University of Stirling
Computing Science and Mathematics

CSCU9P6 Software Engineering II
Practical: week starting 11 February 2019

Collaborative working with Subversion/Eclipse/Subclipse

This week the practical work is a group exercise, with group members working collaboratively on different parts of a single multi-class Java project. A central shared copy of the project for each group will be held on a Subversion server, where each group has its own repository. Individual members will check out their own working copy of the project using the Subclipse plug-in for Eclipse, and will then regularly check in (commit) their updated Java code to the repository, and check out again any Java updated by the other group members. Once all the separate updates have been merged, there will be a completed running application.

There are three videos demonstrating the key steps when using Subclipse, together with the "scripts" for those demonstrations, on the module Canvas Practicals page, and also in `K:\CSCU9P6\Practicals\CollaborativeWorking.html`. Take a look now if you have not done so already. *Headphones are required for the audio in 4X5.*

1. Each member of your group should log on to a PC with their usual CS username and password.
2. Launch ***Eclipse*** from the **Start** menu. If you are offered a choice of workspace, then **DO NOT** choose your Together workspace – they are incompatible.
3. Now, following the demo "Connecting to a Repository", connect to your group's repository at `svn://svn.cs.stir.ac.uk/p6-nn` – where `nn` is your group number (e.g. `03`). For authenticating your access to the repository your username is your usual three letter CS username, and your password will be given to you.
4. Note: You should change your Subversion password fairly soon by visiting `http://svn.cs.stir.ac.uk/cgi-bin/cpf.pl`
   The password that you choose should not be one that you use anywhere else, as it is not held by Subclipse in a highly secure way.
5. **One member of your group should now:**

   In the usual way, build a new Eclipse project in **their own file store** using the files from the folder `K:\CSCU9P6\Practicals\MVC-Example` as the initial content of the project's `src` folder. Check and un-set the read-only attributes on the files if necessary.

   Then following the demo "Sharing an Eclipse Project", set up their `MVC-Example` project for sharing on the group repository.

   The project is now shared and ready for checking out by other members of the group. From this point on, the initial uploading member has no special status – everyone is just sharing the files.

6. **Each other member of the group should now:**

   Activate their SVN Repository Exploring perspective, right-click in the SVN Repositories view on the left, Refresh the repository, expand, Right-Click the shared project name, and select Checkout...

   Accept the default settings in the Checkout As... dialogue, click Next, either accept the default location for the folder that will be downloaded or uncheck the box and browse to a new location – your choice.

   Then click Finish. The project will be downloaded, and is now ready for working on in the Java perspective.

7. You now have your **own working copy** of the project. You can compile, run and play with it — you will see the two controllers' frames appear on the screen. It is the same as last week's MVC exercise: 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.

8. **Each member of the group should carry out a different aspect of the following work on their own PC.**

9. **See the demo "Updating and Synchronizing an Eclipse Project". You should "Synchronize with Repository" regularly so that your work is available to everyone else, and so that their work is incorporated into your own working copy.**

   You can decide amongst yourselves who does what.

10. Carry out the following modifications:

    - 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.] Each controller should also receive 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: The group should now show its fully functional MVC application to a demonstrator: Show how Subversion is holding a copy of the project, and describe your experience of using Subclipse.

That's all!

SBJ January 2019