MACQUARIE UNIVERSITY

Faculty of Science

COMP125 Fundamentals of Computer Science
Workshop Week 2

Learning outcomes

Following are this week's learning outcomes,

a. Perform problem-solving tasks

b. Create a Java project from scratch

c. Apply refactoring to improve a design and program.

d. Use basic debugging features of Eclipse

> Download `Workshop week 2 files` from iLearn and import the project contained inside (`week2tutorial`) in Eclipse. The process of importing Java projects from archive files is explained in week 1 tutorial worksheet. A video describing the process is also available at http://comp.mq.edu.au/units/comp125/video/import-eclipse.mp4

1. **Storm trooper**

   During a storm, you always see the lightning first and then you hear the sound of thunder. This is because sound travels much slower than light. Namely, sound travels approximately at 340 m/s as compared to 300,000 km/s for light. Hint: use the formula $speed = distance/time$.

   In project `week2tutorial`, there is a class file `Task1`. Based on this observation, complete `Task1`, which calculates and displays the distance to a storm in meters, given the time interval in seconds between the lightning flash and the sound of thunder. You can test the correctness of the program by assigning various values to the variable holding the time interval.

   > **Solution:** Refer to `Task1.java` in project `week2tutorialSolution`.

2. **Largest of three values**

   Design (no coding component for this task) a solution that computes the largest of three given integers. Proficient students might like to take this further: how could we find the largest of n elements stored in an array, where n is any positive integer?

Solution: First solution: take three parameters (a,b,c). If $a \geq b, agec$, answer is a. Otherwise, it's a decision between b,c. So check if $b \geq c$. If so, answer is b, otherwise answer is c.

Second solution: design a solution to determine higher of two values (m,n). Here, if $m \geq n$, answer is m, otherwise answer is n. Now to determine highest between three values (a,b,c), the answer is $higher(higher(a,b),c)$.

3. **String theory**

In lectures last week we introduced the `String` type (class) and some of its methods. Briefly review this material. In project `week2tutorial`, there is a class file `Task3`. Please open this file. Notice that there are three methods, a `main` method and two helping methods, `countOccurrences` and `isPalindrome`.

a. (i) Complete the method `countOccurrences` which, when passed a `String aString` and a letter of the English alphabet `aLetter`, returns the number of occurrences of `aLetter` in `aString`.

(ii) The `main` method has a piece of code that inputs a `String` and a `char` from the user. Write a few lines of code that displays the number of occurrences of the `char` input in the `String` input.

b. Complete the method `isPalindrome` which, when passed a `String aString`, returns `true` if `aString` is a *palindrome*. A palindrome is word that when reversed remains unchanged. Some examples of words that are palindrome are "madam", "malayalam", "detartrated". Words that are **not** palindromes are "hello" and "Tenet" (the second being a non-palindrome because the first 'T' is in uppercase and the second 't' is in lowercase).

You can approach this question in any way you feel suitable. For example, you could try to solve it from first principles. Or you could cut-and-paste some code from the StringReversal program available in Week 1, then add one or two lines as needed to that code.

Solution: Refer to `Task3.java` in project `week2tutorialSolution`.

4. **Creating Java project**

Follow the following instructions to create a new Java project.

a. Click on File –> New –> Java Project.

b. Give the project a name. By convention, Java project names are camel-cased, starting with a lowercase letter. For example, `myVeryOwnJavaProject`. For this example, name the project `task4project`.

c. Press ENTER, or click on `Finish`.

d. Double-click on the project. Then right-click on `src` and choose New –> Package.

e. Name the package `comp125`, which is the default package name for all projects in this unit. Press ENTER, or click on `Finish`.

f. Right-click on `comp125` and choose New –> Class. By convention, Java class names are camel-cased, starting with an uppercase letter. For example, `MyClass`. For this example, name the class `Task4`.

g. **Check the button that states `public static void main(String[] args)`**

h. Now you are ready to add code inside the `main` method, and add more methods.

i. Methods that are called by the `main` method, must be prefixed with keyword `static`. For example, if you have a method that returns the square of a `double` passed to it, and is called by `main`, it will be defined as,

```
1        public static double square(double num) {
2                return num*num;
3        }
```

Also, methods that are called by other `static` methods, must be prefixed with static. This is not true for all methods and will be made clearer in the next few weeks.

j. In the `main` method, write a piece of code that computes the sum of the first hundred odd integers, and displays it in the console. Console output is given using
`System.out.println(stuff to output goes here)`

5. **Use basic debugging features of Eclipse**

In project `week2tutorial`, there is a class file `Task5Buggy`. It is a code that computes factorial of 5 (= 1 * 2 * 3 * 4 * 5) but contains two bugs. NOTE: The following process is also illustrated in the *Debugging example* video uploaded on iLearn.

a. Go to statement
`int n = 5;`
and press `Ctrl+Shift+B`. You should see a blue circle appear to the left of the line concerned. You can add/ remove breakpoints by double-clicking where the blue circle has appeared. Repeat the same process for the three lines after that.

b. Click on the green *Bug* button, to the left of the *Run* button. Eclipse should change the view significantly and enter a *Debug mode*.

c. Click on the *Slow motion* button, that stands for emphResume. It executes your program till the next breakpoint is encountered.

d. You can track the variables in the *Variables* window. You might have to click on *Variables* to switch view from *Console* to *Variables*.

e. To exit debug perspective and go back to Java perspective, click on the *Stop* button (the red square button) and then the *Java* button in the top right corner of Eclipse.

f. Report your observations and suggest the fixes.

6. **Refactoring code**

In project `week2tutorial`, there is a class file `Task6Refactoring`. It contains a piece of code that determines if a number has integer square roots. You will see that the code is highly repititive. Your task is to identify potential to move sections of the code to a method, and replace the repetitive code with method calls.

Solution: Refer to `Task6Refactoring.java` in project `week2tutorialSolution`.

7. **Problem-solving (to be submitted)**

The following question is a twist on the age-old *primality* problem. *Primality* is defined on integers, and an integer is prime if it is,

a. More than 1.

b. Divisible only by 1 and itself

The first few prime numbers are: 2, 3, 5, 7, 11, 13, ...

You are required to complete the method `isAlmostPrime` that when passed a positive integer, returns `true` if it has exactly three positive divisors, and `false` otherwise. That is, the method returns `true`, if besides 1 and itself, the integer has exactly one other positive divisor. For example, the method should return `true` for parameter 4, since it has exactly three divisors: 1, 2, and 4. The method should return `false` for parameter 6, since it has four divisors: 1, 2, 3, and 6.

The method should return `false` for any value less than 2 passed to it.

Solution: Refer to `Task7.java` in project `week2tutorialSolution`.

Submit only the file `Task7.java`

10% penalty for submitting incorrect file

You should submit your solution for Question 7 by submitting the file `Task7.java`. Your submission will be graded and is worth 1 mark towards your final grade for COMP125.
Due date: 5pm, Sunday 17th August

Additional tasks (for anyone who has a little spare time)

- Write a method that when passed a `String`, returns the most frequently occuring `char` in that `String`. For example, when passed "abysmal", the method returns 'a'. In case of a tie, return the `char` that occurs first. For example, when passed 'surreal", the method returns 'r'.

  Solution: Refer to `Task3.java` in project `week2tutorialSolution`.

- Write a method that when passed a `String`, returns a `String` containing the most frequently occuring characters in that `String`, in the order of their occurrence. For example, when passed "abysmal", the method returns "a", and when passed "fantastic", the method returns "at".

  Solution: Refer to `Task3.java` in project `week2tutorialSolution`.