Laboratory Sheet 1

This Lab Sheet contains material based on Lectures 1 – 2.

The deadline for Moodle submission of the lab exercise is 24 hours after the end of your scheduled laboratory session in week 2 *(29 September – 3 October 2014).*

You may submit work that is incorrect or incomplete. In order to stretch the stronger members of the class, some of the laboratory exercises are quite challenging so don't worry if you can't complete all of them. You should spend around 3 hours per week on programming exercises.

Default login details: Username = matric + 1st initial of family name, password = last 8 digits of library barcode.

Beginners – start at 'Set Up' Section on page 1.

Experts – skip to 'Submission material' Section on page 5.

Aims and objectives

- Familiarisation with the Eclipse IDE for creating Java projects, and editing, compiling and running Java programs.
- Writing very simple Java programs that consist of just a single class with a single (main) method.

Set Up

When you download Laboratory1.zip from moodle, please unzip this file. You will obtain a subfolder `Laboratory1`. Four sub-folders for the projects `FirstProgram`, `Primes`, `Submission1_1` and `Submission1_2` will be created, and the appropriate Java source code file will be inside each of these folders.

Using Eclipse

Eclipse is an open-source *Integrated Development Environment (IDE)* that will be used for coding, compiling, running, and debugging Java programs. Eclipse provides many more facilities than this, some of which you will use in later courses. It is a many-featured and versatile piece of software, and can appear a little daunting for beginners. We will be using only a very small subset of its capabilities, and a key objective of this lab session is to begin the process of familiarisation with these.

1. Follow the setup above to extract the source code files into the appropriate folders.

2. Launch Eclipse by double clicking on the dark blue Eclipse shortcut icon on the desktop. Eclipse takes a few seconds to start up. You will see Eclipse's 'welcome' screen appear - see Figure 1.

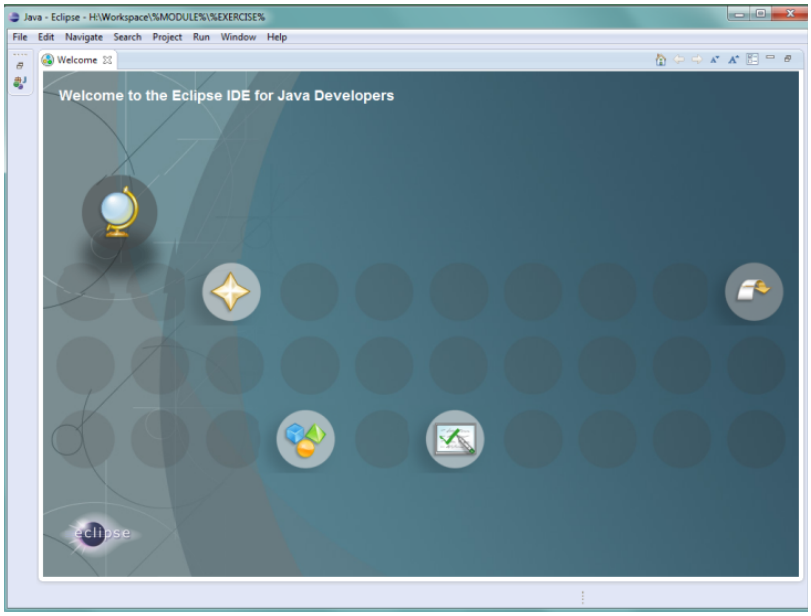

Figure 1. Eclipse's Welcome Screen

3. Clicking on the Workbench arrow (arrow in circle on right hand side) will take you to the default *Java Perspective*, which is subdivided into smaller windows called *Views*. Views display different information about your work. A *Perspective* in Eclipse is a configuration of views. By default, Eclipse shows the Java perspective, which is the one we will use almost exclusively. Figure 2 shows the appearance of the Java perspective when no projects or files are open.

4. When working on a particular program, you have to indicate to Eclipse the folder that contains the files for that program. This is known to Eclipse as the *workspace* folder. For each Laboratory, you will set the workspace to be the subfolder with the name corresponding to that Laboratory – so in this case, the folder Laboratory1. To do this, choose File → switch workspace → other . . ., and in the window that appears, use the browse facility to locate the Laboratory1 folder and set it to be your workspace. Note that when you switch your workspace, Eclipse will restart. You are also likely to find that whenever you log in afresh to the system, you will have to set your Eclipse workspace to the desired folder.

5. Each Java program that you work on will form an Eclipse *project*, and this project has to be created. Typically, some or all of the files that are required for a project will be downloaded, at least in skeleton form, when you download the lab zipfile from moodle. Each project will have its own folder within the appropriate Laboratory folder, the name of the folder necessarily being the same as the name of the project. The folder, and initial contents, will be created on set up, but you will have to create the project.

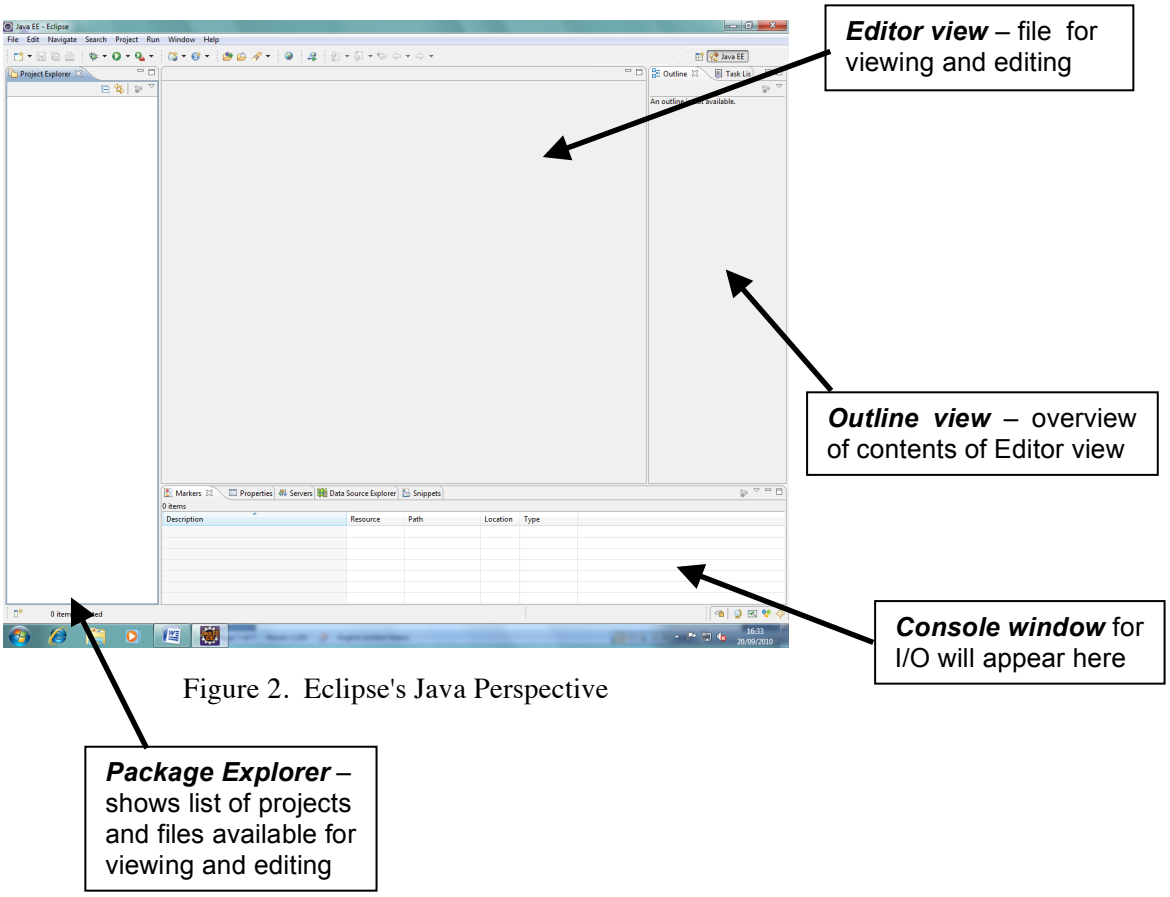

Figure 2. Eclipse's Java Perspective

6. In Eclipse, select File → new → java project to launch the project creation wizard. You have to enter a project name, in this case FirstProgram. (You *don't have* a choice of project name - it must be the name of the folder containing the source code.) Check that 'Use Default Location' is selected, and check that the project folder name is ...\Laboratory1\FirstProgram. Then simply click the Finish button, as the remaining steps in the wizard are unnecessary for your purposes.

7. In the *Package Explorer view* (on the left of the Eclipse window), you should now see an icon representing this project. Clicking on the expander icon reveals the contents of the project – namely components of the Java Run-Time Environment (JRE) together with the *default package*, which contains the sole source code file for this program, i.e. FirstProgram.java. Expanding the default package icon will reveal this file. Then double click the file name to open the file in the central *Editor view*. You will see the source code for the 'Hello World' program, and in the Outline view you will see a summary (brief in this case) of the methods in this class. See Figure 3 (which shows the appearance of the screen after the program has been run).

8. Now click on the Editor view to activate it. Whenever Eclipse detects an error in the java file displayed in the editor view, this is indicated by red underlines in the text, and red crosses to the left (and possibly also the right) of the view. Hovering the cursor over a marker throws up an error message – see below. This feature of Eclipse takes a bit of getting used to; it can be annoying as the compiler often does not give you time to finish typing a line before throwing up some spurious error indicator.

9. The program here is complete and contains no errors, hence Eclipse does not complain. But nonetheless, it has done its work behind the scenes, and you can run the program. To do this, first make sure that the Editor view is active (as indicated by a blue border). Then select the Menu option Run → Run as → Java application, or click the Green 'Play' button in the

toolbar. You should see the output from the program in the *Console view*, at the foot of the Eclipse window.

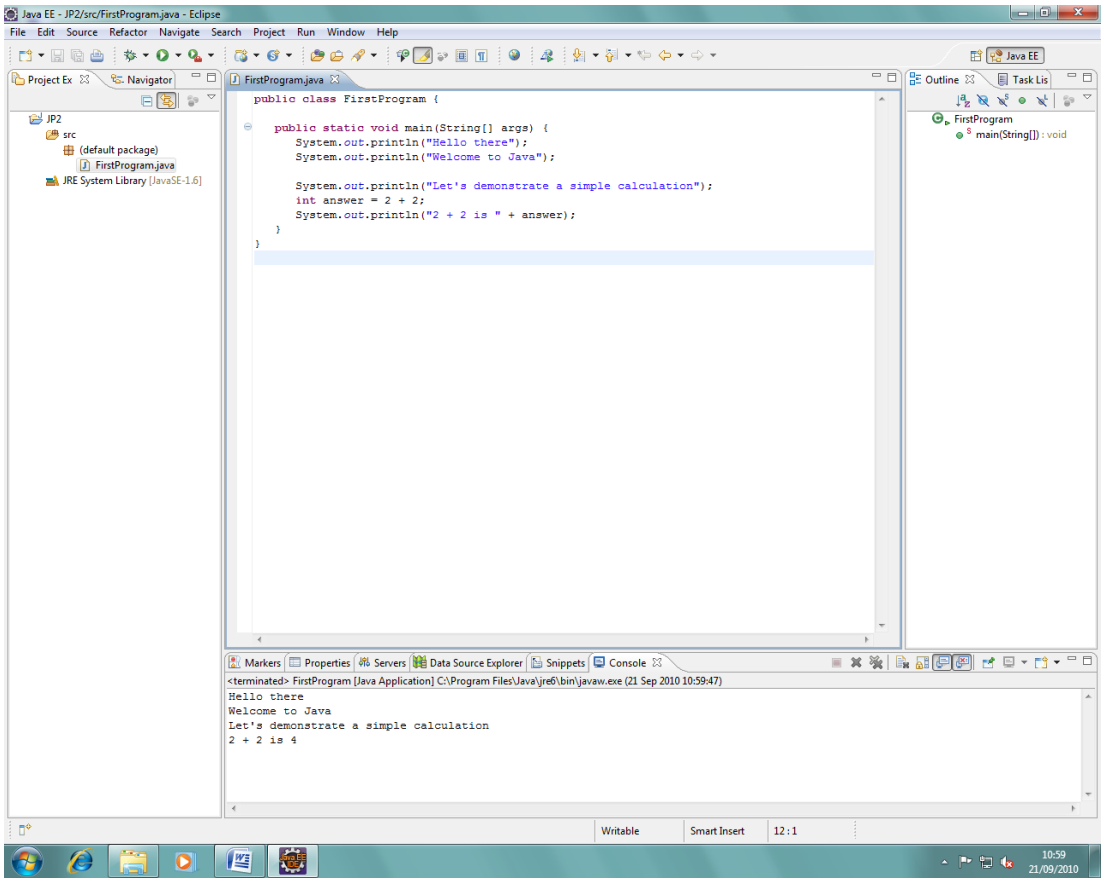

Figure 3. Eclipse displaying FirstProgram execution.

10. Now introduce an error into the Java code. For example, delete one of the letters from the word `static`. You will see a red cross appear on the extreme left of the Editor view, and moving the cursor over this marker reveals a helpful error message. (Error messages are not always as immediately helpful as this.) Experiment with some further errors.

11. Go through the same process with the Primes program, creating a new java project named Primes and opening the source file Primes.java in the Editor view. When you run this program, a suitable choice of input value is required. To provide this input, you need to click on the Console view to make this active and then type in an appropriate value, followed by Return. Again experiment with some artificially introduced errors.

12. You may have noticed that when the cursor hovers over an identifier (in the Editor view) information about that identifier is displayed. You will discover a variety of useful features of the Eclipse editor in due course.

13. You are now ready to work on the programming exercises below that form the submission material for this lab. In each case you have to create a project for the program, by following steps analogous to those above, add your own code into the skeleton provided, and then work towards running and testing the program.

14. One further useful feature of Eclipse: selecting Source → Format formats the code in the active window neatly, using default formatting conventions.

Submission material

Preparatory work for these programming exercises, prior to your scheduled lab session, is expected and essential to enable you to submit satisfactory solutions.

Unit Tests: I have supplied some simple test cases to check that your programs are working properly, for Submission1_1 and Submission1_2. When you create these projects, you should see two source code files in each project – one is the skeleton Java source code for you to fill in the details. The other file contains the Unit tests to check your programs are working properly. In the (likely) event that Eclipse complains about your Unit tests, do the following:

In the package explorer pane (top left):
Right click on package > build path > configure build path > libraries > add Library > JUnit. Click Finish.
Now all the Eclipse build errors should go away.

To execute Submission1_1:
right click on AverageSpeed -> run as Java application
right click on AverageSpeedTest -> run as JUnit test

Submission 1

Write a complete Java program to meet the following specification. The program is to compute and report the average speed of a vehicle over a distance of one mile, given the times at which the vehicle passed cameras at the start and end of the measured mile.

Input to the program should come from the keyboard. The first line contains the time at which the vehicle passed the first camera, represented in the form hh mm ss, using the 24 hour clock. The second line contains the time at which it passed the second camera, in the same format. The user should be prompted to provide each line of input. The output should be the average speed of the vehicle in miles per hour, represented as an integer.

For example, for the input dialogue

```
Type the first time:  17 23 56
Type the second time: 17 24 56
```

the corresponding output would be

```
Average speed = 60 mph.
```

Hint 1: Recall the relationship

speed = distance / time,

so that

speed in miles per hour = (distance in miles) / (time in hours)

and therefore

speed in miles per hour = (distance in miles) / (time in seconds / 3600)

Hint 2: Remember to allow for the fact that midnight might fall when the vehicle is between the two cameras. (You may wish to discount this possibility in a first version of your program.)

Hint 3: Use the `Scanner` class to handle the input, particularly the `nextInt()` method.

Submission 2 (more challenging)

Write a complete Java program to meet the following specification. The program is to read in a date, in the 20th or 21st century, in the form `dd mm yyyy`, and should report the day of the week associated with that date.

For example, for the input dialogue:

```
Enter a date: 29 09 2008
```

the corresponding output should be

```
29 09 2008 is a Monday.
```

Hint 1. One possible method would use the fact that 1st January 1900 fell on a Monday, and would compute the number of elapsed days from that base date.

Hint 2. A year is a leap year if it is exactly divisible by 4, except that if it is divisible by 100 it must be divisible by 400; so 2000 was a leap year but 1900 was not.

Hint 3. For actions that require choosing from a set of options, you could try using an appropriate `if` statement, or a `switch` statement (e.g. see http://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html).

Hint 4. You may have to improvise to cope with the variable number of days in each month. Neat ways of doing this would use features of Java not yet encountered (`HashMaps` or array-based lookup tables, perhaps). Recall the old rhyme: "30 days hath September, April, June and November, all the rest have 31, excepting February alone, which has 28 days clear, and 29 in each leap year".

Note: A lazy software engineer might simply reuse library code in `java.util.Date` or `java.util.Calendar`. However we *won't* give you full marks if you use this approach.

How to submit

You should submit your work before the deadline no matter whether the programs are fully working or not.

When you are ready to submit, go to the JOOSE2 moodle site. Click on Laboratory 1 Submission. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code ...\Laboratory1\Submission1_1\ and drag only the *single* Java file AverageSpeed.java into the drag-n-drop area on the moodle submission page. Then do the same for file DayOfTheWeek.java in ...\Laboratory1\Submission1_2\. **Your markers only want to read your java files, not your class files.** Then click the blue save changes button. Check the two .java files are uploaded to the system. Then click submit assignment and fill in the non-plagiarism declaration. Your tutor will inspect your files and return feedback to you via moodle.