COM6516 Object oriented programming and software design: Practical session 1

The aims of this practical are:

1. to introduce you to the online resources of the module,
2. to ensure you can edit and compile a simple Java program and
3. to introduce some basic features of the Java language.

You should work through this worksheet on your own.

Locating online resources:

- Open a web browser and go to the Blackboard webpage (this is the main medium for delivering course materials).
- Take a look at the information about the course on Blackboard, and find the items such as this document, zipped program code and the sheffield package.

If you are new to Java, then study the example code and work through the tutorials on the Oracle web pages – http://download.oracle.com/javase/tutorial/java/index.html

During the lab session you may wish to use an IDE such as Netbeans. Alternatively you may consider using jEdit to edit your Java code, and a Command Window to compile.

Part 1: Introduction (this should take about 10 minutes to complete)
This part of the practical will ensure that you can edit, compile and run a simple Java program.

- Copy `HelloWorld.java` to a work folder (for example, you could create a folder called `COM6516/labclasses/practical1/` for storing all the work associated with this module), and use either jEdit or Netbeans to open the file.
- Take a look at `HelloWorld.java` and read the comments. Try compiling it in a command window (`javac HelloWorld.java`), and see if you can make sense of the error message. Modify the file as indicated by the comments, and save it.
- Compile the modified file using the java compiler, then run (`java HelloWorld`).

Part 2: Variable types and simple I/O (this will take you a little longer)
Hopefully you will complete the first part of the practical quickly, and can get on to the more interesting part. Here you will write more complex programs, and implement some basic features of Java.

Java is a strongly typed language, so all variables must be declared to have a name and a type (int, double, char etc.) before they are used. Under some conditions variables of one type are promoted automatically, and this can also be done with a cast.

- Copy the file `TypeCast.java` to a work folder and inspect the source code using an editor. Compile it, and run it. Play with the code and make sure that you understand what it is doing, and how the type promotion works in Java.
- Compile the file `QuadraticSolver.java` and run it.

- The program `QuadraticSolver.java` includes javadoc comments. Run javadoc (type `javadoc QuadraticSolver.java` in the command window, or run `javadoc` from within an IDE) and use a web browser to look at the index.html file that is generated by javadoc.
- This program solves a quadratic equation $ax^2 + bx + c = 0$, where the coefficients *a*, *b*, and *c* are known, giving two solutions for *x*, using the formula $x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$
- The program `QuadraticSolver.java` uses both integer and double types to store *a*, *b*, and *c*, and prints out the solution (*x1* and *x2*) for each case. The coefficients are initially set to be *a=1*, *b=2*, and *c=1*, and then we subtract 0.5 from *a*. Why are the solutions different, and which is correct? (Hint: you may wish to refer to `Typecast.java` for an illustration).
- Remove the subtraction of 0.5 from both the integer and the floating-point value of *a*. Set the value of *b* to 2000000 (that is 2 million) in both integer and floating-point parts of the code. What do you think happens now that makes the solutions differ?
- Remove (or comment out) the code for the incorrect variable type.
- Add some code to the program `QuadraticSolver.java` to insert the solutions *x1* and *x2* back into the quadratic equation: i.e. calculate $a(x1)^2+b(x1)+c$ and $a(x2)^2+b(x2)+c$. These values should be very close to 0 if the solutions *x1* and *x2* are correct.
- The 'sheffield package' provides some classes that will enable you to get information from the keyboard or text files easily. This package is located in a subfolder called `sheffield` containing the `EasyReader` (reads from keyboard or file), `EasyWriter` (writes to screen or a file), and `EasyGraphics` (displays to a graphics window) classes.
- The file `KeyboardInput.java` uses the `EasyReader` class from the 'sheffield package' to read numbers from the keyboard. Note that this code includes a statement right at the start of the program `import sheffield.*;` this instructs the compile to include the classes in the 'sheffield package' in the compiled bytecode.
- Using `KeyboardInput.java` as a template, add code to `QuadraticSolver.java` that enables the user to input values for *a*, *b*, and *c*.
- How could you modify your program to catch incorrect input?

Part 3: I/O from a file

The `EasyReader` class can be used to read data from a file if it is given the file name as an argument – e.g. `EasyReader myFile = new EasyReader("myFile.txt");`

In this task you should write a Java program that simulates the behaviour of a cycle computer, which measures distance travelled, time taken, current speed, average speed, and maximum speed. Most cycle computers compute distance and speed from a sensor that detects the time taken for each rotation of the wheel. To simulate the sensor, we have created a file called `'timings.txt'` which simulates a list of rotation times for a cycle ride.

Your program should therefore read data from the file `'timings.txt'`. This file contains a list of numbers, with one number on each row of the file. The first row contains an integer, which is not part of the data but instead states how many floating point numbers there are to follow. These floating point numbers are the time taken in seconds for each rotation of the bicycle wheel, and the distance travelled during each rotation is then π (3.1415927...) multiplied by the wheel diameter. You can assume that the wheel diameter is 0.665 m.

Your program should first read the data from the file `timings.txt` into an array and then calculate each of the following:

(a) The instantaneous speed for each rotation of the wheel in km/h.
(b) The maximum speed over the whole journey, in km/h.
(c) The total distance travelled during the journey, in km.
(d) The total time taken for the journey, in minutes.

How to proceed

The following is just one way that you might choose to develop your program `CycleComputer.java`. It suggests a cautious and controlled way to develop a small program such as the one in this exercise.

- Assume that the data file exists and that the data in it are well formed. Thus there will be no need for explicit error checking in the program**.**
- Write a simple Java program that reads data from the file, one item at a time, and displays each item on the screen as it is read, using `System.out.println()`. You need a loop structure for this. Since the first line of the file states how many items are to follow, a '`for`' loop can be used. You can read an integer from an `EasyReader` object using the `readInt()` method, and the subsequent numbers using the `readDouble()` method. Visually check that the display of data matches the data in the text file, and make sure that you have imported `sheffield.*`.
- Next, modify your code to declare an array of double after you have read the first line of the file. If you do not know how to use arrays in Java then check the Oracle tutorials. The first line of the file will give the size of the array to be declared (but remember that array indexing starts at 0 in Java). Using the `for` loop, read each subsequent data item into the array. Then write a second for loop that displays each item in the array. Visually check that the display of data matches the data in the text file. Now you have successfully read the data into an array, you can complete the exercise.
- Use a `for` loop (or `for-each` loop) to calculate the instantaneous speed, which is the wheel circumference (π multiplied by the wheel diameter of 0.665 m) divided by the time taken for the bicycle wheel to rotate, and store these values in an array. Display this information, remembering to convert from m/s to km/h.
- Use the array of instantaneous speeds to calculate the maximum speed over the whole journey, and display this information.
- Use the timings array to calculate the total distance travelled, and the total time taken.
- Make sure you have formatted your code so it is readable, and tell the user what you are displaying using a string message when you display the results.
- Now you should look again at your solution. Is it efficient? Is it extensible? Have you used *methods* to calculate (a) to (d) above? If you are familiar with object oriented programming, could you create a `Trip` class to model each trip?