Graphical User Interfaces (pt 1)

Based on slides by Prof. Burton Ma

Model—View—Controller

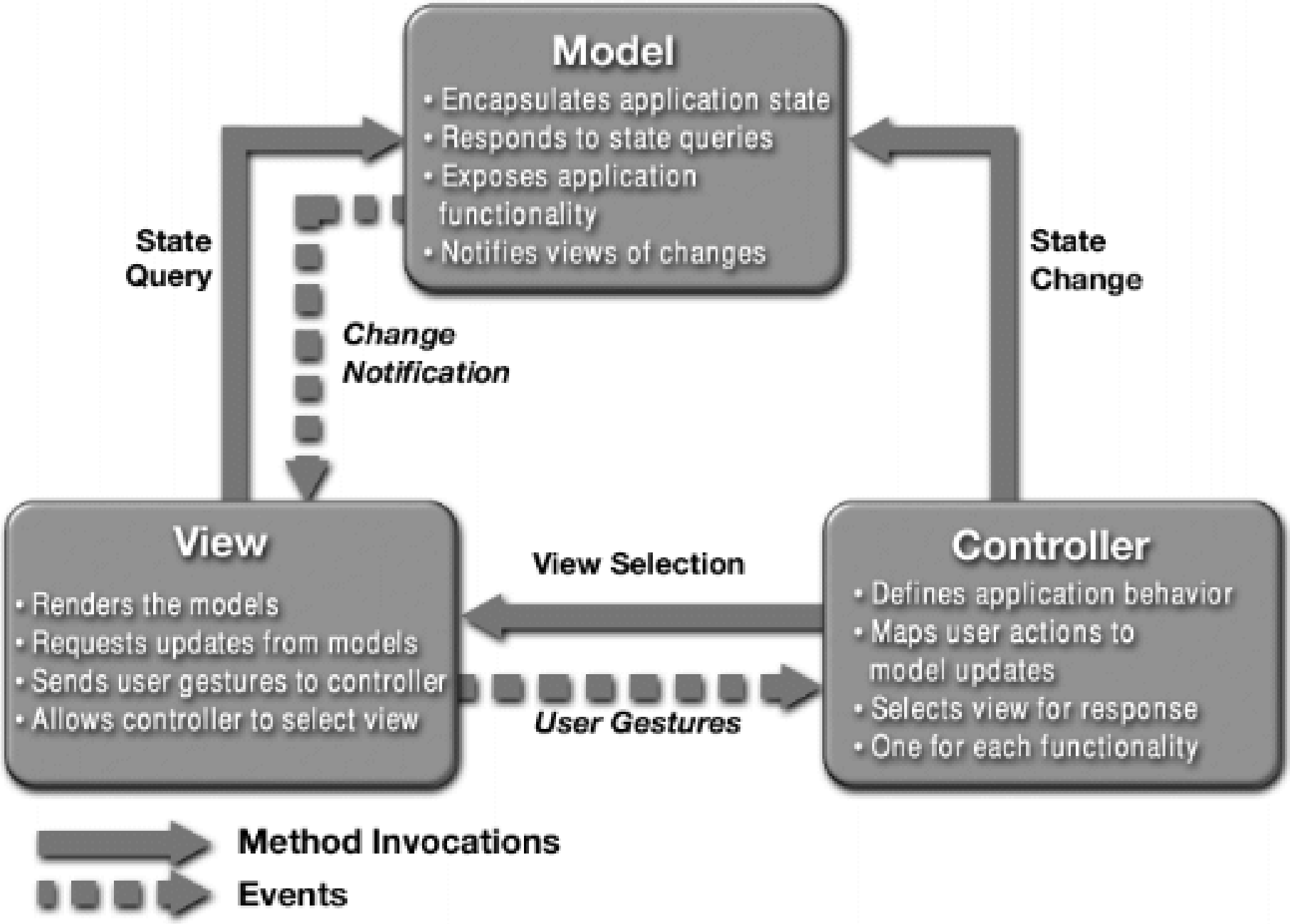
TV
- on : boolean - channel : int - volume : int
+ power(boolean) : void + channel(int) : void + volume(int) : void

Model

View

Controller

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

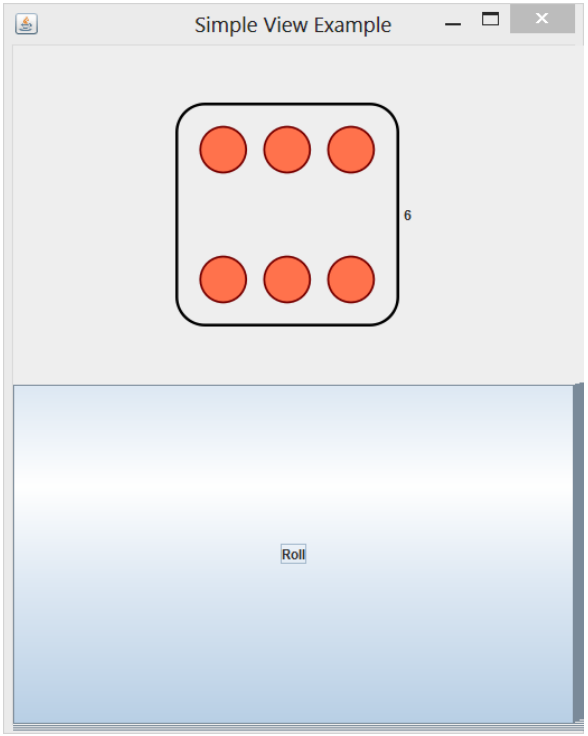


http://java.sun.com/developer/technicalArticles/javase/mvc/

- 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

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

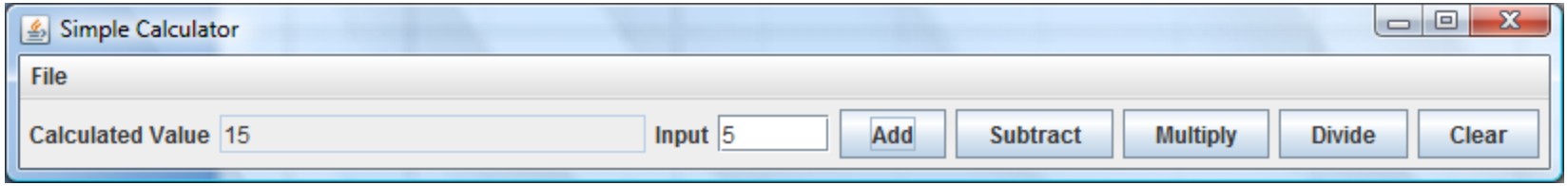
App to Roll a Die

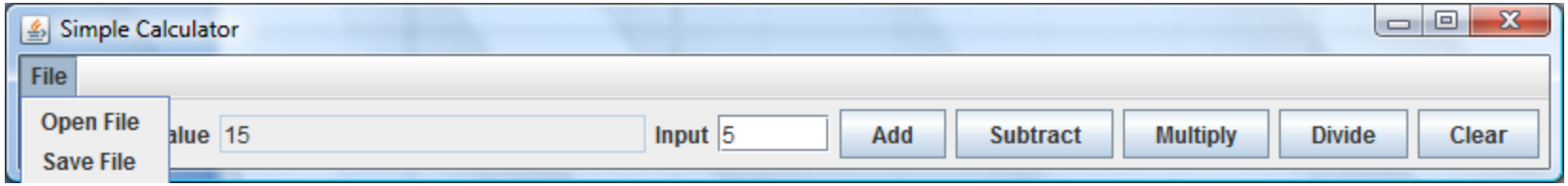
- We can also write the application using the model-view-controller pattern
  - SimpleModel.java
  - SimpleView.java
  - SimpleController.java
  - SimpleApp.java

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





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

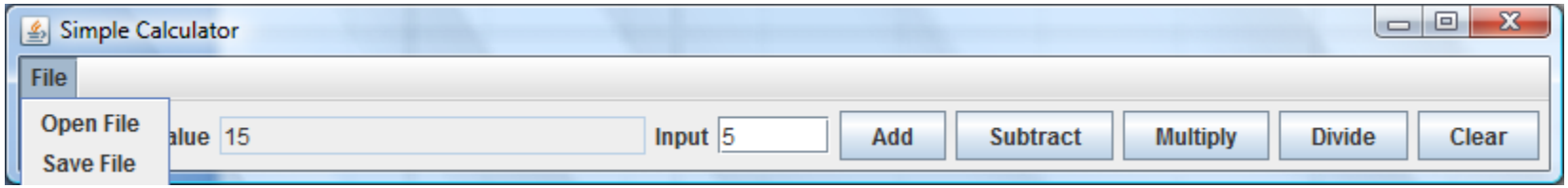
```
public class CalcMVC
{
   public static void main(String[] args)
   {

      CalcModel model = new CalcModel();
      CalcView  view  = new CalcView(model);

      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) {}
}
```