University of Stirling
Computing Science and Mathematics

CSCU9P6 Software Engineering II
Practical: week starting 4th February 2019

The MVC architecture

1. Open K:\CSCU9P6\`Practicals` and find the **folder** `MVC-Example`.

   `MVC-Example` is an adaptation of the general MVC framework for two controllers and two views per controller.

2. As before, create a new Together project by creating a new folder somewhere suitable, copying the files from the `MVC-Example` folder to a new `src` subfolder, and using **File/New/Project/Modelling/Java Modelling Project**.

3. Compile, run and play — you will see the two controllers' frames appear on the screen. Each controller has a button that sends "clear" messages directly to *its own two views*, which are simply `JPanel`s within the controller's `JFrame`. The database (model) has an A counter and a B counter. Each controller has one view that shows the A component of the database, and one that shows the B component. `Controller1` has an "Increment A" button that sends a modify message to the model (incrementing internal counter A in the model), which causes *both* views shown by `Controller1` to be updated via the subscribe/notify protocol – although, of course, only the A view will show something new! It also has a Quit button (so no window closing reaction is required — instead clicking the window close box has no effect!). Ideally, `Controller2`'s views should also be updated automatically via the subscribe/notify protocol, but this has not been implemented yet, and `Controller2` has a "Refresh views" button to explicitly force its views to update themselves from the model.

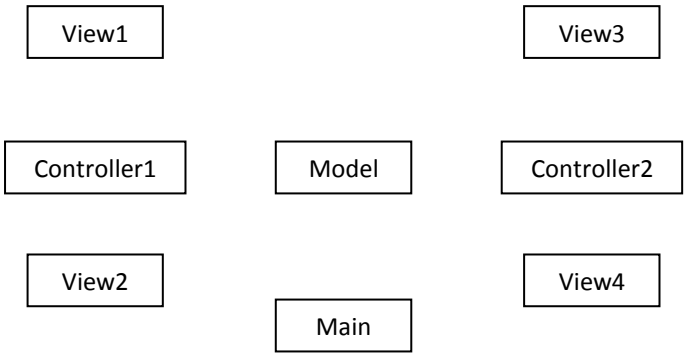
4. Take a look at the class diagram and code through the Together interface.

   - You may not be able to see all the class diagram at once, in which case you will find Zoom options in the Diagram menu, and controls 

above the Diagram pane that let you adjust the display (or use the +, -, * and / keyboard short-cuts). There is also an Overview that lets you scan around the class diagram: To show it, click on the Eye button  on the toolbar above the Diagram pane. In the small window that pops up you can drag the shaded area around. Press Escape or click anywhere else to dismiss the Overview.

   - You might also find it useful to alter the amount of detail on display in the class diagram: Click the Diagram Preferences button  above the class diagram, or select Preferences at the bottom of the Diagram menu, look for View Management in the left hand column of the preferences dialogue that pops up. You can choose Analysis, Design or Implementation from the Diagram level section. You can preview the effect by clicking Apply before finishing.

   - You may need to resize/reposition the classes in the diagram: Try the automatic Layout facility in the class diagram toolbar:  – it does **not** seem to do a good job with these classes, so let's try something else: Drag the classes into roughly the positions below, on next page (by trial and error, this arrangement works quite well):



- You will probably find that the association links do not look good. You can straighten them out using the Order Links options: button  in the Diagram pane toolbar, or in the Diagram menu.

5. Carry out the following modifications through the Together interface. Compile, test and observe carefully after each step:

   - Following the example code in `View1`, modify `View3` and `View4` to properly partake in the subscribe/notify protocol with the model: They need to implement the `Observer` interface; they need to subscribe to the model, and their `update` methods should have method headers compatible with the `Observer` interface. At the same time you can comment out from `Controller2` everything to do with the "Refresh views" button, as it will become redundant.

   - Instantiate `Controller2` *several times* in `Main` — it should be easy to duplicate them and have them all work properly: Their views all subscribe to the model and they all get notified when the model is modified via the "Increment A" button in `Controller1`. [You could also have multiple instantiations of `Controller1` if you wished.] Note that since the `Controller2` constructor specifies where the frame is located, all three frames will appear together on the screen – you will need to move them to see them all!

   - It would be cosmetically nicer if `Controller1`'s and `Controller2`'s *constructors* received an extra parameter, a `String`, that they used in setting up their `Frame`'s title, so that `Main` can ensure that all the instances of all the controllers are clearly distinguished on screen. Do that now. Remember that each constructor *call* in `Main` will need to be modified too to pass an extra actual parameter! [The method call for setting the frame title is already present in the controller constructor bodies, it just needs altering.] It would also be nice if each controller received additional constructor parameters to indicate where on the screen it should be located.

   - Add a button to `Controller2`, to cause a new method in `Model` to be invoked that increments the second counter (`dataBaseB`) – also making sure that all observing views get updated appropriately.

> **Checkpoint:** Now show your fully functional MVC application to a demonstrator: Show how all relevant views are updated when the buttons are clicked. Explain the whole chain of steps from clicking on the "increment B" button to Controller1's display of B being updated.

7. [Rather more intricate] By cloning `Controller2` and `View3`, build a new controller with, say, one view that displays *either* the A database component *or* the B component. The controller should have two buttons: one to switch the view to display A, and one to switch it to display B. [Hint: The view can have a single `boolean` variable to indicate which database component it is supposed to be displaying. The controller's button responses cause the `boolean` to be set/unset – by calling a method in the view. The view's `update` method fetches data as indicated by the `boolean`.]

That's all

SBJ January 2019