Model View Controler

Who Does What in GUI Program

- The Model – View – Controller style of programming is a why of deciding who does what in a GUI program.
- Model – The application logic, independent of user interaction
- View – The graphical display to the user, the widgets.
- Controller – The code for dealing with user input, which appears as events.

- GUI programs can quickly become unreadable
  - Unless we adopt a logical style to divide up the code.

Simple GUI Programs

- In a program like `GuessGUI` the `GuessGUI` class can do everything.
  - The application logic is simple
  - There are only a few GUI components.
  - The user input is simple.
- The `main` program will just create a `GuessGUI` object.
- The GUI display is set up in the constructor.
  - The View
- The user input is delivered to `actionPerformed`.
  - The Controller
- The simple program logic will be in `actionPerformed` and helper methods.
  - The Model

Slightly Longer Programs

- When the application logic gets slightly more complicated.
  - It makes sense to have a separate model class.
- The View and Controller are handled by one class.
  - The View is set up in the constructor.
  - The Controller is the `actionPerformed` method.
- The main program creates two objects.
  - The Model object.
  - The View - Controller object.

Interactions Between Model and View-Controller

- The View needs to know about the Model.
  - It displays information about the Model.
- The Controller needs to know about the Model.
  - User input will change the Model.
- The Model does not need to know about the View – Controller.

- Main creates the Model object first.
  - And passes it as a parameter to the View – Controller constructor.
- The View – Controller object then remembers the Model for future use.

MinMaxAve Example

- The model is an object that is given numbers, one at a time.
- It remembers the minimum, maximum and average of the objects it has seen.

- The user interface lets the user type in a series of numbers, one at a time.
- It will display the minimum, maximum and average of the numbers it has seen so far.
- The user can also clear the numbers and start again.

M – V – C Architecture: Model

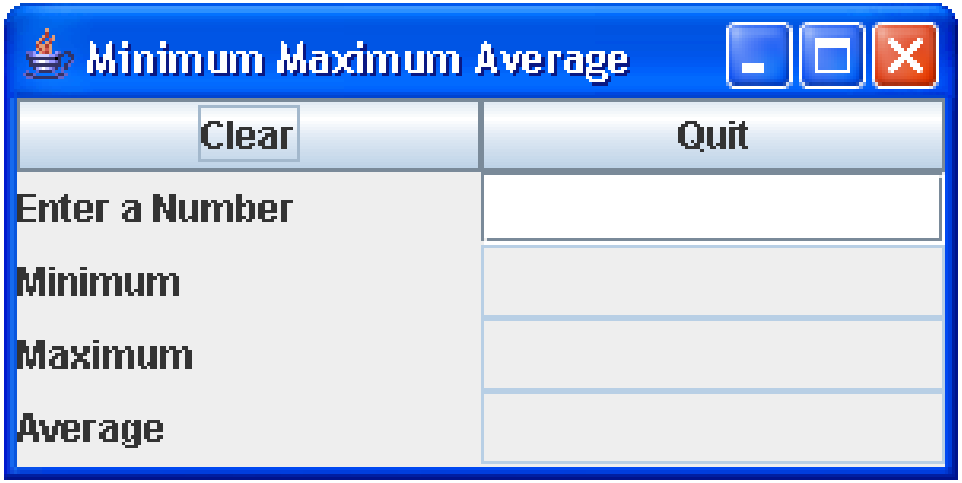
- We can call the model class `MinMaxAve`.
- It will have the following methods
  - Constructor
  - `addNumber`.
  - `getMin`
  - `getMax`
  - `getAve`
  - `clear`.
- We can write this class without thinking about the user interaction.

M – V – C Architecture: View - Controller

- We can call this class GUI.
- It is a View because it extends `JFrame`.
  - In Java a view is provided by a `JFrame` object
  - The GUI constructor will create the view.
- It is a Controller because it implements `ActionListener`.
  - In Java a controller is called a listener.
  - class GUI will have a method called `actionPerformed`.
- The GUI constructor will have a parameter of type `MinMaxAve`.
  - It will save it in an instance variable for later use.

class GUI

- We can design the GUI by sketching it.
- We need two buttons for Clear and Quit.
- We need a label and a text field to enter numbers.
- We need three label / text field combinations to display information about the model.



M – V – C Architecture: Main

- The main program will
  - Create a `MinMaxAve` object first.
  - Create a `GUI` object with the `MinMaxAve` object as a parameter.
  - That's all.
- We have now made the most important decisions about this program.
  - Who is responsible for what.

- `MinMaxAve` knows how to calculate minima, maxima and averages.
- `GUI` constructor knows how to display details of `MinMaxAve`.
- `GUI` action performed knows how to process the user input.
- Main knows how to integrate `MinMaxAve` and `GUI`.

Code Highlights

- The `main` program creates the two objects in the right order.
- It passes `mma` as a parameter to `g`.

- `GUI` extends `JFrame` and implements `ActionListener`.
  - It is both a view and a controller.
- The `GUI` constructor stores the `MinMaxAve` parameter in an instance variable called `mma`.
  - It is used to call methods in `actionPerformed`.
- The size is worked out by trial and error.
  - Initially it was 500 x 500.

Code Highlights (2)

- The default Border Layout is replaced by a 5 x 2 Grid Layout.
- Each widget or group of widgets is created and set up in sequence.
  - This makes the widget creating easier to read.
  - It is very easy for widget creation to become unreadable.
  - We should work hard on the legibility of this part of the code.
- We only listen to three of the widgets.

- `actionPerformed` calls methods of `mma` both
  - To change the model via `addNumber` and `clear`
  - To access information about the model (to change the view) via `getMin`, `getMax`, `getAve`.

Code Highlights (3)

- class `MinMaxAve` remembers (as instance variables)
  - `min` – smallest number added so far.
  - `max` – largest number added so far.
  - `count` – how many numbers added so far.
  - `total` – total of numbers added so far.
- The first number added is special.
  - Recognise it because `count == 0`.
  - Set `min` and `max` to it.

Code Highlights (4)

- Subsequent numbers
  - Check to see if n, the current number being added is smaller than min.
    - If so, it is the new min.
  - Similar for max.

- The accessors getMin, getMax, getAve return the special value -1 if no numbers have been added.
  - This is recognised by count == 0.

- Note that we calculate the real average.

Larger Programs

- The View and Controller can become quite large.
- It then makes sense to have separate objects for this functionality.

- This lets us have more than one View of the Model.

- More Later.