CSY2030
Systems Design & Development
Model View Controller

Reusability

- By thinking more about how things are split up and making methods and classes as minimal and generic as possible they are more reusable
- The way software is made reusable is via **separation of concerns**
  - A concern is a single set of related operations. For example:
    - Creating a GUI is a concern Processing the data is a concern
    - Loading/Saving the data from a file is a concern
    - Handling user input is a concern
    - Validating user input is a concern

Separation of Concerns - Java

- For many user applications in Java we can split the concerns into the following:
  - The user input (Action listeners)
  - Storing the application's data
  - Displaying the GUI
- The above concerns can be achieved using a software architecture called the **Model View Controller (MVC)**

Model View Controller

- There is a common software architecture called Model-View-Controller which defines this separation of concerns
- MVC defines a software component to handle each concern as well as defining how the three components are related
- MVC is made up of the following components:
  - **Model**
  - **View**
  - **Controller**

Model

- The Model deals with storing and retrieving the data that the application requires
- Models do all the processing on the data:
  - Text formatting
  - Calculations
  - Storing in files/databases

View

- A View is the Visible part of the application.
  - In most cases, the GUI
- Each view stores a specific part of the GUI.
  - This is may be a window, panel or even a single component
- The view uses a controller for its *ActionListener*
- The view gets its data directly from the model
- The view is aware of both the controller and the model

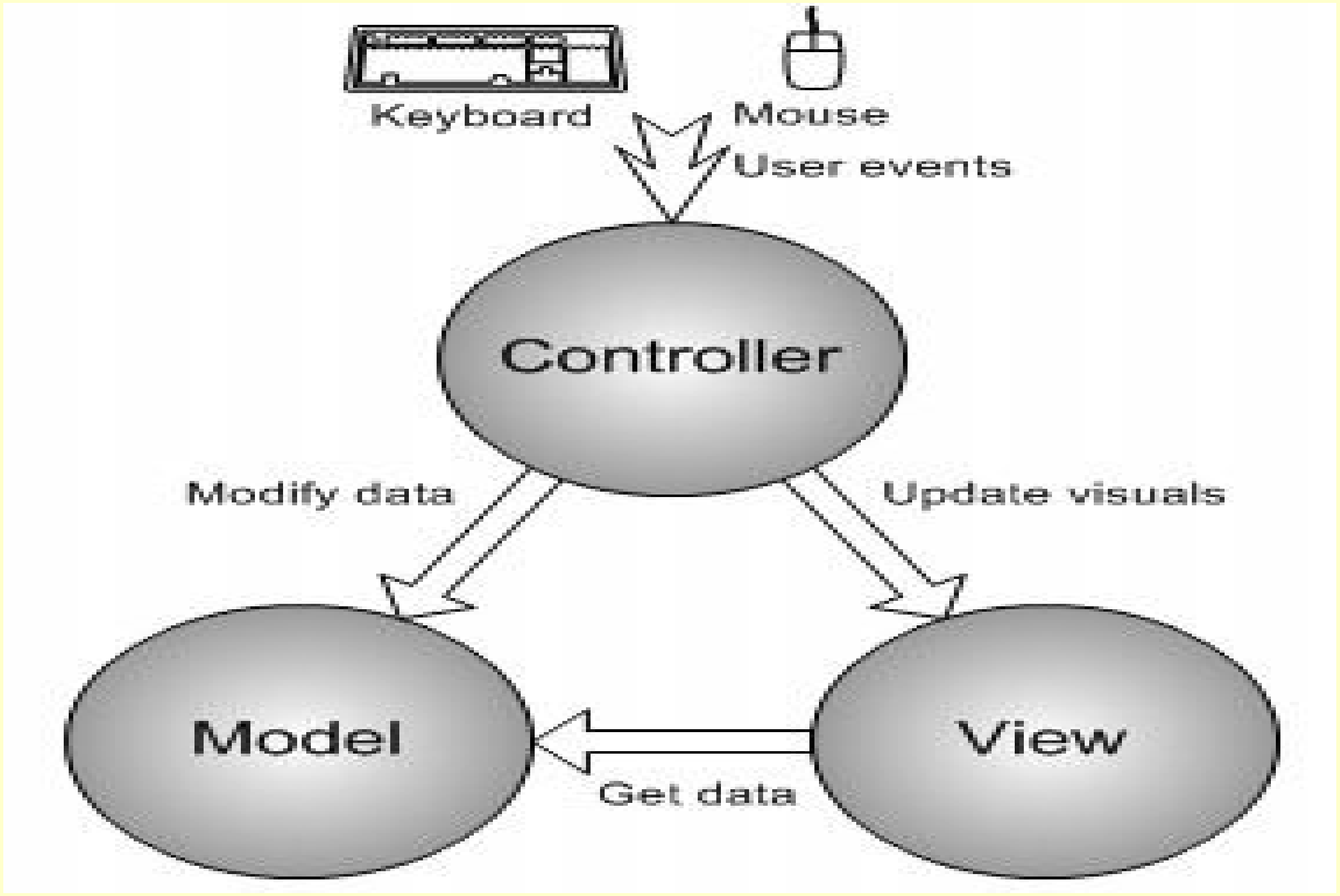
Controller

- Controllers handle all events within the application
  - It’s the user input
- They usually implement *ActionListeners* or other GUI event listeners
  - They can implement more than one Event Listener
- Controllers know about both the Model and the View
- Controllers can be linked to more than

MVC – Program Flow

- The program flow in MVC is:
  - The View (GUI) is displayed
  - The user interacts with the GUI e.g. clicking a button
  - The Controller (Action Listener) is triggered with some information about what happened in the view (e.g. which button was pressed)
  - The controller updates the model in some way
  - The view is refreshed, reading the updated data from the model

MVC – Program Flow

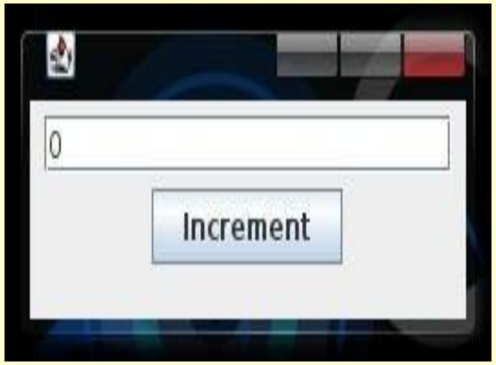


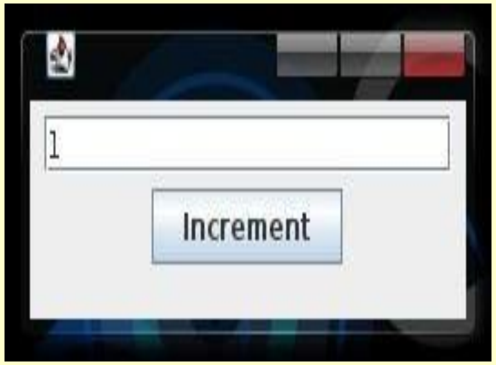
MVC – The code

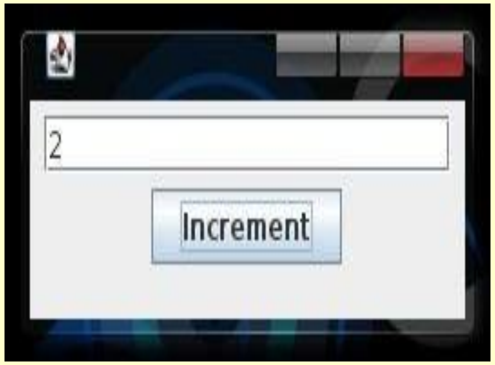
- MVC strives for Separation of Concerns and reusability of components
- As there is only one *main()* method in a program is not reusable.
- As such, none of the components contain a main method()
- The main method will create the Model, View and Controller

MVC – Basic example

- Simple example: A button that increments a counter each time it's clicked







MVC Example - Model

```
public class Model {
    private int total = 0;

   public void increment() {
     total++;
   }

   public int getTotal() {
     return total;
   }
}
```

MVC Example - View

```
public class View {
    private Model model;

    private JFrame frame;
    private JTextField text;
    private JButton button;

    public View(Controller controller,                  Model model) {
        this.model = model;

        controller.addView(this);

        frame = new JFrame();
        frame.setLayout(new FlowLayout());
        text = new JTextField(20);
        frame.add(text);

        button = new JButton("Increment");
        button.addActionListener(controller);

        frame.add(button); frame.setSize(250, 100);
        frame.setVisible(true);

        refresh();
        }

        public void refresh() {
            text.setText(Integer.toString(model.getTotal()));
        }
    }

}
```

The view needs access to both the controller and the model

However, the controller can manage more than one view at a time

The view must be assigned to the Controller

And the view must Know about the controller

MVC Example - Controller

```
public class Controller implements ActionListener {

     private ArrayList<View> views;
     private Model model;

     public Controller(Model model)
     {    this.model    = model;
          this.views    = new ArrayList<View>();
     }

    public void addView(View view) {
         this.views.add(view);
     }

     public void actionPerformed(ActionEvent e)
     {    model.increment();
          for (View v: views)
                   v.refresh();
     }


  }
```

The controller is storing multiple views in an ArrayList

When the controller is Added to the view, It assigns itself to the controller

MVC – Putting it together

```java
public class MVCExample {

    public static void main(String[] args) {
      Model model = new Model();
      Controller controller = new Controller(model);
      View view = new View(controller, model);
    }

}
```

MVC

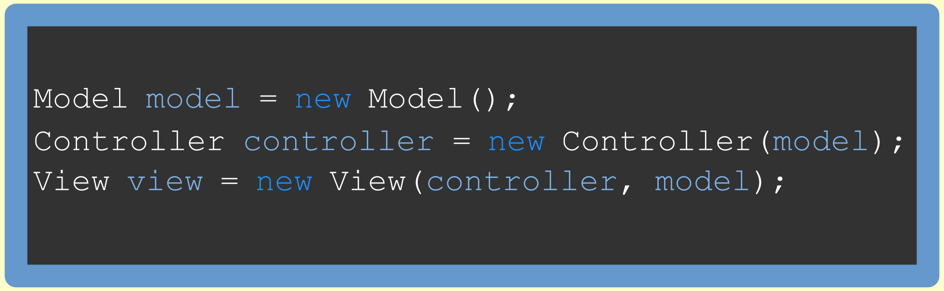
- MVC is very flexible
- Flexibility is desirable because it means making changes is easy.
- I can re-use the program, and change only the View to display the textual representation of the numbers (see next 2 slides)

MVC Example - View

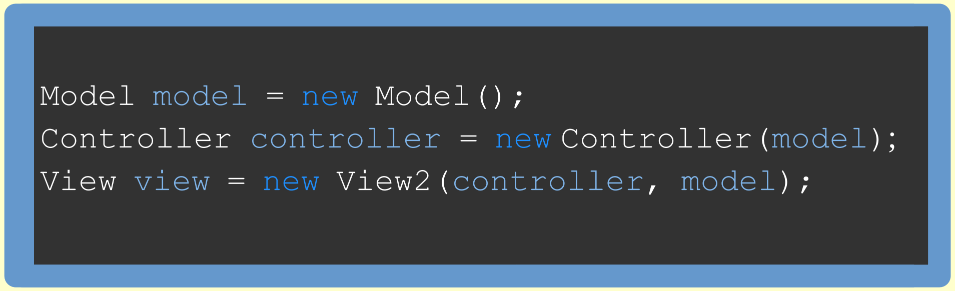
```
public  class       View2  {
        private     Model  model;

        private     JFrame  frame; JTextField
        private     text; JButton  button;
        private

        public  View(Controller  controller,                 Model  model)  {
                this.model  =  model;

                 controller.addView(this);

                 //…... refresh();
                 }


                 public  void  refresh() {
                          int  total  =  model.getTotal();
                          String[]  values  =  {"Zero", "One", "Two", "Three", "Four", "Five", "Six"};
                          text.setText(values[total]);
                 }
                 }
```

MVC Example

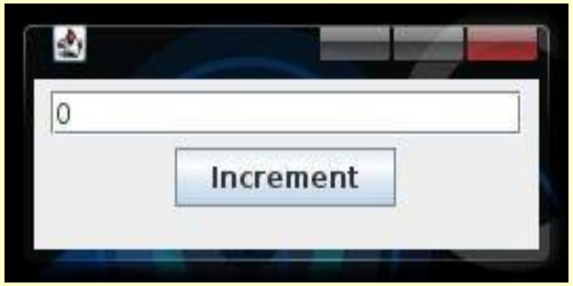
- I can now create a program to either display the textual or numerical representation by swapping out the view being used



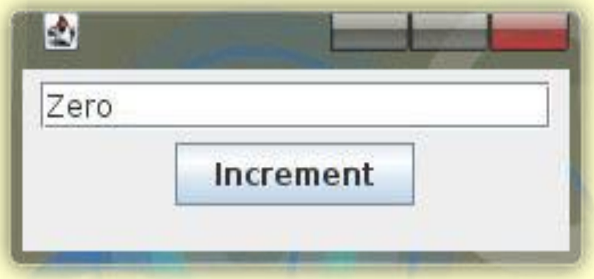
```
Model model = new Model();
Controller controller = new Controller(model);
View view = new View(controller, model);
```



```
Model model = new Model();
Controller controller = new Controller(model);
View view = new View2(controller, model);
```



0



Zero

MVC Example

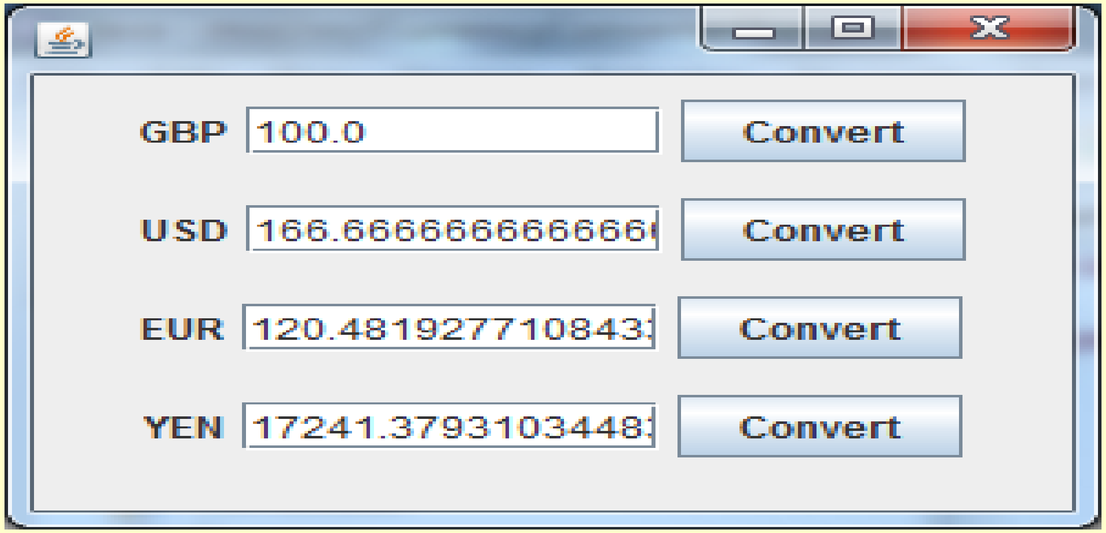
- This becomes very useful when you develop larger programs
- Larger programs tend to share pieces of functionality with other programs
- By separating the components out it allows you to easily reuse existing code in different programs
- Alternatively, you can easily implement similar functionality in the same program
- For example, a user option to decide which version of the view to display

MVC – Better example

- A good use of MVC is having a program with multiple views that use the same model and controller
- Consider a Currency Converter that converts from £ to various other currencies
- For this component the GUI will have:
  - A label for each currency
  - A text field for each currency to show the number
  - A button to convert to other currencies

MVC Example

- We will create the following Currency Exchange Rate application:



GBP	100.0	Convert
USD	166.6666666666666	Convert
EUR	120.4819277108433	Convert
YEN	17241.3793103448	Convert

- Need to split application into 3 parts i.e
  - **Model** is the exchange rates (data) and currency calculations
  - **View** is the display of 4 panels (GBP, USD, EUR, YEN) on the application window
  - **Controller** is the listeners (buttons) for view to interact with model

Application

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

public class CurrencyConverterRun {

	public static void main(String[] args) {
		JFrame window = new JFrame();                           // set up main window
		window.setLayout(new FlowLayout());                     // lay out of window is left to right

		CurrencyConverterModel model = new CurrencyConverterModel();                    // create model
		CurrencyConverterController controller = new CurrencyConverterController(model);   // create controller with model

		model.set("GBP", 100);      // model is relative to GBP

		CurrencyConverterView gbpView = new CurrencyConverterView("GBP", model, controller);     // set up GBP view
		window.add(gbpView.getPanel());                         // add GBP view to main window

		CurrencyConverterView usdView = new CurrencyConverterView("USD", model, controller);     // set up USD view
		window.add(usdView.getPanel());                         // add USD view to main window

		CurrencyConverterView eurView = new CurrencyConverterView("EUR", model, controller);      // set up EUR view
		window.add(eurView.getPanel());                         // add EUR view to main window

		CurrencyConverterView yenView = new CurrencyConverterView("YEN", model, controller);      // set up YEN view
		window.add(yenView.getPanel());                         // add YEN view to main window

		window.setSize(300, 500);                   // set size of main window
		window.setVisible(true);                    // display all the views on the main window
	}
}
```

Model

```
import java.util.HashMap;

public class CurrencyConverterModel {
        private double gbpValue = 0.0;                              // stores the conversion value relative to GBP
        private HashMap<String, Double> rates;                      // stores pairs of values e.g store “GBP” with 1

        public CurrencyConverterModel() {
                rates = new HashMap<String, Double>();              // create a list of pairings
                // add 4 pairings to model  – note exchange rates relative to GBP
                rates.put("GBP", 1.0);
                rates.put("USD", 0.6);
                rates.put("EUR", 0.83);
                rates.put("YEN", 0.0058);
        }

        public double getTotal(String currency) {
                double rate = 1/rates.get(currency);    // get method will get value associated with currency
                return this.gbpValue * rate;
        }

        public void set(String baseCurrency, double amount) {
                double rate = rates.get(baseCurrency);
                this.gbpValue = amount * rate;
        }
}
```

View

```
import javax.swing.*;

public class CurrencyConverterView  {                    // this sets up a view for each currency
        private JPanel panel;
        private JLabel label;
        private JTextField text;
        private JButton button;

        private CurrencyConverterModel model;
        private CurrencyConverterController controller;

        private String currency;

        public CurrencyConverterView(String currency, CurrencyConverterModel model, CurrencyConverterController controller) {
                this.model = model;
                this.currency = currency;
                this.controller = controller;
                this.controller.addView(currency, this);
                this.panel = new JPanel();
                this.button = new JButton("Convert");
                this.button.addActionListener(this.controller);
                this.button.setActionCommand(this.currency);
                this.label = new JLabel(currency);
                this.panel.add(label);
                this.text = new JTextField(10);
                this.panel.add(text);
                this.panel.add(button);
                refresh();
        }

        public String getValue() {
                return this.text.getText();
        }

        public void refresh() {
                double total = this.model.getTotal(this.currency);
                this.text.setText(Double.toString(total));
        }

        public JPanel getPanel() {
                return this.panel;
        }
}
```

Controller

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.HashMap;

public class CurrencyConverterController implements ActionListener {
	private HashMap<String, CurrencyConverterView> views;		// create a list of currency and view pairs
	private CurrencyConverterModel model;				// store data

	public CurrencyConverterController(CurrencyConverterModel model) {
		this.model = model;						// set up data
		this.views = new HashMap<String, CurrencyConverterView>();	// set up views as pairings for each currency
	}

	public void addView(String currency, CurrencyConverterView view) {
		this.views.put(currency,view);		// add view to application
	}

	public void actionPerformed(ActionEvent e) {
		CurrencyConverterView callingView = this.views.get(e.getActionCommand()); // add listener to view’s button
		// set action associated with pressing button
		this.model.set(e.getActionCommand(), Double.parseDouble(callingView.getValue()));
		// refresh each view after button pressed
		for (CurrencyConverterView v: views.values())
				v.refresh();
	}
}
```