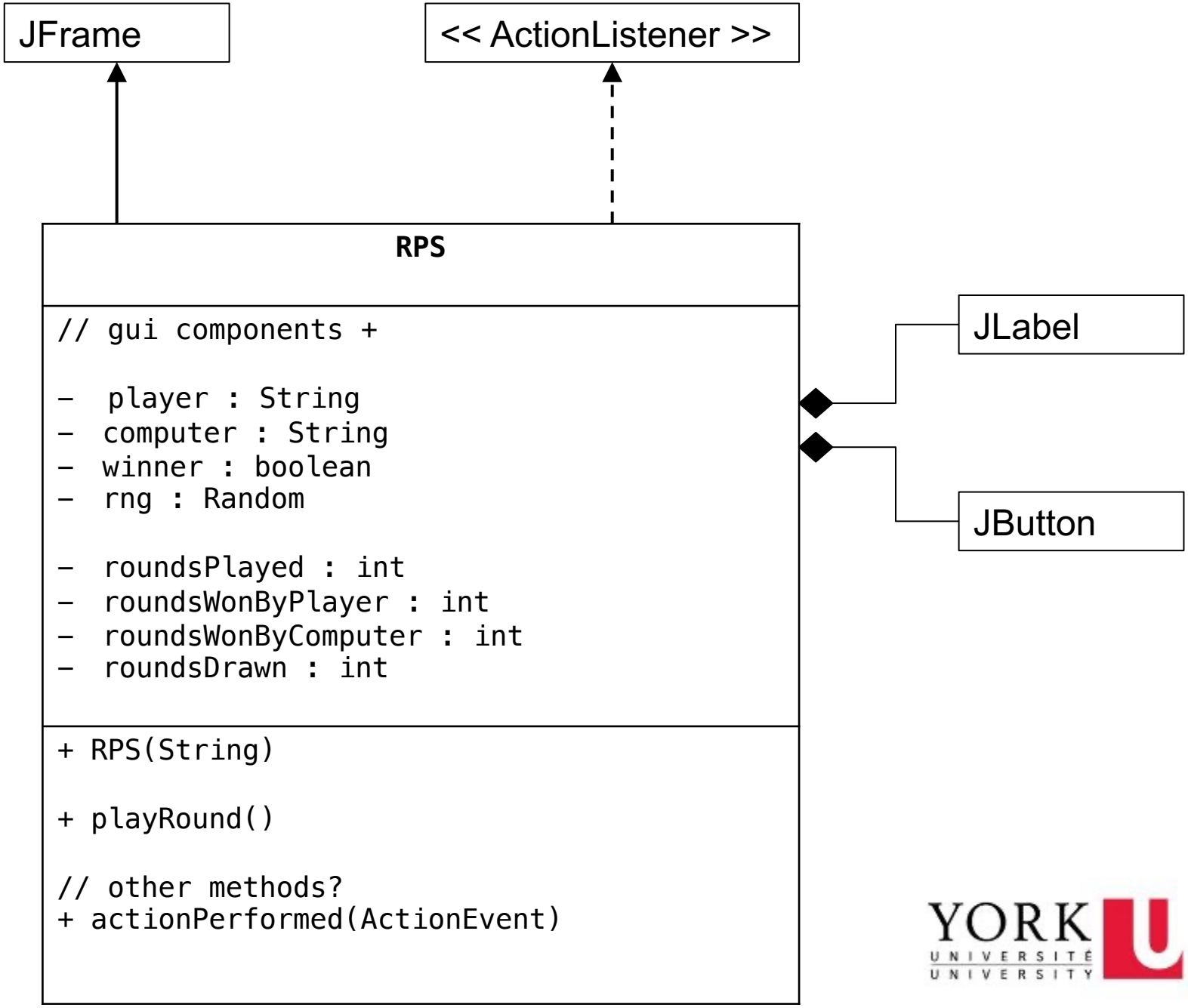


Naïve approach

- One class (extends JFrame)
- Create the components (JLabels, JButtons)
- Make the class implement ActionListener
- Register ”this” as the listener for each Button
- Get listener to update the JLabels when buttons pressed
  - can use action commands

- Each time a button is pressed, call a method to calculate random choice for the computer, compare with the players choice (from the button), and output results

- What do we need to store?



RPS.java

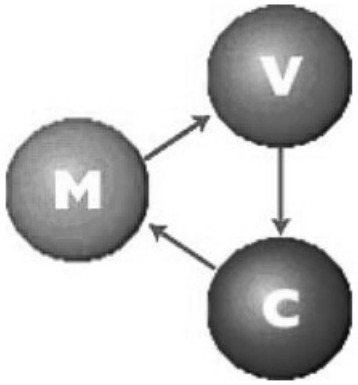
- please see code for week 9 (on eclass)
- and watch the accompanying video for this lecture for the walkthrough
  - this version implements RPS as a single class (extension of JFrame, that also implements the ActionListener)
  - this class combines/mixes together all aspects relating to the state of the RPS game (model), the visual gui (view), and the management of user interactions with the game state through the gui (controller).

- In the next version (remaining slides, we consider organizing this code as separate model/view/controller classes to illustrate the principals of the MVC design pattern

- This pattern is very instrumental to know and use for interactive gui-based applications

Model - View - Controller (design pattern approach)

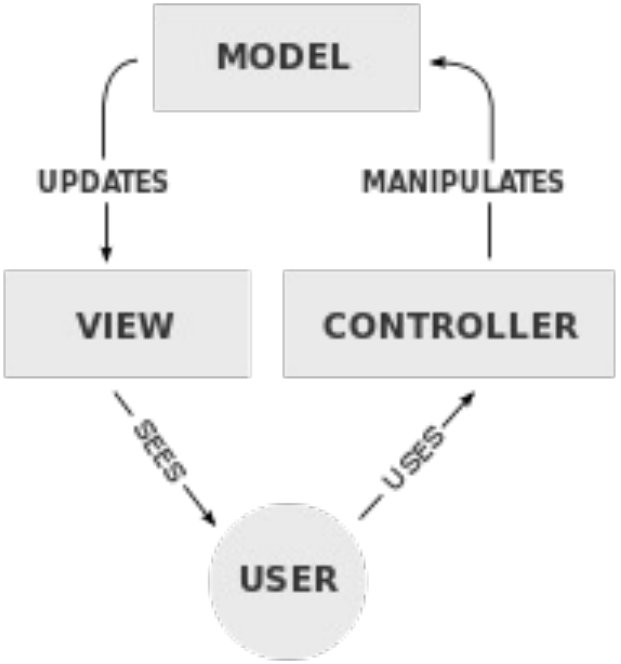
- MVC architecture => breaks up a visual application into several parts
  - A *model* that represents the data for the application
  - The *view* that is the visual representation of that data
  - A *controller* that takes user input on the view and translates that to changes in the model.



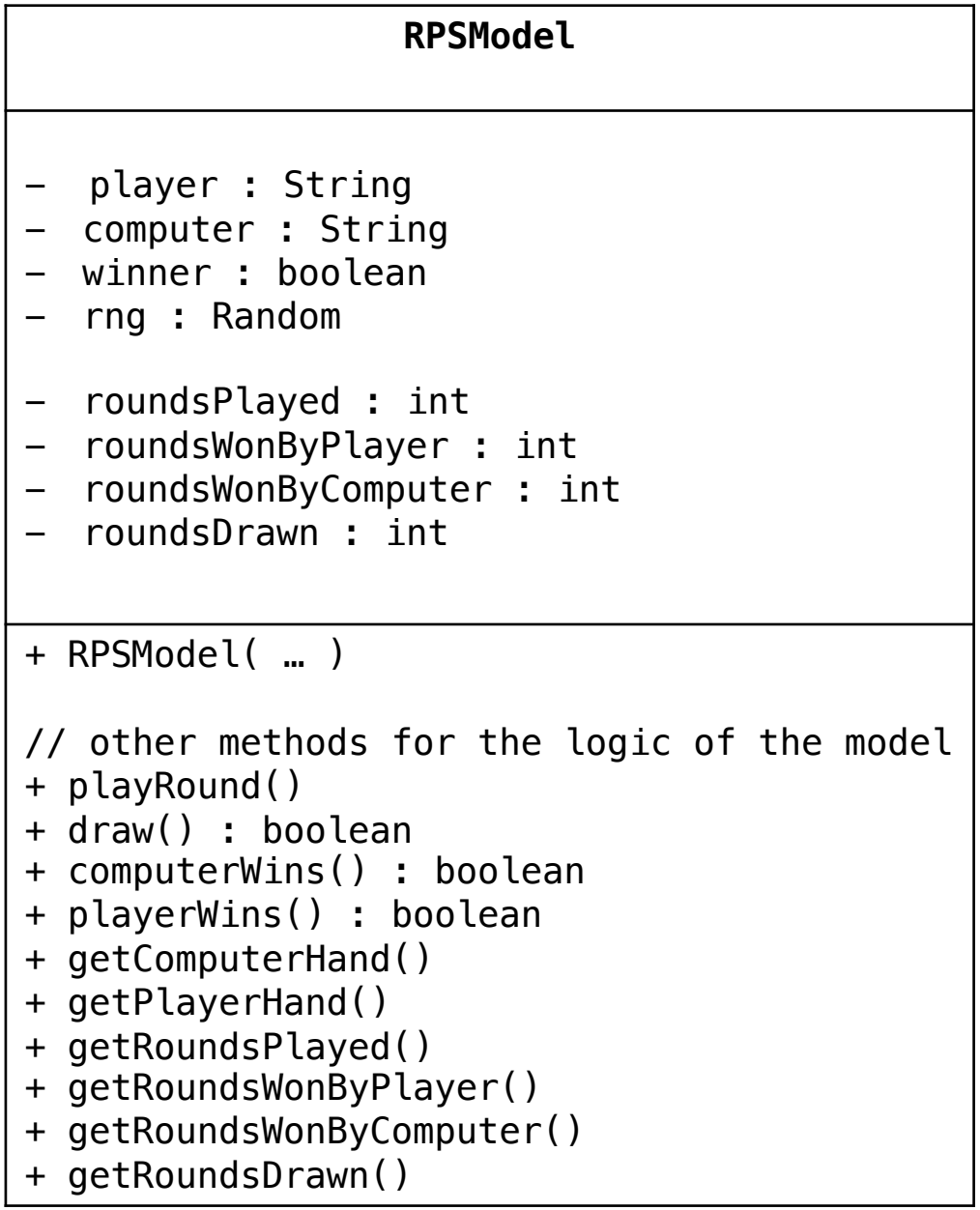
[Amy Fowler, ibid.]

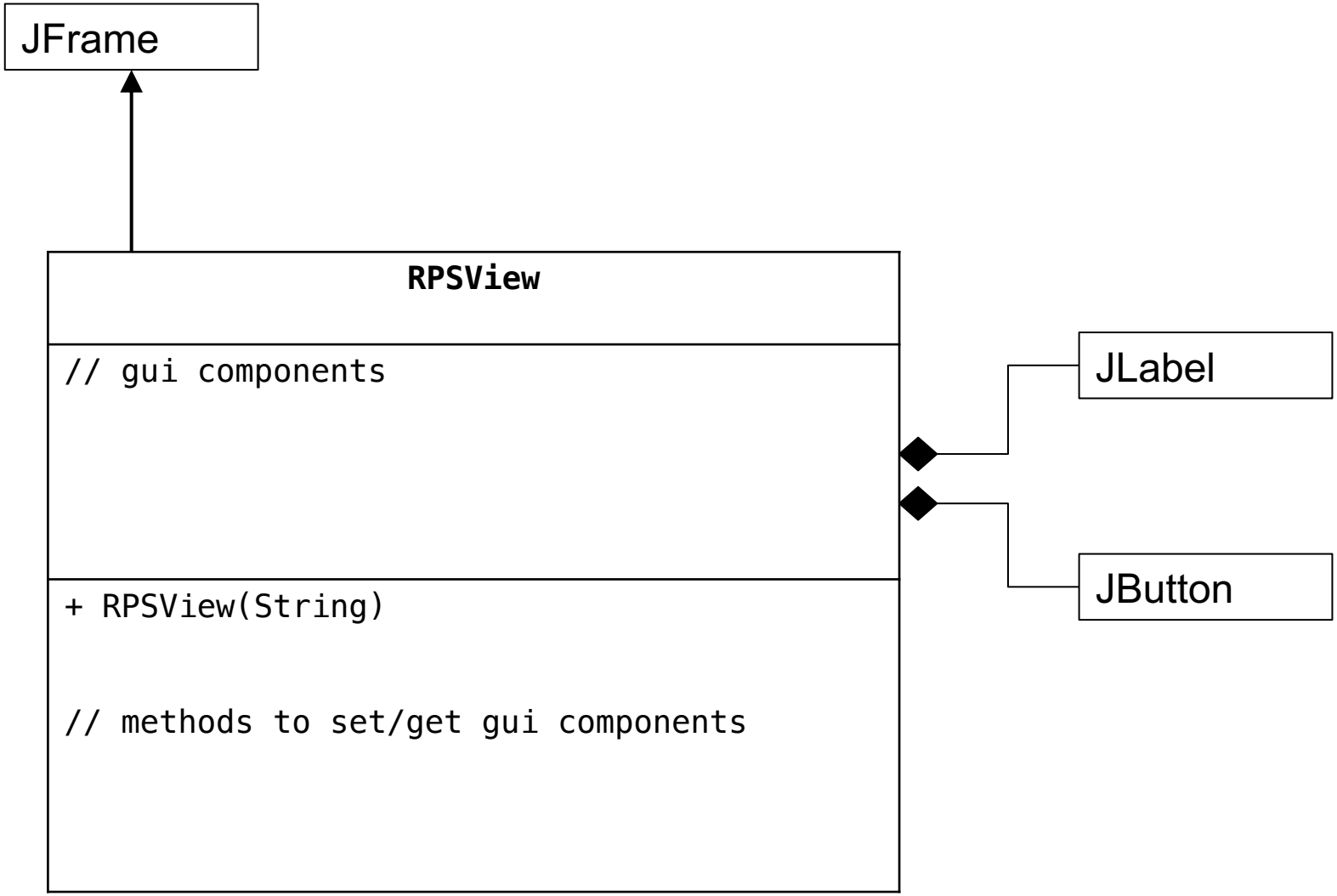
MVC

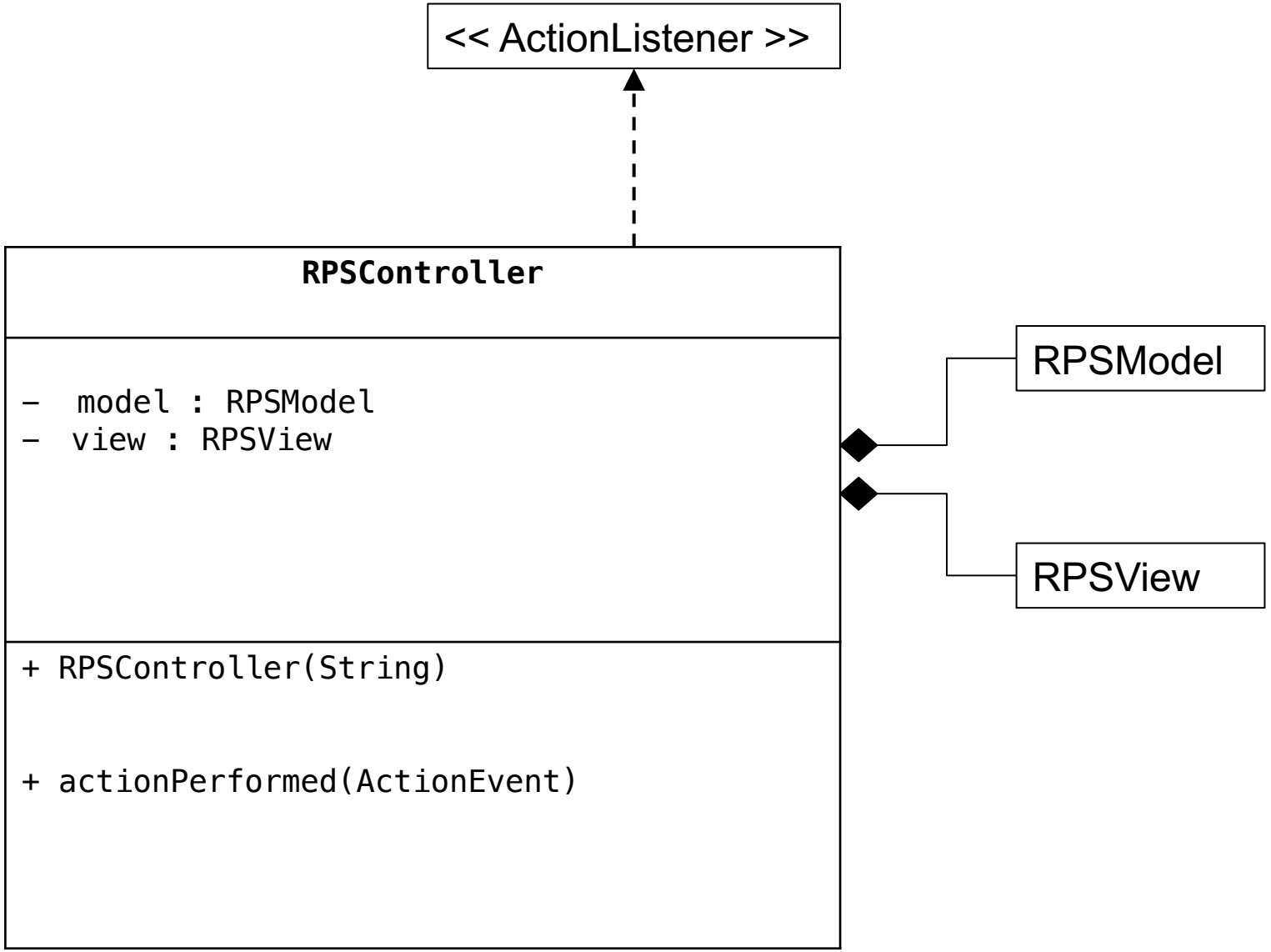
- A **controller**
  - can send commands to the model to update the model's state (e.g., editing a document).
- A **model**
  - notifies its associated views and controllers when there has been a change in its state.
- A **view**
  - requests information from the model that it uses to generate an output representation to the user.



[http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller]







RPSModel, RPSView, RPSController

- see Week 9 code sample on eclass (for the code relating to this example), along with the video walkthrough

- Notes:
  - RPSController doesn't really initialize anything, but holds as its class fields, an RPSModel and an RPSView
    - setModel and setView methods are then called to assign a new RPSModel and RPSView object to the RPSController
    - this way the RPSController has access to both the view and model, and it is the class that will provide the ActionListener
  - RPSGame is a client that creates each of the above objects, connecting the RPSController to the RPSView
  - When RPSView is instantiated, a reference to the RPSController object is passed to its constructor (so that the view can register this as the listener for the JButtons in the view)

- PLEASE LOOK THROUGH THE WEEK9 CODE
  - make sure you understand it
  - this is the type of approach you should think about for assignment 2 (i.e. this is the standard way you would decouple the GUI part of an application from the backend/model part of the application).
  - For example: in a1, if you were creating a card game, the cards and the game logic would all be captured within a Model class (i.e. this would have the objects you needed for your cards, hand, players, etc). The View class would be your main class for the GUI itself
  - We will have more examples like this in coming weeks