Graphical User Interfaces

notes Chap 7

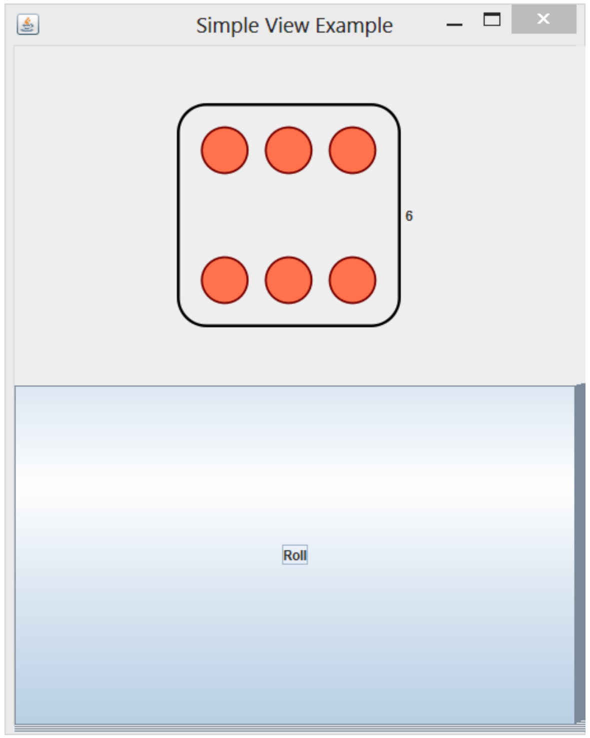
Java Swing

- 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
```

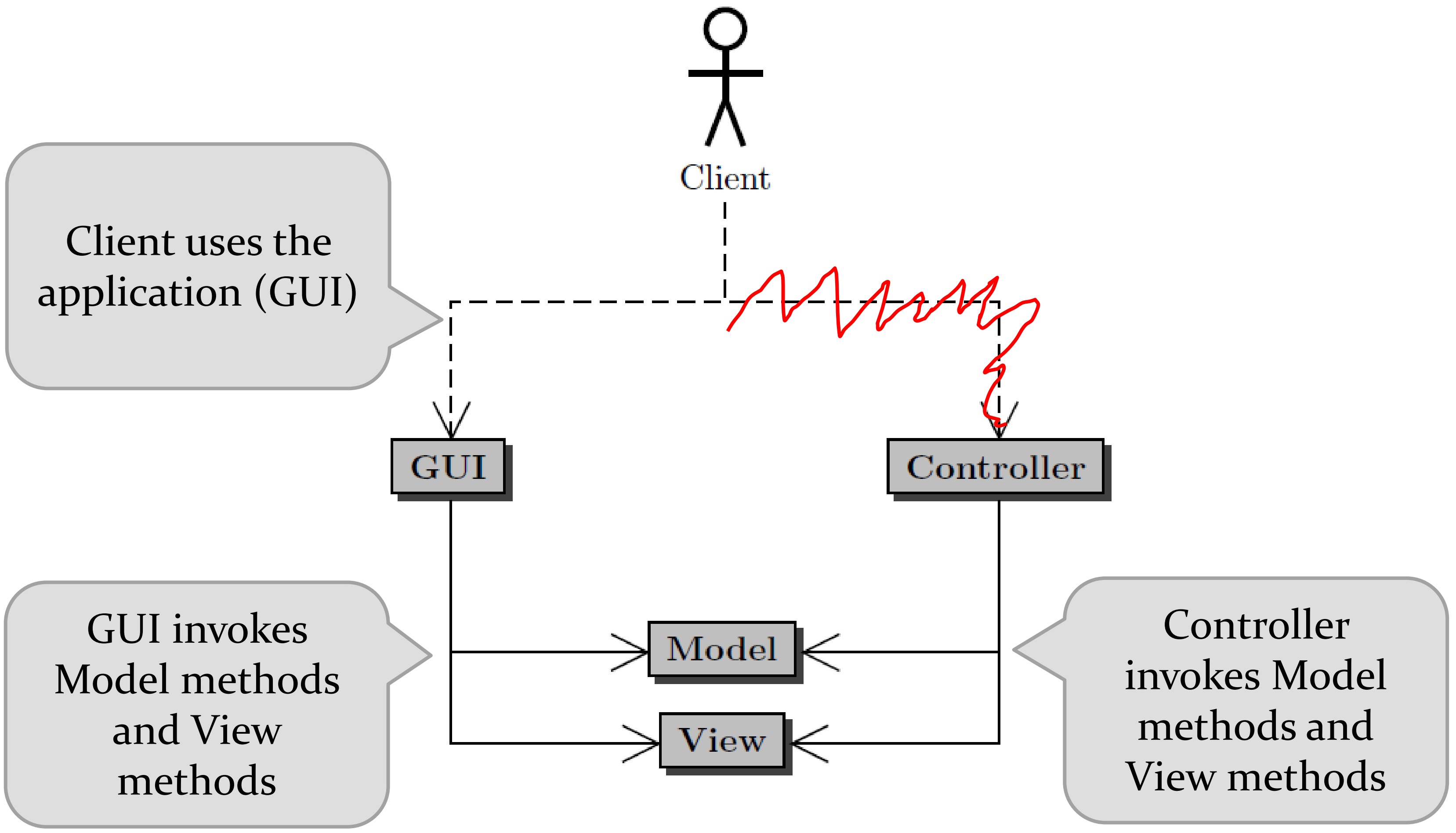
Model

View

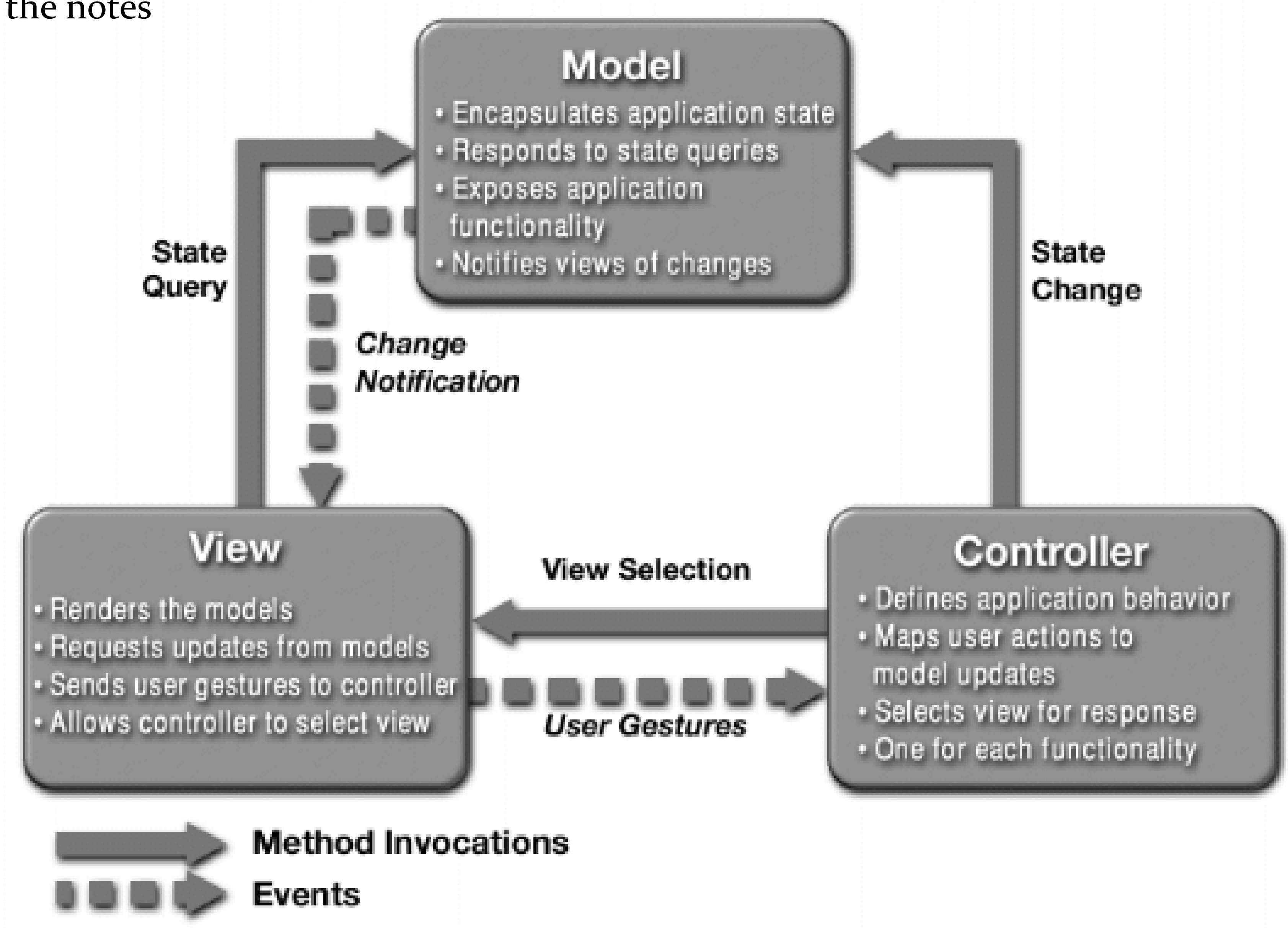
Controller

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

Model-View-Controller


Client
Client uses the application (GUI)
GUI
Controller
Model
View
GUI invokes Model methods and View methods
Controller invokes Model methods and View methods

a different MVC structure
than in the notes


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

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

App to Roll a Die: MVC

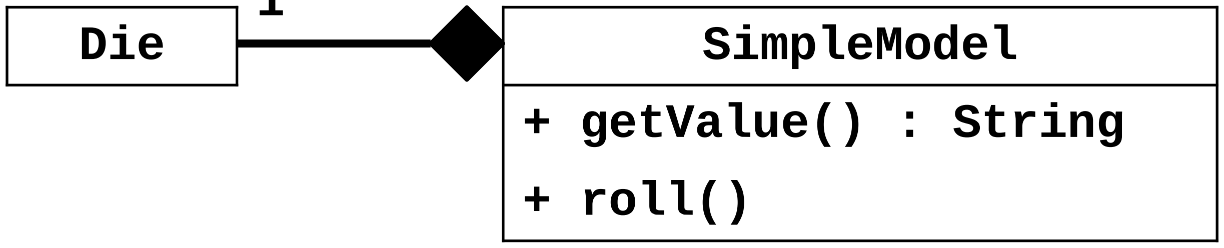
- we can also write the application using the model-view-controller pattern

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

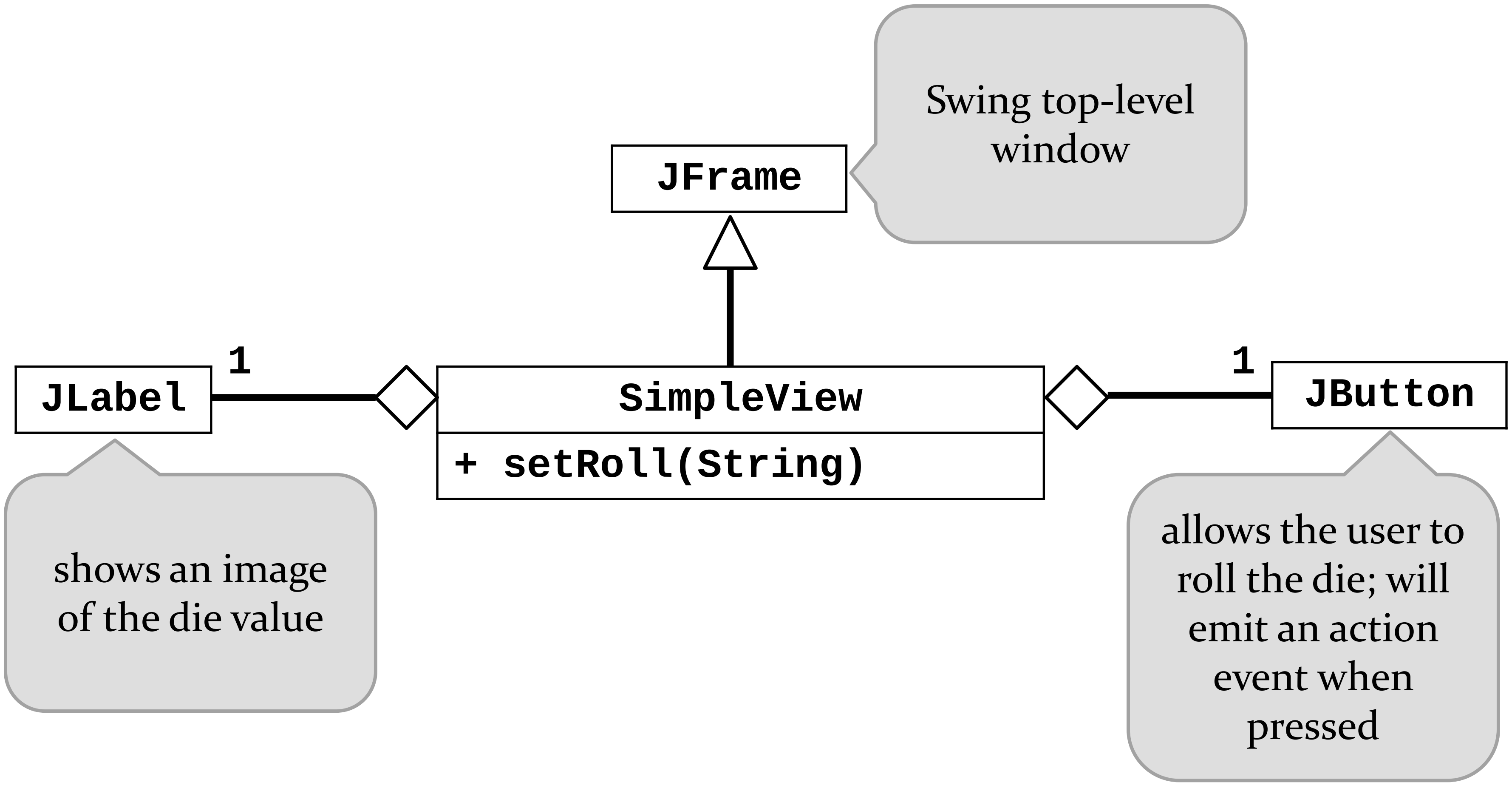


App to Roll a Die: View

- 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)

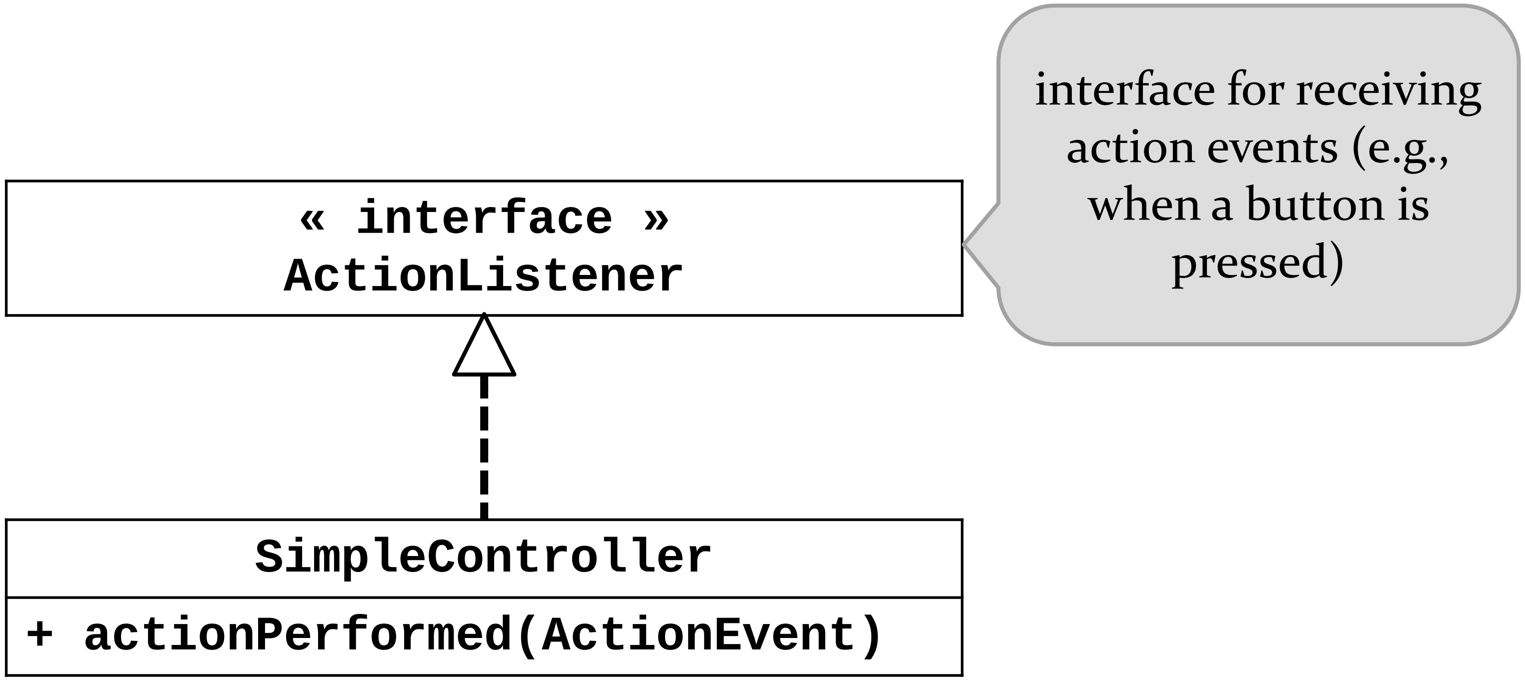
- a visual (or other) display of the model
  - an image of the current face of the die
- a user interface that allows a user to interact with the view
  - roll button

App to Roll a Die: View

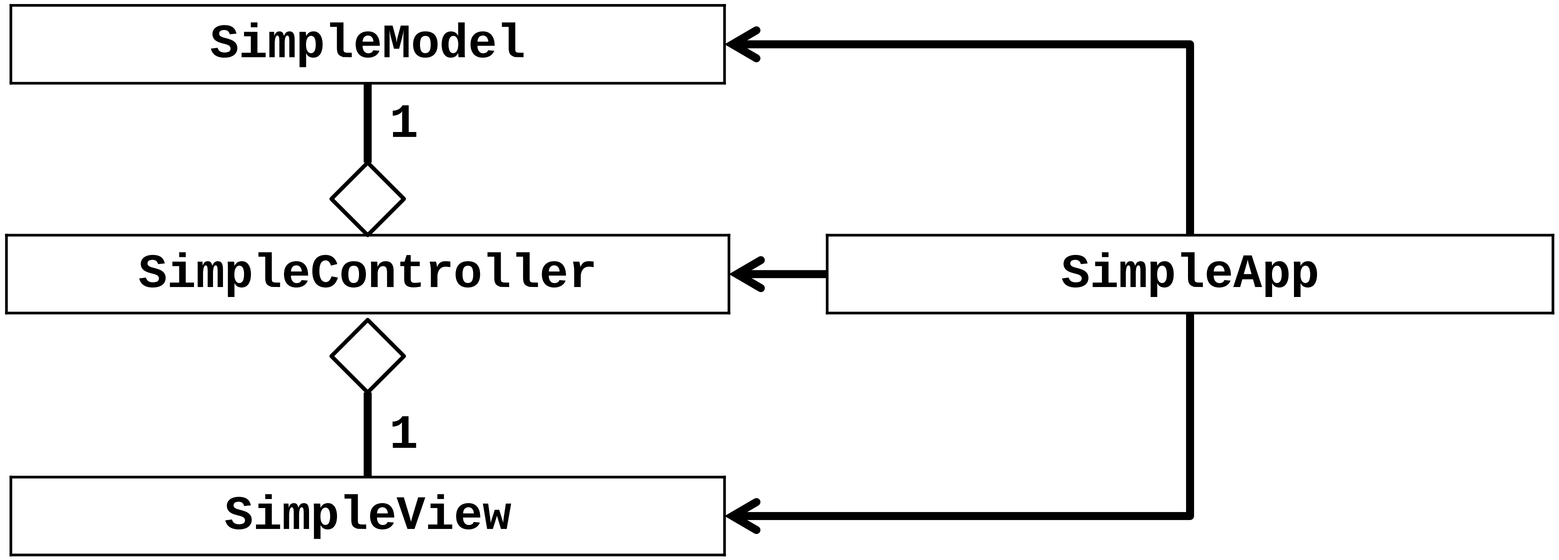

Swing top-level window
JFrame
JLabel
1
SimpleView
+ setRoll(String)
1
JButton
shows an image of the die value
allows the user to roll the die; will emit an action event when pressed

App to Roll a Die: Controller

- controller
  - methods that map user interactions to model updates


interface for receiving action events (e.g., when a button is pressed)
« interface »
ActionListener
SimpleController
+ actionPerformed(ActionEvent)

App to Roll a Die: MVC



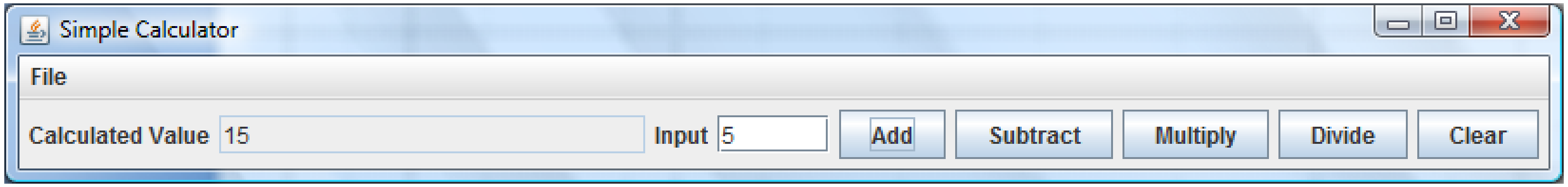
App to Roll a Die

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

Simple Calculator

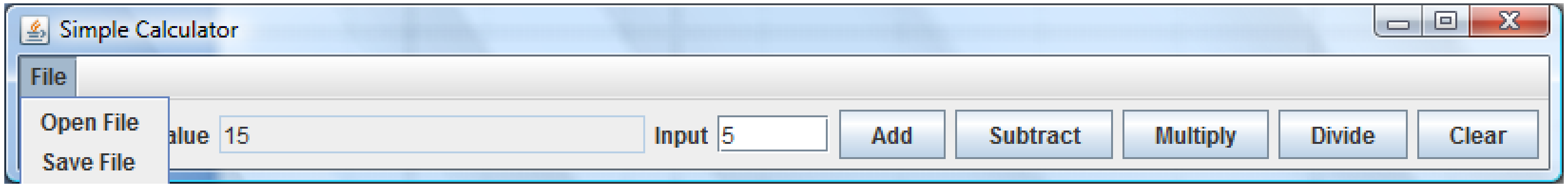
- implement a simple calculator using the model-view-controller (MVC) design pattern
- features:
  - sum, subtract, multiply, divide
  - clear
  - records a log of the user actions
    - save the log to file
    - read the log from a file

Application Appearance



Calculated Value 15

Input 5



Calculated Value 15

Input 5

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

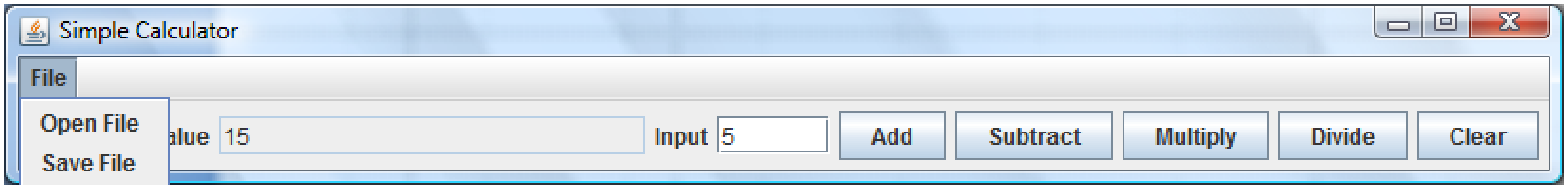
CalcMVC Application

```
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
  - records a log of the user actions
    - save the log to file
    - read the log from a file



BigInteger:
Immutable arbitrary-precision integers

```
CalcModel
--------------------------------------------------------------
- calcValue : BigInteger
- log : ArrayList<String>
--------------------------------------------------------------
+ getCalcValue() : BigInteger
+ getLastUserValue() : BigInteger
+ sum(BigInteger) : void
+ subtract(BigInteger) : void
+ multiply(BigInteger) : void
+ divide(BigInteger) : void
+ clear() : void
+ save(File) : void
+ open(File) : void
- updateLog(String operation, String userValue) : void
```

CalcModel: Attributes and Ctor

```
public class CalcModel
{
  private BigInteger calcValue;
  private ArrayList<String> log;

  // creates the log and initializes the attributes
  //   using the clear method
  CalcModel()
  {
    this.log = new ArrayList<String>();
    this.clear();
  }
```

CalcModel: clear

```
// sets the calculated value to zero, clears the log,
//   and adds zero to the log
public void clear()
{
  this.calcValue = BigInteger.ZERO;
  this.log.clear();
  this.log.add(this.calcValue.toString());
}
```

CalcModel: getLastUserValue

```
// empty log looks like
// [0]
// non-empty log looks like:
// [0, +, 5, =, 5, -, 3, =, 2, *, 7, =, 14]
public BigInteger getLastUserValue()
{
  if(this.log.size() == 1)
  {
    return BigInteger.ZERO;
  }
  final int last = this.log.size() - 1;
  return new BigInteger(this.log.get(last - 2));
}
```

CalcModel: getCalcValue

```
// BigInteger is immutable; no privacy leak
public BigInteger getCalcValue()
{
  return this.calcValue;
}
```

CalcModel: sum

```
// sums the user value with the current calculated value
//   and updates the log using updateLog
public void sum(BigInteger userValue)
{
  this.calcValue = this.calcValue.add(userValue);
  this.updateLog("+", userValue.toString());
}
```

CalcModel: subtract and multiply

```
public void subtract(BigInteger userValue)
{
  this.calcValue = this.calcValue.subtract(userValue);
  this.updateLog("-", userValue.toString());
 }


public void multiply(BigInteger userValue)
{
  this.calcValue = this.calcValue.multiply(userValue);
  this.updateLog("*", userValue.toString());
}
```

CalcModel: divide

```
// cannot divide by zero; options:
// 1. precondition  userValue != 0
// 2. validate userValue; do nothing
// 3. validate userValue; return false
// 4. validate userValue; throw exception
public void divide(BigInteger userValue)
{
  this.calcValue = this.calcValue.divide(userValue);
  this.updateLog("/", userValue.toString());
}
```

CalcModel: save

```
// relies on fact ArrayList implements Serializable
public void save(File file)
{
  FileOutputStream f = null;
  ObjectOutputStream out = null;
  try {
    f = new FileOutputStream(file);    // can throw
    out = new ObjectOutputStream(f);   // can throw
    out.writeObject(this.log);         // can throw
    out.close();
  }
  catch(IOException ex)
  {}
}
```

CalcModel: open

```
public void open(File file) {
  FileInputStream f = null;
  ObjectInputStream in = null;
  ArrayList<String> log = null;   // object to read from file
  try {
    f = new FileInputStream(file);             // can throw
    in = new ObjectInputStream(f);             // can throw
    log = (ArrayList<String>) in.readObject(); // can throw
    in.close();
    this.log = log;
    final int last = this.log.size() - 1;
    this.calcValue = new BigInteger(this.log.get(last));
  }
  catch(IOException ex) {}
  catch(ClassNotFoundException ex) {}
}
```