Department of Computing

COMP125 Fundamentals of Computer Science
Workshop - Workshop - First Steps

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.

1. **Debugging**

   Your tutor will demonstrate the process of debugging a method with the example of method `sumEven` in class `DebuggingDemo`. You should then debug the method `isAscending` in the same class.

   Solution: Refer to code in `workshop02solution.zip`

2. **Bugs buggy**

   In class `Buggy`, the code in `main` attempts to add all items of the array `a`. However, the code contains a few bugs. Identify and correct them using the debugger.

   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: Three bugs:

   a. result should be initialized to 0, not 1

   b. start loop counter i from 0, not 1

   c. add a[i] every time, not i

3. The method `allEven` **attempts to** return `true` if the array passed contains only even numbers, and `false` otherwise. However it contains a bug. Identify and explain that bug. Write the corrected method.

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

Solution: Method returns `true` as soon as one even number is found. This (returning `true`) should be done after all numbers are checked and none of them is odd.

```java
public static boolean allEven(int[] a) {
        for(int i=0; i < a.length; i++) {
                if(a[i]%2 != 0) { //an odd number
                        return false;
                }
        }
        //if control reaches here, it means
        //no odd numbers were found
        return true;
}
```

4. The following method attempts to return `true` if the two arrays passed are identical (same items in the same order), and `false` otherwise. However, it has a bug. Trace the following method for input arrays {12, 6, 15} and {12, 6, 15, 8}, identify the bug and fix the code.

```java
public static boolean identical(int[] a, int[] b) {
        for(int i=0; i < a.length; i++) {
                if(a[i] != b[i]) {
                        return false;
                }
        }
        return true;
}
```

Solution: Methods doesn't check if the size of the two arrays is same or not. Debugged version:

```java
public static boolean identical(int[] a, int[] b) {
        if(a.length != b.length) {
                return false;
        }
        for(int i=0; i < a.length; i++) {
                if(a[i] != b[i]) {
                        return false;
                }
        }
        return true;
}
```

5. The following method attempts to return the sum of all odd numbers in the array. However, it has a bug (and a tricky one). Trace the method for input array {5, 8, 4, 0, 7, -3, 6} to identify the bug and get rid of it.

```java
public static int sumOdds(int[] a) {
        int result = 0;
        for(int i=0; i < a.length; i++) {
                if(a[i]%2 == 1) {
                        result = result + a[i];
                }
        }
        return result;
}
```

Solution: Negative odd numbers when divided by 2 leave remainder -1 (and not 1). Debugged version:

```java
public static int sumOdds(int[] a) {
        int result = 0;
        for(int i=0; i < a.length; i++) {
                if(a%2 != 0) {
                        result = result + a[i];
                }
        }
        return result;
}
```

6. Complete the method `onlyNegatives` in class `MainTask`. For example,

   `onlyNegatives(new int[]{-10, -20, -17, -1})` –> `true`

   `onlyNegatives(new int[]{-10, -20, 0, -1})` –> `false`

   `onlyNegatives(new int[]{-10, -20, -17, 1})` –> `false`

   `onlyNegatives(new int[]{})` –> `true` (all items in the array **ARE** negative)

   `onlyNegatives(new int[]{-10})` –> `true`

   `onlyNegatives(new int[]{3, -20, -17, -1, -9, -14})` –> `false`

Solution:

```java
/**
 *
 * @param arr
 * @return true if all the items of array arr are negative
 * (less than 0), false otherwise.
 * note, return true if array arr has no items
 * (it's empty)
 */
public static boolean onlyNegatives(double[] arr) {
        for(int i=0; i < arr.length; i++) {
                if(arr[i] >= 0) { //NOT negative
                        return false; //to be completed
                }
        }//end loop
        //if didn't return false during the loop
        //it means no non-negatives
        //so all negatives
        //so...
        return true;
}
```

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

7. **String theory**

   For this question, you will need the following two methods that operate on a String object (assuming the object name is `str`):

   a. `str.length()`: returns the number of characters in the String.

   b. `str.charAt(int)`: returns the character at passed index, provided the index is valid (between 0 and `str.length() - 1`, including `str.length() - 1`).

   Example:

```java
String s = "hello";
System.out.println(s.length()); //displays 5
System.out.println(s.charAt(0)); //displays 'h'
System.out.println(s.length(4)); //displays 'o'
//System.out.println(s.charAt(-1)); //Boo...
//System.out.println(s.charAt(5)); //Boo...
```

   In project `workshop02template`, there is a class file `StringTheory`. Please open this file. Notice that there are two methods, a `main` method and a helping method, `countOccurrences`.

   Complete the method `countOccurrences` which, when passed a `String str` and a character `ch`, returns the number of occurrences of `ch` in `str`.

   Solution:

```java
public static int countOccurrences(String str, char ch) {
        int count = 0;
        for(int i=0; i<str.length(); i++) //for each character
                if(str(i) == ch) //found a match
                        count++;

        return count;
}
```

8. 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` (which is inside the project imported from `workshop02solution.z`

9. 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` (which is inside the project imported from `workshop02solution.z`

10. 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` (which is inside the project imported from `workshop02solution.z`