# EECS2030Z Test 2

# Version A

## **GETTING STARTED**

- 1. Start eclipse
- 2. Import the test project by doing the following:
  - 1. Under the **File** menu choose **Import...**
  - 2. Under General choose Existing Projects into Workspace and press Next
  - 3. Click the **Select archive file** radio button, and click the **Browse...** button.
  - 4. Select the file **test2A.zip** and click **OK** 
    - If you do not see the file named **test2A.zip** in your directory, then open a terminal and copy and paste the following command:

cp /eecs/dept/www/course/2030/labtest/test2A.zip .

and re-import the project.

- 5. Click Finish.
- 3. All of the files you need for this test should now appear in eclipse.
- 4. Open a terminal. You will use this terminal to submit your work.
- 5. Copy and paste the command cd workspace/Test2A/src/test2 into the terminal and press enter.

## Question 1 (20 marks total)

#### **SOLUTION**

Implement the utility class described by this API. You do not have to include javadoc comments.

submit 2030 test2A Utility2A.java

## Question 2 (10 marks total)

## **SOLUTION**

Implement the constructors for the class described by this API. Use constructor chaining where possible when implementing your constructors. You do not have to include javadoc comments in your code, but see Question 3C below.

submit 2030 test2A Counter.java

# Question 3 (20 marks total)

-A

State the definition of a method precondition.

A method precondition is a condition that the client must ensure is true immediately before a method is invoked.

Grading scheme: 3 marks total.

B

Suppose that the method name alternatingsum from Question 1 was implemented like so:

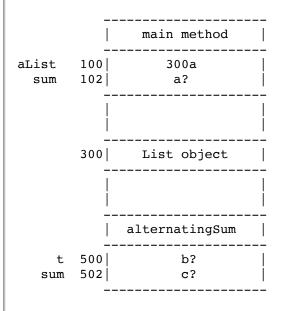
```
public static int alternatingSum(List<Integer> t) {
   int sum;
   /* code not shown here that assigns the correct value to sum */
   return sum;
}
```

Suppose a client writes a main method that includes the following two lines of Java code:

```
List<Integer> aList = Arrays.asList(8, 9, 10, 11);
int sum = Utility2A.alternatingSum(aList);
```

The first line of code creates the list [8, 9, 10, 11], and the second line of code calls the alternatingSum method from Question 1.

The memory diagram illustrating the state of memory for the two lines of client code is shown below. What suitable values of a, b, and c would complete the memory diagram?



$$a = -2$$
,  $b = 300a$ ,  $c = -2$ 

a and c are both equal to the value of the alternating sum (8 - 9 + 10 - 11 = -2). b is equal to the address of the list.

Grading scheme: 3 marks total (1 for each of a, b, and c).

#### ·C.

Provide the Javadoc necessary to *exactly* reproduce the API documentation for the constructor Counter(String name, int value) from Question 2.

```
/**
  * Initialize this counter to have the given name and value.
  *
  * @param name the non-null name of this counter
  * @param value the value of this counter
  * @throws IllegalArgumentException if value is less than zero
  */
```

Grading scheme: 4 marks total (1 for method description, 1 for the first parameter description, 1 for the second parameter description, 1 for the exception). The answer must appear exactly as shown above (missing or extra white space is acceptable, and the leading \* are not required.).

### D. (6 marks)

Provide 3 test cases for the method Utility2A.totalArea. Make sure that each test case tests a different feature of the method (i.e., don't provide 3 test cases that all check if the correct area is returned). For each test case, provide a one sentence explanation of what the test case is testing.

```
There are many possible test cases; below are 4 examples:
```

```
: [1, 5, 10]
widths
heights
                    : [2, 3, 4]
expected return value: 57
explanation
                    : tests typical widths and heights
widths
                    : [1, 5, -10]
heights
                    : [2, 3, 4]
expected return value: 17
explanation
                    : tests a negative width
widths
                    : [1, 5, 10]
heights
                   : [2, -3, 4]
expected return value: 17
explanation
                    : tests a negative height
widths
                    : [1]
heights
                    : [2, 3, 4]
expected return value: IllegalArgumentException
explanation
                    : tests that an exception is thrown if list sizes are different
```

Grading scheme: 6 marks total.

2 marks for each test case (1 mark for providing appropriate inputs, 0.5 mark for providing the expected return value or result, 0.5 mark for the explanation).

Consider the following class and its implementation of equals(Object): public class Bool { private boolean value; private String name; @Override public boolean equals(Object obj) { if (this == obj) { return true; if (obj == null) { return false; if (this.getClass() != obj.getClass()) { return false; Bool other = (Bool) obj; if (!this.name.equals(other.name)) { return false; if (this.value && other.value == false) { return false; return true; }

Explain whether or not the implementation shown above satisfies the equals contract.

The implementation does not satisfy the equals contract because x.equals(x) == true is not always true! This occurs when this.value == false because x.equals(x) returns in the following if statement:

```
if (this.value && other.value == false) {
  return false;
}
```

Grading scheme: 4 marks total.

2 marks for stating x.equals(x) == true is not always true. 2 marks for the explanation.

submit 2030 test2A answers.txt