Java Programming 2 – Lab Sheet 1

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

The deadline for Moodle submission of this lab exercise is 5pm on Thursday 28 September 2017.

Default login details: Username = matric + 1st initial of family name, password = last 8 digits of library barcode (please change your password!).

Beginners: start at "Using Eclipse" on page 1.

Experts: follow steps 1--8 under "Using Eclipse" below to launch Eclipse and unzip the starter code, and then go to "Submission Material" on page 5.

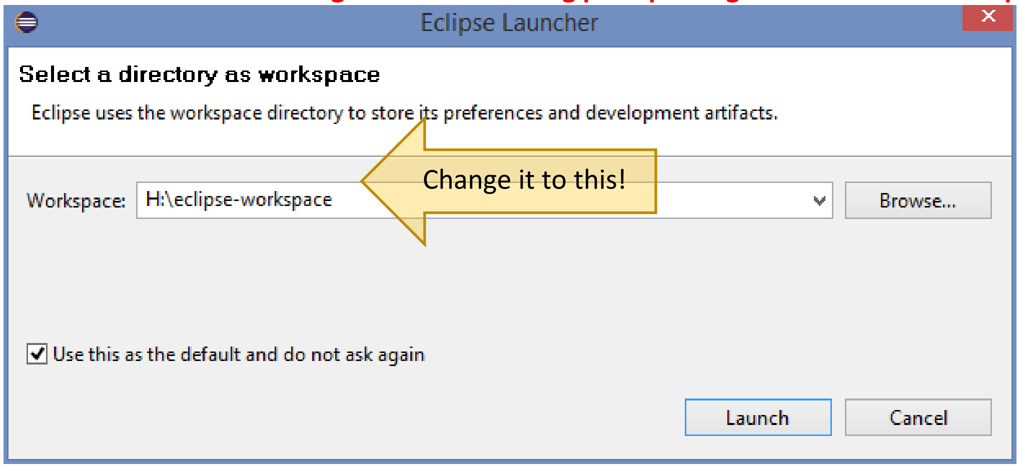
Aims and objectives

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

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 small subset of its capabilities, and a key objective of this lab session is to begin the process of familiarisation with these.

1. Launch Eclipse by clicking the Windows menu and choosing "Eclipse Java Oxygen". After it launches, you will first be asked to choose a **workspace folder** – this is the folder that will contain all of the files you are working on. **When you run Eclipse, you MUST modify the "Workspace" field to point to "H:\eclipse-workspace" or else your work will not be available when you log into a different machine. You might also want to tick the "Use this as default and do not ask again" to avoid being prompted again to choose a workspace.**



Eclipse Launcher

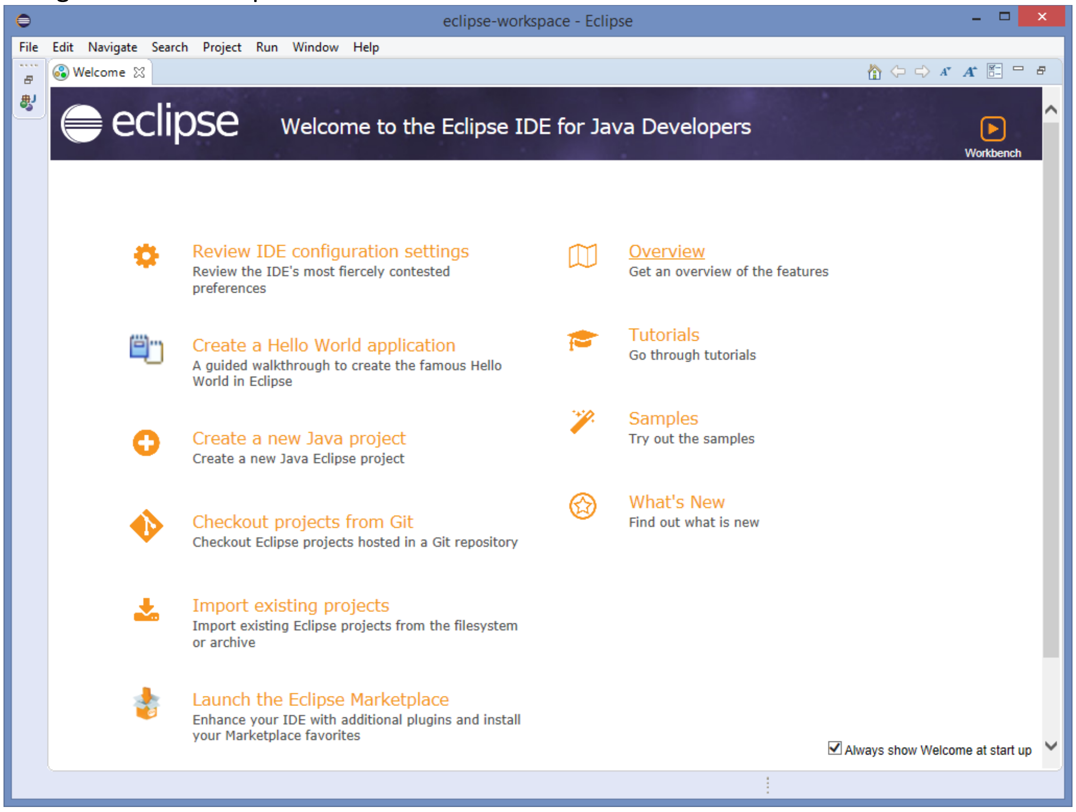
Select a directory as workspace

Eclipse uses the workspace directory to store its preferences and development artifacts.

Workspace: H:\eclipse-workspace

☑ Use this as the default and do not ask again

2. Eclipse takes a few seconds to start up. You will see Eclipse's 'welcome' screen appear - see below. You can un-tick the “Always show Welcome at start up” box to ensure that you go straight to the workspace in future.


eclipse-workspace - Eclipse
File Edit Navigate Search Project Run Window Help
Welcome
eclipse
Workbench
Always show Welcome at start up

Welcome to the Eclipse IDE for Java Developers

Review IDE configuration settings
Review the IDE's most fiercely contested preferences

Create a Hello World application
A guided walkthrough to create the famous Hello World in Eclipse

Create a new Java project
Create a new Java Eclipse project

Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository

Import existing projects
Import existing Eclipse projects from the filesystem or archive

Launch the Eclipse Marketplace
Enhance your IDE with additional plugins and install your Marketplace favorites

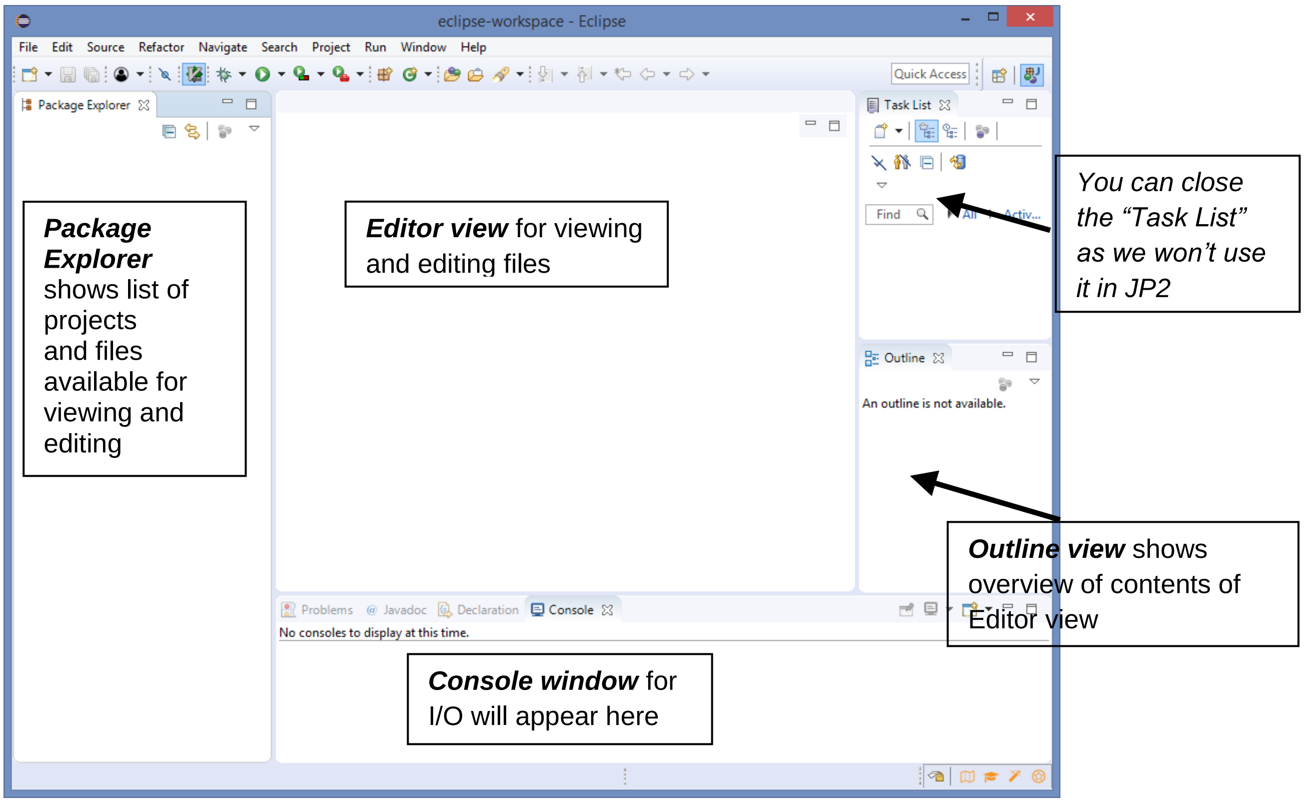
Overview
Get an overview of the features

Tutorials
Go through tutorials

Samples
Try out the samples

What's New
Find out what is new

3. Clicking on the Workbench arrow (arrow in box on top right hand side) will take you to the main IDE. The screenshot below shows the appearance of the default Java perspective when no projects or files are open.


eclipse-workspace - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Package Explorer
Task List
Find
All
Activ...
Outline
Problems
Javadoc
Declaration
Console

An outline is not available.

No consoles to display at this time.

Package Explorer shows list of projects and files available for viewing and editing

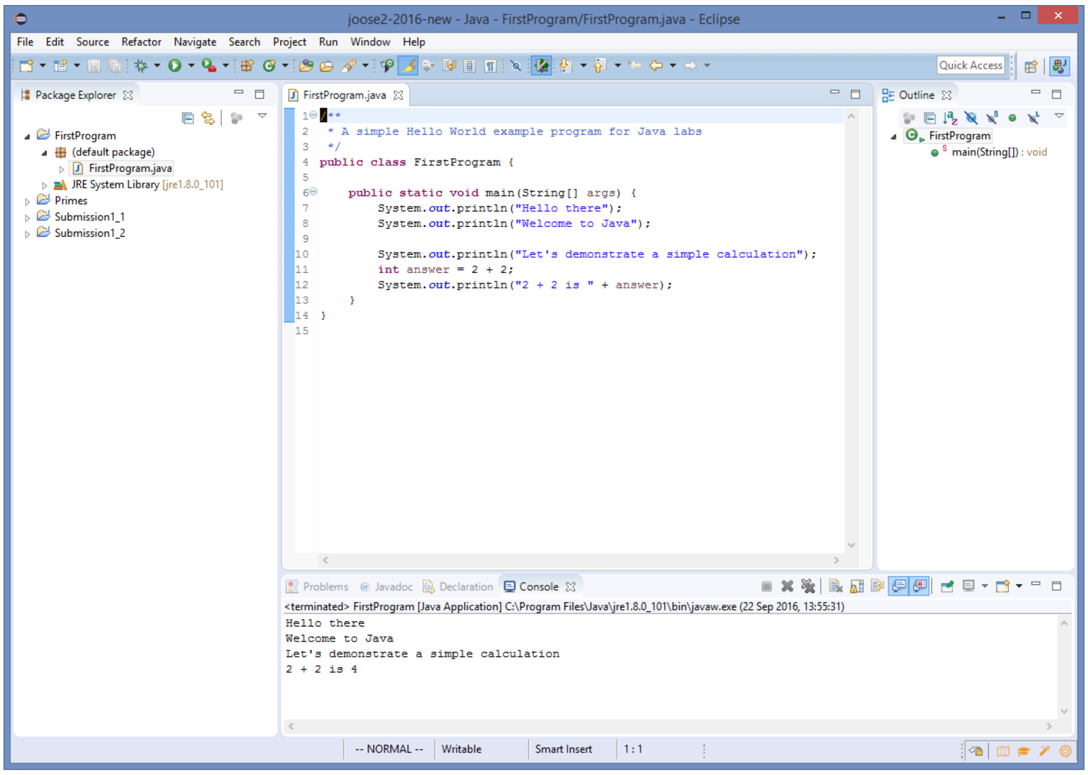
Editor view for viewing and editing files

You can close the “Task List” as we won’t use it in JP2

Outline view shows overview of contents of Editor view

Console window for I/O will appear here

4. 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, the name of the folder necessarily being the same as the name of the project. You will need to import the project(s) into your workspace so that you can run and modify them.
5. In Eclipse, select **File → Import** … to launch the import wizard – this will be used to import the starter projects into your workspace. Then select **General → Existing Projects into Workspace** and click **Next**.
6. Choose **Select archive file** and then browse to the location where you downloaded `Laboratory1.zip`. Select `Laboratory1.zip` and click **Open**.
7. You should now see a set of four projects in the Projects window: **FirstProgram**, **Primes**, **Submission1_1**, and **Submission1_2**. Ensure that the checkboxes beside all of these are selected (e.g., by pressing **Select All** if they are not), and then press **Finish**.
8. In the **Package Explorer** view (on the left of the Eclipse window), you should now see icons representing the four projects. Clicking on the expander icon ▶ beside **FirstProgram** 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.
9. 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, but in general this can be a very useful feature once you get used to it.
10. 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.



```java
/**
 * A simple Hello World example program for Java labs
 */
public class FirstProgram {

    public static void main(String[] args) {
        System.out.println("Hello there");
        System.out.println("Welcome to Java");

        System.out.println("Let's demonstrate a simple calculation");
        int answer = 2 + 2;
        System.out.println("2 + 2 is " + answer);
    }
}
```

```
<terminated> FirstProgram [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (22 Sep 2016, 13:55:31)
Hello there
Welcome to Java
Let's demonstrate a simple calculation
2 + 2 is 4
```

11. 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 – for example, try deleting punctuation marks like semicolons or curly brackets to see what errors are indicated.
12. Go through the same process with the **Primes** program, 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.
13. 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.
14. 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.

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.

The projects Submission1_1 and Submission1_2 contain the starter code for each project. Each project contains one file, which is the skeleton Java source code for you to fill in the details where indicated. You can run each program by right clicking and choosing **Run as -> Java application** or by using the **Run** pull-down menu.

Submission 1

Your task is to complete a Java program to support a simple number guessing game. The game should proceed as follows:

1. The program selects a random number (the "target") between 1 and 20.
2. The user is prompted to enter a number:
    a. If the user guess is larger than the target, the program prints "Too high!"
    b. If the user guess is smaller than the target, the program prints "Too low!"
    c. If the user guess is correct, the program prints "Just right!"
3. The above process continues until the user guesses the correct number.
4. After the guessing process is complete, the program prints "You took N guesses.", where N is the number of guesses taken to get the correct number.

One possible execution trace of the program is as follows – note that as the target number is chosen randomly, this is just an example. Lines in ***green italics*** represent input provided by the user.

```
Enter your guess:
10
Too low!
Enter your guess:
15
Too high!
Enter your guess:
13
Too high!
Enter your guess:
12
Too high!
Enter your guess:
11
Just right!
You took 5 guesses.
```

The provided starter code in GuessingGame.java initialises a Scanner for reading user input, and also uses the Math.random() function to select a random number between 1 and 20. You should write code where indicated in the file to implement the guessing game described above.

Hint 1: See the **Primes** program for an example use of the **Scanner** class to read input from the user.

Hint 2: We will never test your code with invalid input – e.g., by giving it values that are not integers – so you do not need to worry about dealing with that case.

Submission 2 (more challenging)

For this submission, you should implement an enhanced version of the number guessing game, with the following updates:

- Instead of using the hard-coded value 20 as in Submission 1, you should instead prompt the user to give the maximum value and use that to select the target value.
- You should also prompt the user for the maximum number of permitted guesses, and should only allow the user at most that many guesses to reach the target. If the maximum number of guesses is reached before the target is guessed, you should print "No more guesses." and end the program.
- In addition to the checks on the user guess above, you should also include the following check:
    - If the user guess is less than or equal to zero, or is greater than the maximum value, you should print "Out of range!"

The following are two possible results of playing this game – again, as the number should be randomly chosen, each run can have a different result.

```
Enter maximum number:
20
Enter maximum number of guesses:
5
Enter your guess:
100
Out of range!
Enter your guess:
-5
Out of range!
Enter your guess:
10
Too high!
Enter your guess:
5
Too high!
Enter your guess:
2
Just right!
You took 5 guesses.
```

```
Enter maximum number:
10
Enter maximum number of guesses:
1
Enter your guess:
5
Too high!
Out of guesses. The correct answer is 4
You took 1 guesses.
```

Hint 1: You should adapt your code from Submission 1 – try adding one enhancement at a time and testing each one before adding the next.

Hint 2: Except for the out-of-range checks mentioned above, we will not test your code with any other form of invalid input.

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 JP2 moodle site. Click on Laboratory 1 Submission. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code – probably **H:\eclipse-workspace\Submission1_1\src\** -- and drag only the *single* Java file **GuessingGame.java** into the drag-and-drop area on the moodle submission page. Then do the same for file **GuessingGame2.java** in **H:\eclipse-workspace\Submission1_2\src\**. **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.

Outline Mark Scheme

Your tutor will mark your work and return you a score in the range "Excellent" (*****) to "Very poor" ("). Example scores might be:

5*: you complete both submissions correctly with no bugs

4*: you complete Submission 1 correctly and have made a reasonable attempt at Submission 2

3*: you completed only Submission 1, or you have made a reasonable attempt at both submission even if they do not work correctly

2*: you have made some attempt at both submissions

1*: minimal effort

For this assignment only, we will not deduct for inappropriate code formatting or commenting – however, the tutors may comment on such stylistic issues and will deduct for them in the future.