Graphical User Interfaces

notes Chap 7

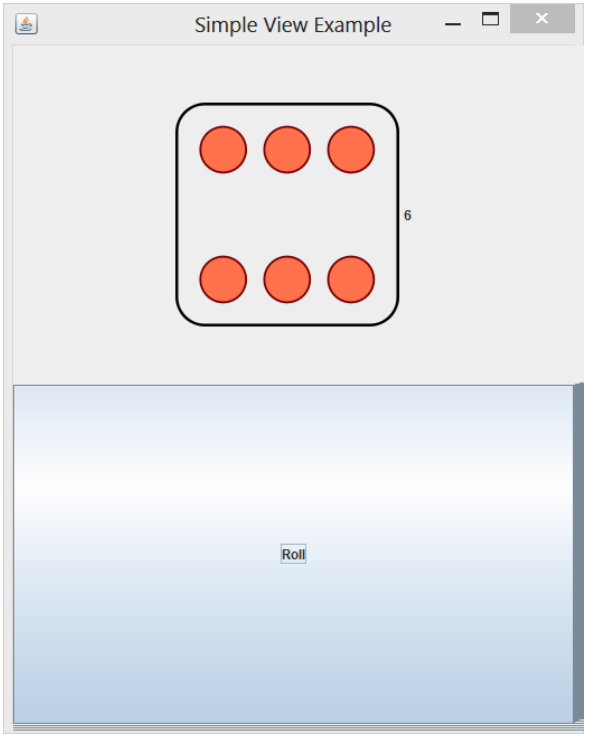
Java Swing

- Swing is a Java toolkit for building graphical user interfaces (GUIs)
  - http://docs.oracle.com/javase/tutorial/uiswing/TOC.html

- old version of the Java tutorial had a visual guide of Swing components
  - http://da2i.univ-lille1.fr/doc/tutorial-java/ui/features/components.html

App to Roll a Die

- a simple application that lets the user roll a die
  - when the user clicks the "Roll" button the die is rolled to a new random value
    - "event driven programming"



App to Roll a Die

- this application is simple enough to write as a single class
  - SimpleRoll.java

Model-View-Controller

- model
  - represents state of the application and the rules that govern access to and updates of state
- view
  - presents the user with a sensory (visual, audio, haptic) representation of the model state
  - a user interface element (the user interface for simple applications)
- controller
  - processes and responds to events (such as user actions) from the view and translates them to model method calls

Model—View—Controller

TV
- on : boolean - channel : int - volume : int
+ power(boolean) : void + channel(int) : void + volume(int) : void

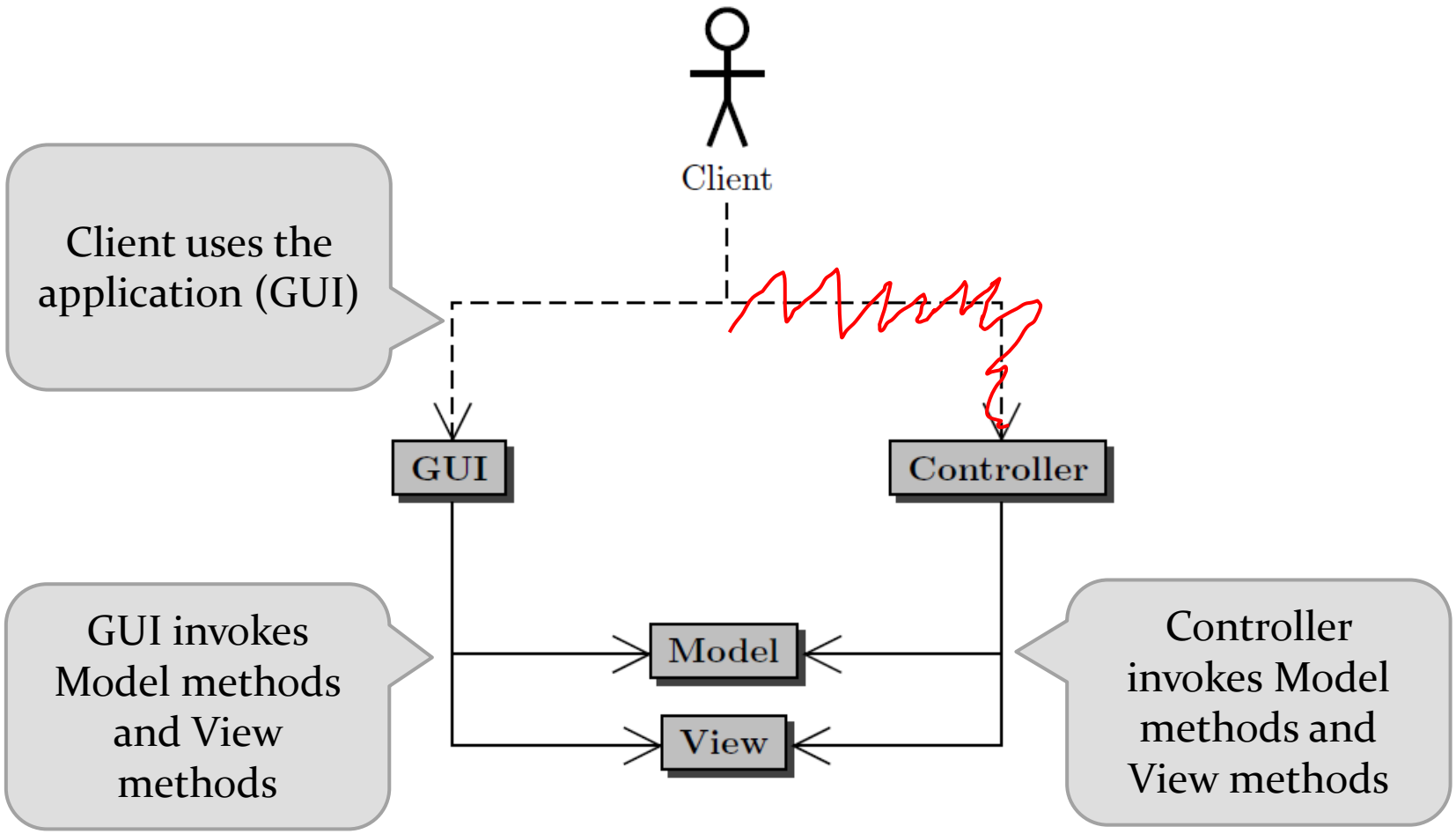
Model

View

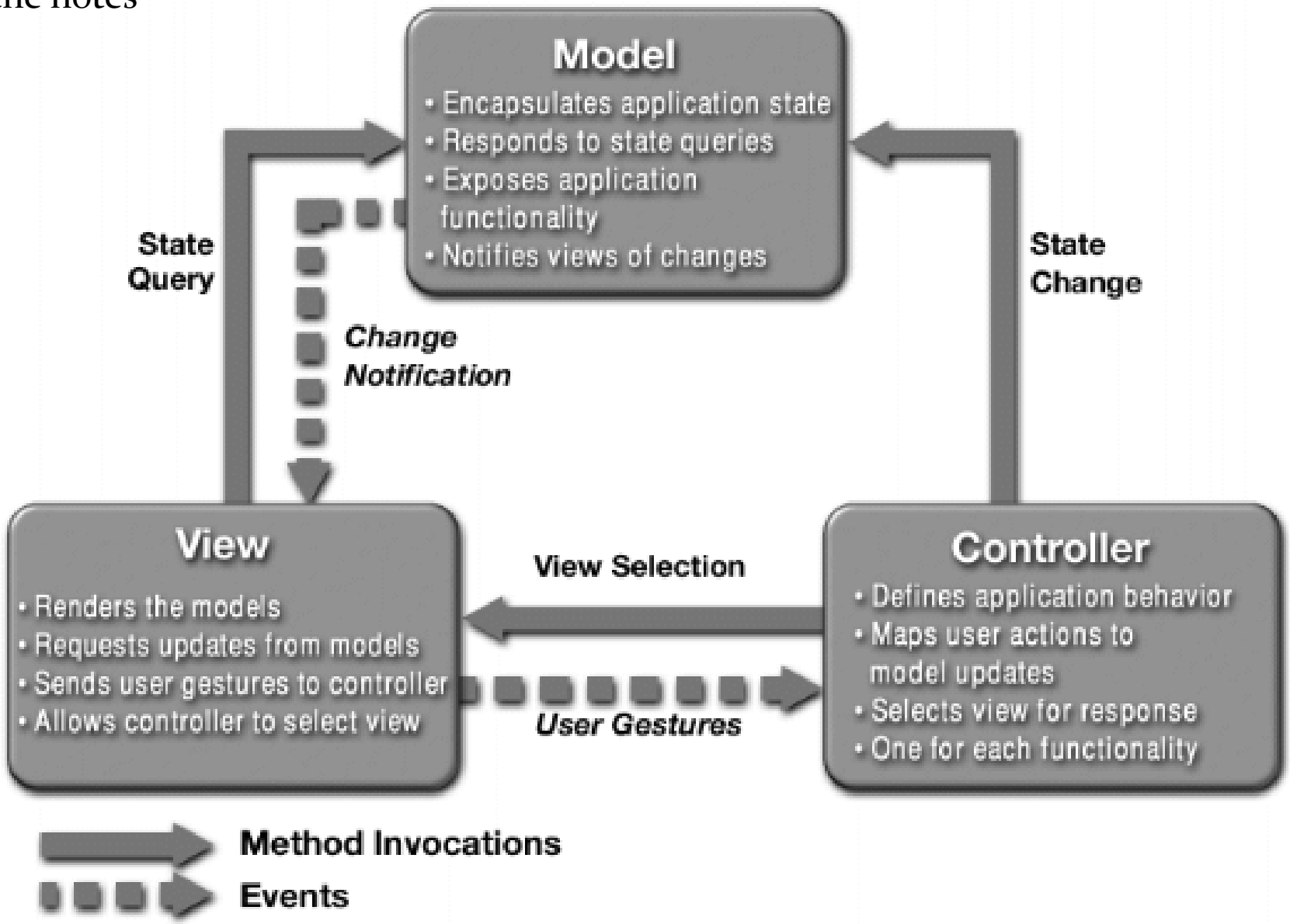
Controller

RemoteControl
+ togglePower() : void + channelUp() : void + volumeUp() : void

Model-View-Controller



a different MVC structure
than in the notes



http://java.sun.com/developer/technicalArticles/javase/mvc/

App to Roll a Die: MVC

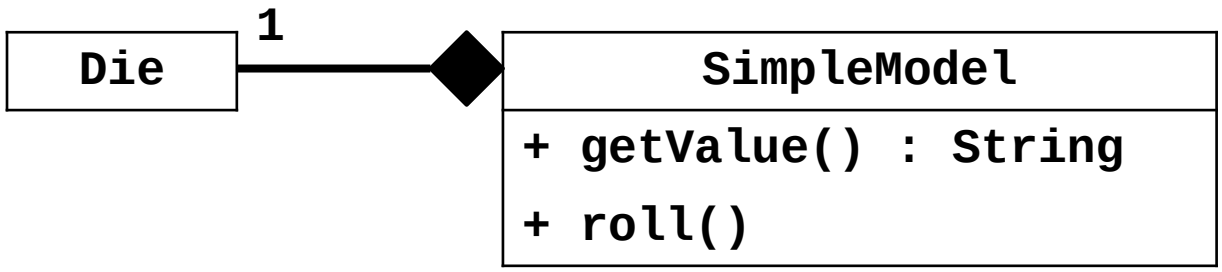
- we can also write the application using the model-view-controller pattern

App to Roll a Die: Model

- model
  - the data
  - methods that get the data (accessors)
  - methods that modify the data (mutators)

- the data
  - a 6-sided die
- accessors
  - get the current face value
- mutators
  - roll the die

App to Roll a Die: Model

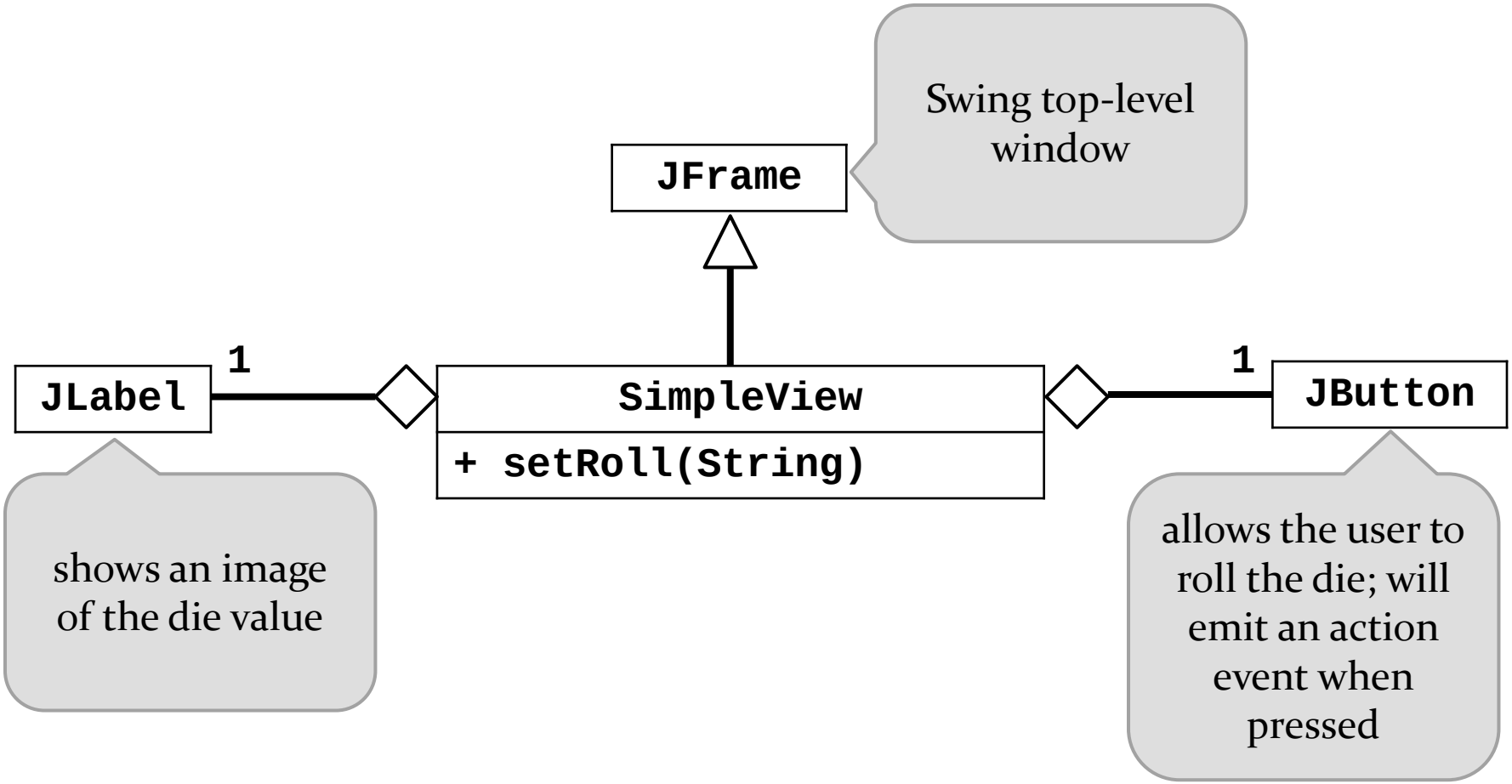


App to Roll a Die: View

- view
  - a visual (or other) display of the model
  - a user interface that allows a user to interact with the view
  - methods that get information from the view (accessors)
  - methods that modify the view (mutators)

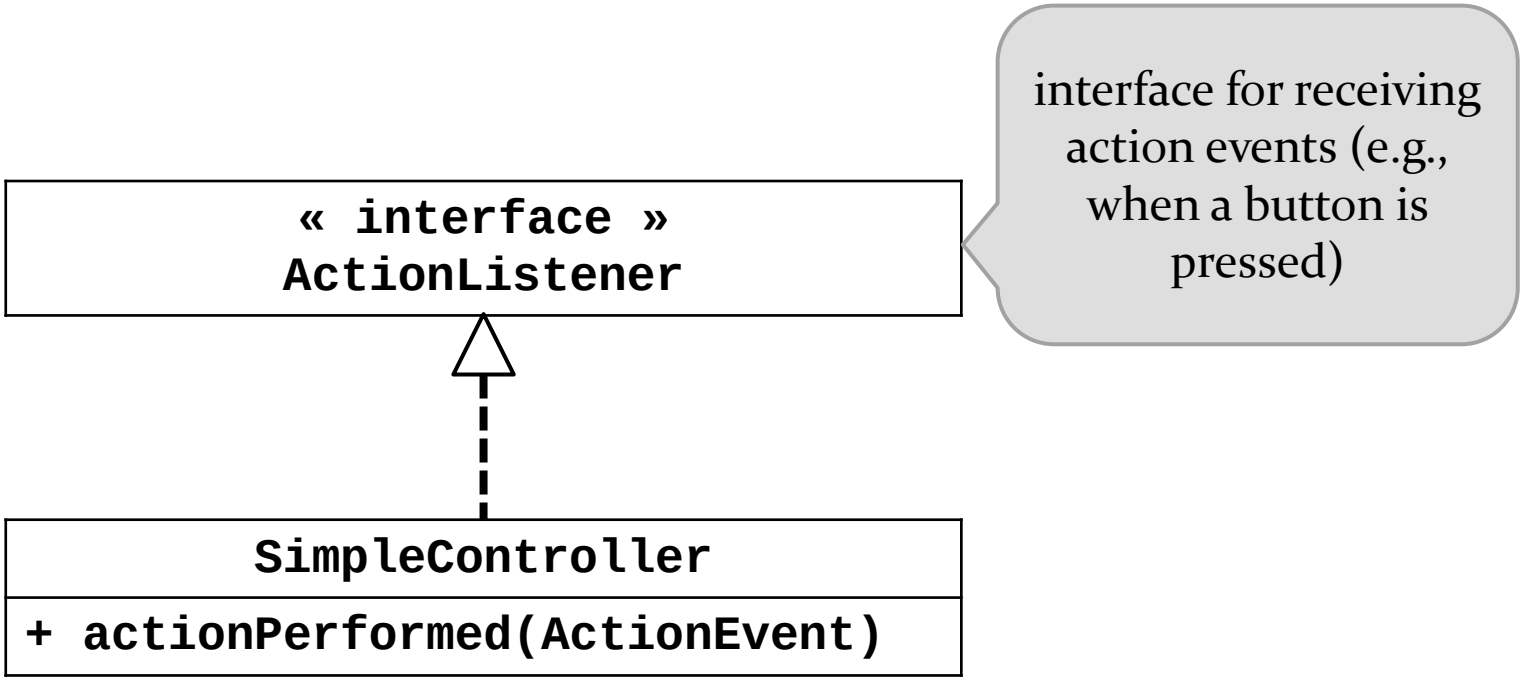
- a visual (or other) display of the model
  - an image of the current face of the die
- a user interface that allows a user to interact with the view
  - roll button

App to Roll a Die: View

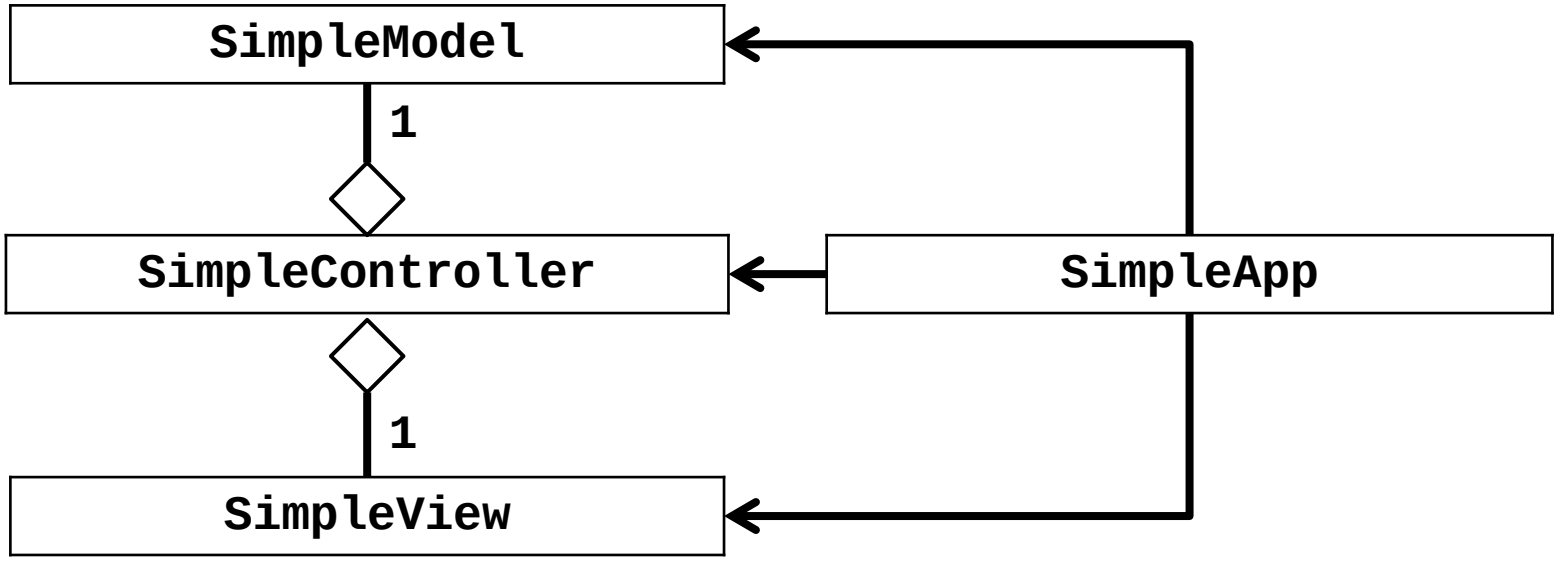


App to Roll a Die: Controller

- controller
  - methods that map user interactions to model updates



App to Roll a Die: MVC



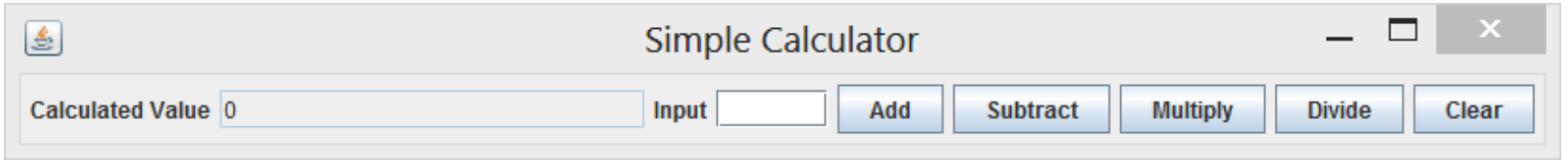
App to Roll a Die

- we can also write the application using the model-view-controller pattern
  - SimpleModel.java
  - SimpleView.java
  - SimpleController.java
  - SimpleApp.java

Simple Calculator

- implement a simple calculator using the model-view-controller (MVC) design pattern
- features:
  - sum, subtract, multiply, divide
  - clear

Application Appearance



Creating the Application

- the calculator application is launched by the user
  - the notes refers to the application as the GUI
- the application:
  1. creates the model for the calculator, and then
  2. creates the view of the calculator

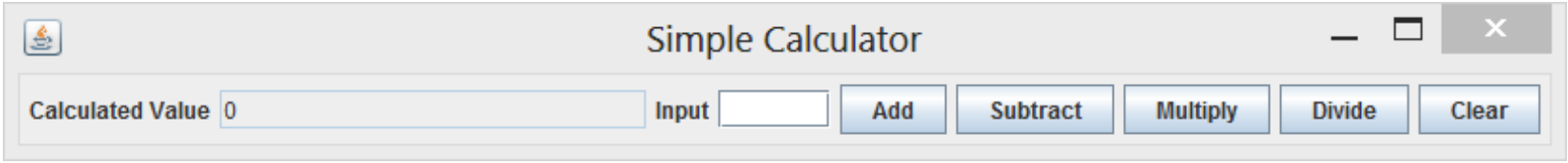
CalcMVC Application

```
public class CalcMVC
{
    public static void main(String[] args)
    {
        CalcController controller = new CalcController();
        CalcModel model = new CalcModel();
        CalcView  view  = new CalcView(model, controller);
        controller.setModel(model);
        controller.setView(view);

        view.setVisible(true);
    }
}
```

Model

- features:
  - sum, subtract, multiply, divide
  - clear



CalcModel
- calcValue : int
+ getCalcValue() : int + getLastUserValue() : int + sum(int) : void + subtract(int) : void + multiply(int) : void + divide(int) : void + clear() : void

CalcModel: Attributes and Ctor

```
public class CalcModel
{
  private int calcValue;

  /**
   * Creates a model with a calculated value of zero.
   */
  public CalcModel() {
    this.calcValue = 0;
  }
```

CalcModel: clear

```
/**
 * Clears the user values and the calculated value.
 */
public void clear() {
  this.calcValue = 0;
}
```

CalcModel: getCalcValue

```
/**
 * Get the current calculated value.
 *
 * @return The current calculated value.
 */
public int getCalcValue() {
  return this.calcValue;
}
```

CalcModel: sum

```
/**
 * Adds the calculated value by a user value.
 *
 * @param userValue
 *          The value to add to the current calculated
 *          value by.
 */
public void sum(int userValue) {
  this.calcValue += userValue;
}
```

CalcModel: subtract and multiply

```
public void subtract(int userValue) {
  this.calcValue -= userValue;
}


public void multiply(int userValue) {
  this.calcValue *= userValue;
}
```

CalcModel: divide

```
/**
  * Divides the calculated value by a user value.
  *
  * @param userValue
  *          The value to multiply the current calculated
  *          value by.
  * @pre. userValue is not equivalent to zero.
  */
 public void divide(int userValue) {
   this.calcValue /= userValue;
 }
```

Other model examples

- consider the Boggle app from CSE1030 last term
  - http://www.eecs.yorku.ca/course_archive/2013-14/F/1030/labs/07/lab7.html
- consider Eclipse
- pick your favourite game and design a model for the game