MACQUARIE University

Department of Computing

COMP125 Fundamentals of Computer Science
Workshop Week 2

Learning outcomes

Following are this week's learning outcomes,

a. Perform problem-solving tasks

b. Identify and eliminate bugs from an incorrect implementation

c. Write methods that deal with arrays.

> Download `Workshop week 2 files` from iLearn and import the project contained inside (`workshop02template`) in Eclipse. The process of importing Java projects from archive files is explained in week 1 tutorial worksheet.

IMPORTANT: Work in pairs for entire duration of workshop

1. **Tracing**

   On a piece of paper, trace the following codes. Indicate clearly, the values of all variables when the code finishes execution.

   (a)

```
int a = 160, b = 70;
while(a%b != 0) {
        int temp = b;
        b = a%b;
        a = temp;
}
//what are the values of a and b?
```

   Solution: a = 20 b = 10

   (b)

```
int result = 0;
for(int i=2; i<=10; i+=2) {
        result = result + i;
}
```

Solution: result = 2+4+6+8+10 = 30

(c)

```
int result = 0;
for(int i=1; i<=10; i++) {
        if(i%2 == 0) {
                result = result + i;
        }
}
```

Solution: Same output. result = 2+4+6+8+10 = 30

(d)

```
//Function definition
int collatz(int n) {
        int counter = 0;
        while(n != 1) {
                if(n%2 == 0) {
                        n = n/2;
                }
                else {
                        n = 3*n+1;
                }
                counter++; //counter = counter + 1
        }
        return counter;
}
```

```
//Function call:
int result = collatz(10);
```

Solution: n = 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1 counter = 6

(e)

```
//Function definition
int highest(int[] a) {
        int max = a[0];
        for(int i=1; i < a.length; i++) {
                if(a[i] > max) {
                        max = a[i];
                }
        }
```

```
        return max;
```

```
//Function call:
int[] arr = {6, 3, 2, 8, 7, 12, 5, 3, 9};
int largest = highest(arr);
```

Solution: highest = 6 -> 8 -> 12

2. The method `allEven` **attempts to** return `true` if the array passed contains only even numbers, and `false` otherwise. However it contains a few logical and syntactical bugs. Identify and explain these bugs. Write the corrected method.

```
public static boolean allEven(int[] a) {
        for(int i=1, i <= a.length, i++) {
                if(a[i]%2 != 0) {
                        return false;
                }
                else {
                        return true;
                }
}
```

Solution:

```
public static boolean allEven(int[] a) {
        for(int i=1, i <= a.length, i++) {
                if(a[i]%2 != 0) {
                        return false;
                }
                else {
                        return true;
                }
        closing bracket missing
        return true missing after the loop
}
```

```
public static boolean allEven(int[] a) {
        for(int i=1; i < a.length; i++) {
                if(a[i]%2 != 0) {
                        return false;
                }
        }
        return true;
}
```

3. **Demonstration of Eclipse Debugging capabilities**.

   Please take out your phones (Yes, we are actually asking you to do that!) and watch a (not so) cool video we uploaded here that demonstrates debugging! If you have headphones - awesome,

otherwise just watch it with a reasonable volume. Five minutes of your life we guarantee you won't get back.

4. **Bugs buggy**

   In class `Buggy`, code in `main` tries to compute the sum of all items of the array `{10, 70, 20, 90}`, but it has some bugs, resulting in the value being computed as 7 instead of 190. Use the debugger as described in the video to identify and eliminate these bugs.

   a. Place breakpoints on lines you'd like the execution to halt at.

   b. Run the program in debug mode.

   c. Trace the variables. This should help you identify where the problem lies.

   The value displayed when the bugs are eliminated should be 190.

   Solution: The variable `factorial` should be initialised to 1 (instead of 0), and the loop expression should be $i \leq n$ (instead of $i < n$).

5. **Methods (Functions) operating on arrays**

   Complete the following method in class `MainTask`:

   `public static boolean onlyNegatives(double[])`

   Notice we don't write the name of the parameter while communicating method headers, as we don't care about the name, only the type. It's only when we actually write the code, we give it a name (in this case, the name given to the array is `arr`).

   A sample set of method calls is supplied in `main` along with expected outcome.

6. **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 `week2project`.

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

   d. Right-click on `src` 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 `Week2`.

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

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

   g. Methods that are directly 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,

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

   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.

   h. 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>);
```

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

7. Write a method that when passed a `String`, returns the most frequently occurring `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:** In `AdditionalTasks.java`
> (inside the project imported from `workshop02solutionCode.zip`)

8. Write a method that when passed a `String`, returns a `String` containing the most frequently occurring 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:** In `AdditionalTasks.java`
> (inside the project imported from `workshop02solutionCode.zip`)

9. Write a method that when passed a floating-point array (`double[]`), returns the number of unique items, that is, the number of items that occur exactly once in the array.

> **Solution:** In `AdditionalTasks.java`
> (inside the project imported from `workshop02solutionCode.zip`)

10. Write a method that when passed two integer arrays, returns `true` if they are identical, `false` otherwise.