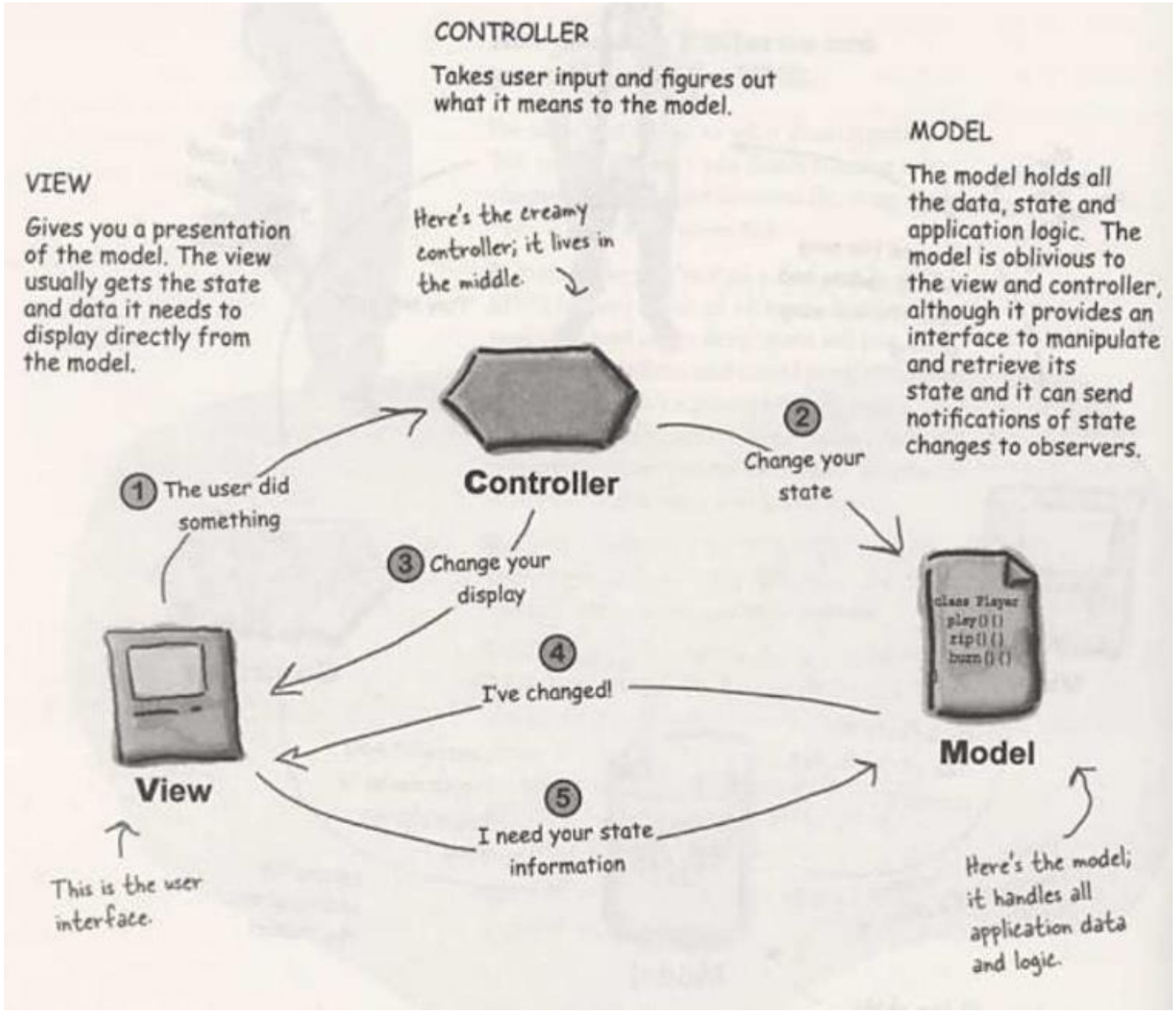
MVC

MVC

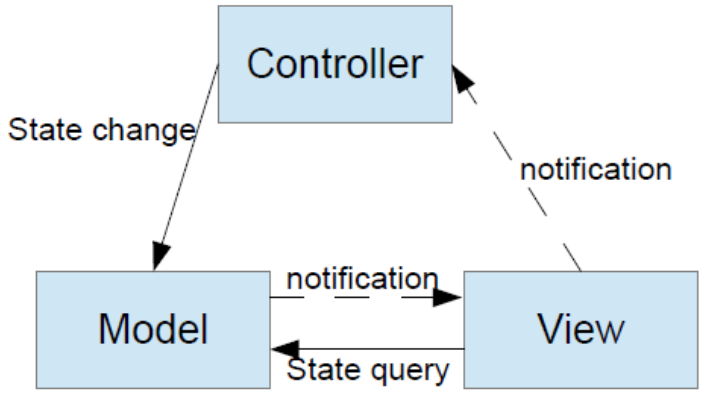


Source: https://ii.uni.wroc.pl/~lukstafi/pmwiki/uploads/Java/java-lecture10.pdf

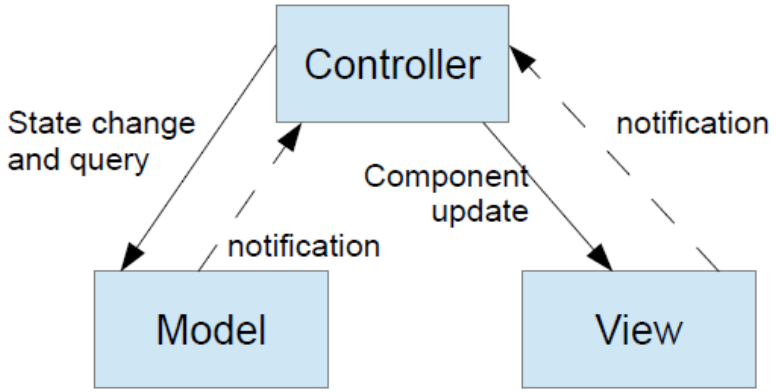
MVC

Swing architecture is rooted in the model-view-controller (MVC) design. MVC architecture calls for a visual application to be broken up into three separate 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.*



Classical MVC



"Apple Cocoa style"
(or PAC)

Model

- *It encapsulates the application data and logic*
- *The (non-graphical) model part of the program deals with storing, maintaining or manipulating the data.*
- *It has no reference to the view*
- *It controls access to and updates of its data*

View

Goal: to present information to the user

- *The graphical view part displays the data and provides the components for user interaction, e.g. buttons.*
- *It is agnostic of the application logic*
- *It does not store application data*
- *It contains a mechanism to be notified of the model changes*
- *It notifies its changes (user interaction) to the controller*

Controller

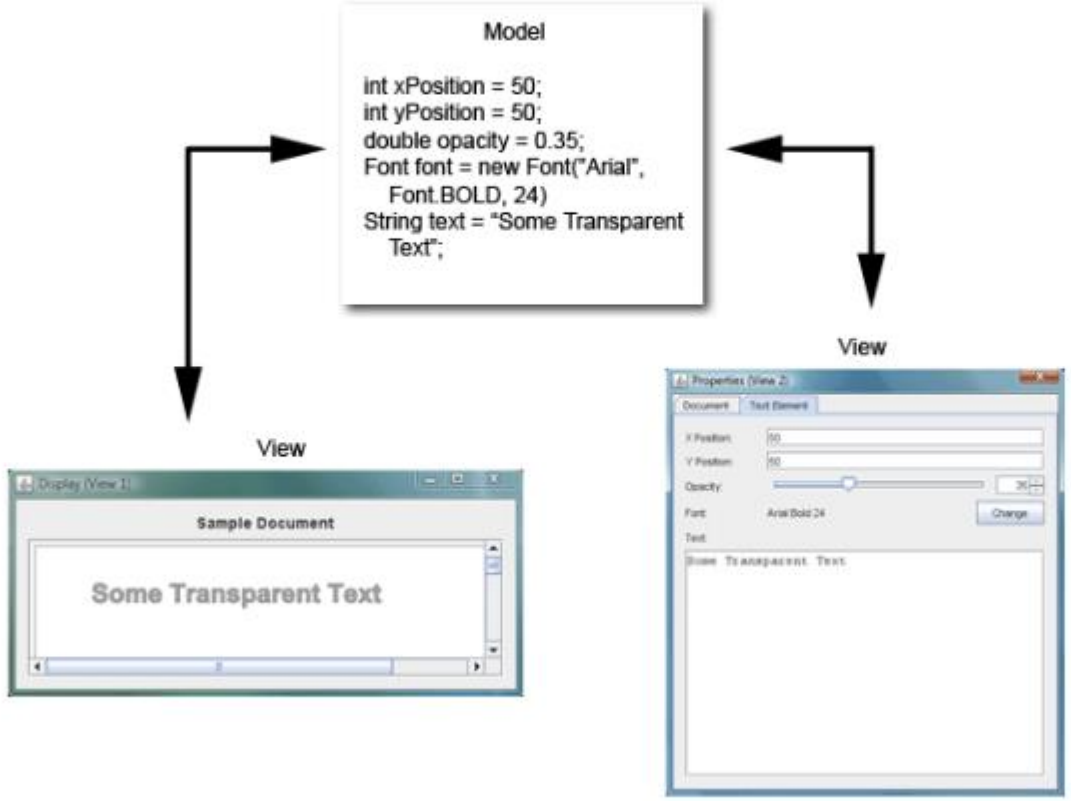
- *It is the link between the view and the model*
- *The (non-graphical) control part ensures that the user actions result in the desired responses by the program*
- *It receives notifications from the view (user interaction) and modifies the model*

MVC Advantages

- *It allows the development of software by several teams in parallel*
  - *Some developers will work on the model, other on the view.*
- *It facilitates the maintenance of software*
  - *We can change the view without re-developing the model*
- *It allows multiples views on the same model*

Multiple views on the same model

- *MVC facilitates the development of multiples views on the same model*



Source: https://miashs-www.u-ga.fr/~davidjer/inff3/2324/inff3-2324-cours9.pdf

MVC Example - View

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

class CalcView extends JFrame {
    private static final String INITIAL_VALUE = "1";

    //... Components
    private JTextField m_userInputTf = new JTextField(5);
    private JTextField m_totalTf     = new JTextField(20);
    private JButton    m_multiplyBtn = new JButton("Multiply");
    private JButton    m_clearBtn    = new JButton("Clear");
    private CalcModel m_model;
```

MVC Example – View (cont.)

```
  /** Constructor */
CalcView(CalcModel model) {
    //... Set up the logic
    m_model = model;
    m_model.setValue(INITIAL_VALUE);

    //... Initialize components
    m_totalTf.setText(m_model.getValue());
    m_totalTf.setEditable(false);

    //... Layout the components.
    JPanel content = new JPanel();
    content.setLayout(new FlowLayout());
    content.add(new JLabel("Input"));
    content.add(m_userInputTf);
    content.add(m_multiplyBtn);
    content.add(new JLabel("Total"));
    content.add(m_totalTf);
    content.add(m_clearBtn);
```

MVC Example – View (cont.)

```
//... finalize layout
        this.setContentPane(content);
        this.pack();
        this.setTitle("Simple Calc - MVC");
        // The window closing event should probably be passed to the Controller in a real program, but this is a short example.
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    void reset() {
        m_totalTf.setText(INITIAL_VALUE);
    }
    String getUserInput() {
        return m_userInputTf.getText();
    }
    void setTotal(String newTotal) {
        m_totalTf.setText(newTotal);
    }
    void showError(String errMessage) {
        JOptionPane.showMessageDialog(this, errMessage);
    }
    void addMultiplyListener(ActionListener mal) {
        m_multiplyBtn.addActionListener(mal);
    }

    void addClearListener(ActionListener cal) {
        m_clearBtn.addActionListener(cal);
    }
```

MVC Example - Controller

```
import java.awt.event.*;

public class CalcController {
    //... The Controller needs to interact with both the Model and View.
    private CalcModel m_model;
    private CalcView  m_view;

    CalcController(CalcModel model, CalcView view) {
        m_model = model;
        m_view  = view;

        //... Add listeners to the view.
        view.addMultiplyListener(new MultiplyListener());
        view.addClearListener(new ClearListener());
    }
```

MVC Example – Controller (cont.)

```
class MultiplyListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            String userInput = "";
            try {
                userInput = m_view.getUserInput();
                m_model.multiplyBy(userInput);
                m_view.setTotal(m_model.getValue());

            } catch (NumberFormatException nfex) {
                m_view.showError("Bad input: '" + userInput + "'");
            }
        }
    }
class ClearListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            m_model.reset();
            m_view.reset();
        }
    }
}
```

MVC Example - Model

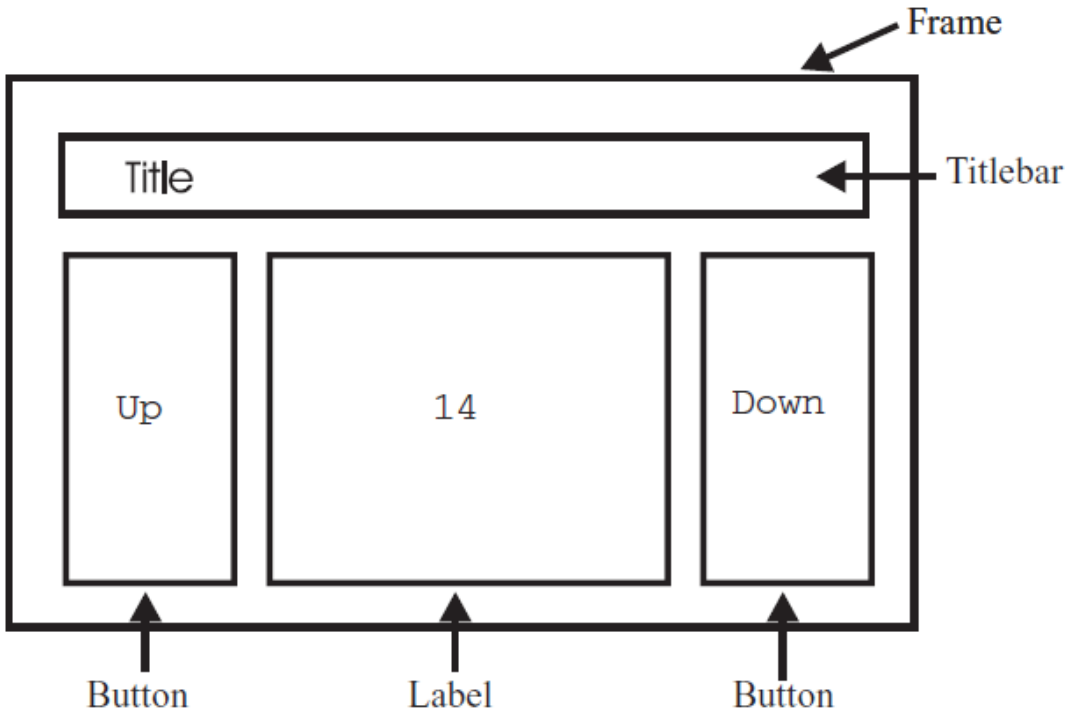
```
import java.math.BigInteger;

public class CalcModel {
    private static final String INITIAL_VALUE = "0";
    //... Member variable defining state of calculator.
    private BigInteger m_total;  // The total current value state.

     CalcModel() {
        reset();
    }
    public void reset() {
        m_total = new BigInteger(INITIAL_VALUE);
    }
    public void multiplyBy(String operand) {
        m_total = m_total.multiply(new BigInteger(operand));
    }
    public void setValue(String value) {
        m_total = new BigInteger(value);
    }

    public String getValue() {
        return m_total.toString();
    }
}
```

MVC Example - Main

```
import javax.swing.*;

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

        CalcModel      model      = new CalcModel();
        CalcView       view       = new CalcView(model);
        CalcController controller = new CalcController(model, view);

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

MVC Example 2 – Counter

- The counter has a variable *value* which is an integer. Initially the value is 0.
- The counter allows three *operations*
  - increment – increments the value of the counter by 1.
  - decrement – decrements the value of the counter by 1.
  - getValue – returns the current value of the counter.



Counter- Model

```
public class CounterModel {
  private int value;

 // The constructor initializes the counter to  0
  public CounterModel() {
    value = 0;
  }

  public void increment(){
    value++;
  }

  public void decrement(){
    value--;
  }

  public int getValue(){
    return(value);
  }
}
```

Counter- Model Test

```
public class CounterModelTest {

  private static  boolean passed = true;

  public static void main(String[] args) {
     CounterModel cm = new CounterModel();

     checkValue(0,cm.getValue());
     cm.increment();
     checkValue(1,cm.getValue());
     cm.decrement();
     checkValue(0,cm.getValue());
     for (int i = 0; i < 37; i++) {
       cm.increment();
     }
     checkValue(37,cm.getValue());
     for (int i = 0; i < 21; i++) {
        cm.decrement();
      }
     checkValue(16,cm.getValue());
     if(passed){
       System.out.println("Test passed.");
     }
     else{
       System.out.println("Test NOT passed.");
     }
  }

  private static void checkValue(int expectedValue, int observedValue){
    if(expectedValue == observedValue){
      System.out.println("Values are both equal to "+expectedValue);
    }
    else{
       System.out.println("ERROR expected value "+expectedValue+
                          " and observed value "+observedValue+" differ!");
       passed = false;
    }
  }
}
```

Counter- View – Counter Panel

```
import java.awt.*;
import javax.swing.*;
public class CounterPanel extends JPanel {

  private CounterModel counter;
  private JLabel valueLabel;

  public CounterPanel() {
     counter = new CounterModel();

     BorderLayout bordLay = new BorderLayout();
     this.setLayout(bordLay);

     JButton upButton   = new JButton("Up");
     JButton downButton = new JButton("Down");
     valueLabel = new JLabel(""+counter.getValue(),SwingConstants.CENTER);

     this.add(upButton,BorderLayout.WEST);
     this.add(downButton,BorderLayout.EAST);
     this.add(valueLabel,BorderLayout.CENTER);

     // The next three lines will later be used to incorporate the listener.
     // CounterListener countList = new CounterListener(this);
     // upButton.addActionListener(countList);
     // downButton.addActionListener(countList);
  }

  public void increment(){
    counter.increment();
    valueLabel.setText(""+counter.getValue());
  }

  public void decrement(){
    counter.decrement();
    valueLabel.setText(""+counter.getValue());
  }
}
```

Counter- View – Counter Frame and Counter Drive

```
import javax.swing.*;
import java.awt.*;
import its.SimpleFrame.SimpleFrame;
public class CounterFrame extends SimpleFrame {

  public CounterFrame() {
    CounterPanel counterPane = new CounterPanel();
    this.getContentPane().add(counterPane,BorderLayout.CENTER);
  }
}
```

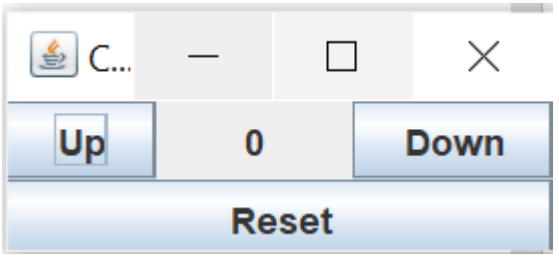
```
public class CounterDriver {
  public static void main(String[] args) {
     CounterFrame cfr = new CounterFrame();
     cfr.showIt("Counter");
  }
}
```

Counter- Control

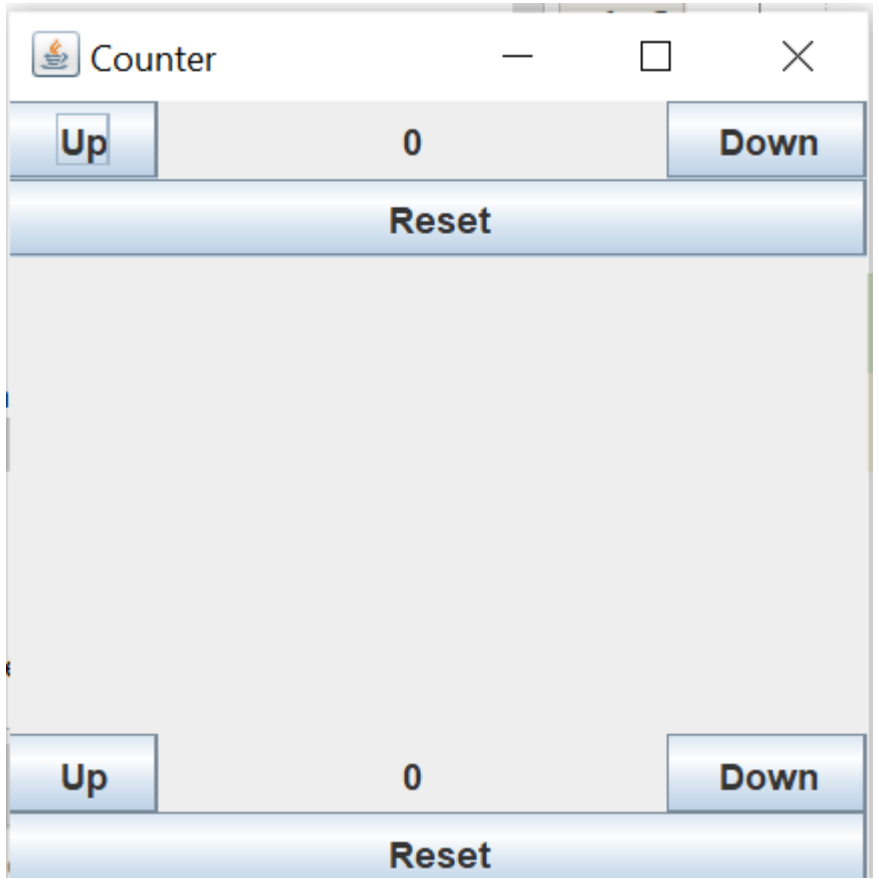
```
import java.awt.event.*;
public class CounterListener implements ActionListener{


  private CounterPanel countPane;


  public CounterListener(CounterPanel counp) {

    countPane = counp;

  }


// This method is called by the runntime system.The programmer has to add the code to be
executed as a response to the event.

  public void actionPerformed(ActionEvent evt){

    String actionCommand = evt.getActionCommand();

    if(actionCommand.equals("Up")){

      countPane.increment();

    }

    else if(actionCommand.equals("Down")){

      countPane.decrement();

    }

    else{

      System.out.println("ERROR: Unexpected ActionCommand");

    }

  }

}
```

Counter- Control updates

Update #1: Add Reset Button



Update #2: Add another 3 buttons and label.



Counter- Model update #1

```
public class CounterModel {
  private int value;

 // The constructor initializes the counter to  0
  public CounterModel() {
    value = 0;
  }

  public void increment(){
    value++;
  }

  public void decrement(){
    value--;
  }

  public void reset(){
    value = 0;
  }

  public int getValue(){
    return(value);
  }
}
```

Counter- View – Counter Panel update #1

```
import java.awt.*;

import javax.swing.*;

public class CounterPanel extends JPanel {


  private CounterModel counter;

  private JLabel valueLabel;


  public CounterPanel() {

     counter = new CounterModel();


     BorderLayout bordLay = new BorderLayout();

     this.setLayout(bordLay);
     JButton upButton   = new JButton("Up");

     JButton downButton = new JButton("Down");

     JButton resetButton = new JButton("Reset");

     valueLabel = new JLabel(""+counter.getValue(),SwingConstants.CENTER);
     this.add(upButton,BorderLayout.WEST);

     this.add(downButton,BorderLayout.EAST);

     this.add(valueLabel,BorderLayout.CENTER);

     this.add(resetButton,BorderLayout.SOUTH);



// The next three lines will later be used to incorporate the listener.

      CounterListener countList = new CounterListener(this);

      upButton.addActionListener(countList);

      downButton.addActionListener(countList);


      resetButton.addActionListener(countList);



  }
```

```
  public void increment(){
    counter.increment();
    valueLabel.setText(""+counter.getValue());
  }

  public void decrement(){
    counter.decrement();
    valueLabel.setText(""+counter.getValue());
  }

  public void reset(){
    counter.reset();
    valueLabel.setText(""+counter.getValue());
  }
}
```

Counter- Control update #1

```
import java.awt.event.*;
public class CounterListener implements ActionListener{


  private CounterPanel countPane;


  public CounterListener(CounterPanel counp) {

    countPane = counp;

  }


// This method is called by the runntime system.The programmer has to add the code to be
executed as a response to the event.

  public void actionPerformed(ActionEvent evt){

    String actionCommand = evt.getActionCommand();

    if(actionCommand.equals("Up")){

      countPane.increment();

    }

    else if(actionCommand.equals("Down")){

      countPane.decrement();

    }

    else if(actionCommand.equals(“Reset”)){

      countPane.reset();

    }

    else{

      System.out.println("ERROR: Unexpected ActionCommand");

    }
```

Counter- View – Counter Frame update #2

```
import javax.swing.*;
import java.awt.*;
import its.SimpleFrame.SimpleFrame;
public class CounterFrame extends SimpleFrame {

  public CounterFrame() {
    CounterPanel counterPane = new CounterPanel();
    CounterPanel counterPane2 = new CounterPanel();

    this.getContentPane().add(counterPane,BorderLayout.NORTH);
    this.getContentPane().add(counterPane2,BorderLayout.SOUTH);

  }
}
```