

ITSC 1213 - Testing

Introduction

You've probably heard this sentence before "You have a bug in your program", it's very likely that it is telling you to worry about roaches infesting your computer but what we really mean is that there are errors in the program.

This lab is adapted from http://www.cs.sjsu.edu/web_mater/cs46a/cs46alab/lab6/tutorial.html

Concepts covered in this lab:

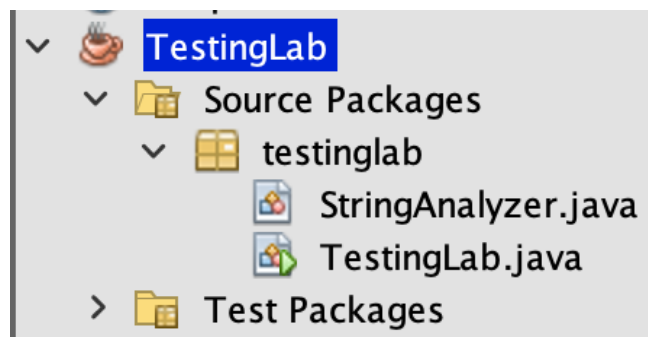
- Reading the Stack Trace
- Unit Testing
- Debugging

Required files or links

- Code snippets linked in document.

Part A: Reading the stack trace and unit testing

- 1) Create a new NetBeans project called TestingLab with a main class.
- 2) Create a new Java class called `StringAnalyzer`. Make sure it is in the same package as the main class. Copy the `StringAnalyzer` class from [this code snippet](#). Make sure to include the correct package declaration when adding this file to a dedicated package in your project.



- 3) In the class with the main method, add a method called `validate` as follows

```

private static String validate(char expected, char result) {
    if (result != expected) {
        return ("The result " + result + " does not match expected: "
            + expected + " ---->> Failed");
    } else {
        return ("The result " + result + " match expected: "
            + expected + " ---->> OK");
    }
}

```

- 4) Now, in the main class add the following statements to the main method to call the test method that we just added

```

public static void main(String[] args) {
    StringAnalyzer wa = new StringAnalyzer();

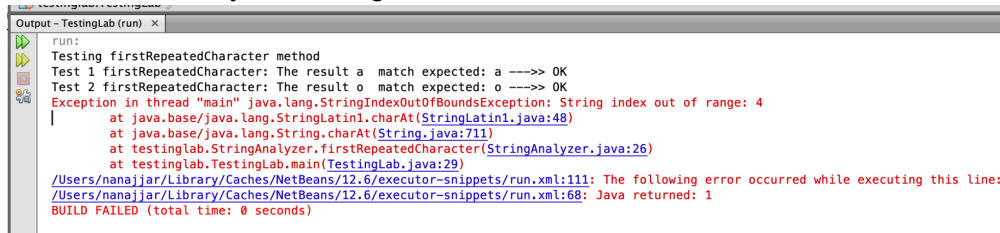
    System.out.println("Testing firstRepeatedCharacter method");
    String s = "aardvark";
    char result = wa.firstRepeatedCharacter(s);
    System.out.println("Test 1 firstRepeatedCharacter: " + validate('a', result));

    s = "roommate";
    result = wa.firstRepeatedCharacter(s);
    System.out.println("Test 2 firstRepeatedCharacter: " + validate('o', result));

    s = "mate";
    result = wa.firstRepeatedCharacter(s);
    System.out.println("Test 3 firstRepeatedCharacter: " + validate('o', result));
    System.out.println("-----");
}

```

- 5) Examine the contents of the code and predict the output of running the main class.
- 6) Now run the main class, what output do you get?
- 7) You can see that the program runs but throws a runtime exception. Let's take a closer look at this error by examining the stack trace.



```

Output - TestingLab (run) x
Run:
Testing firstRepeatedCharacter method
Test 1 firstRepeatedCharacter: The result a match expected: a ---->> OK
Test 2 firstRepeatedCharacter: The result o match expected: o ---->> OK
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: 4
    at java.base/java.lang.StringLatin1.charAt(StringLatin1.java:48)
    at java.base/java.lang.String.charAt(String.java:711)
    at testinglab.StringAnalyzer.firstRepeatedCharacter(StringAnalyzer.java:26)
    at testinglab.TestingLab.main(TestingLab.java:29)
/Users/nanajjar/Library/Caches/NetBeans/12.6/executor-snippets/run.xml:111: The following error occurred while executing this line:
/Users/nanajjar/Library/Caches/NetBeans/12.6/executor-snippets/run.xml:68: Java returned: 1
BUILD FAILED (total time: 0 seconds)

```

- 8) The first complaint is about the method `charAt` of the `java.lang.String` class. That's a Java library class. It is extremely unlikely that a library class has a bug. It is far more likely that the problem is caused by how we are using it in our program. In this case we're calling one of its methods (`charAt`) with bad parameters.
- 9) The next complaint is about a line in our code of the `firstRepeatedCharacter` method of the `StringAnalyzer` class.

10) Now let's look at the line of code this complaint is referencing

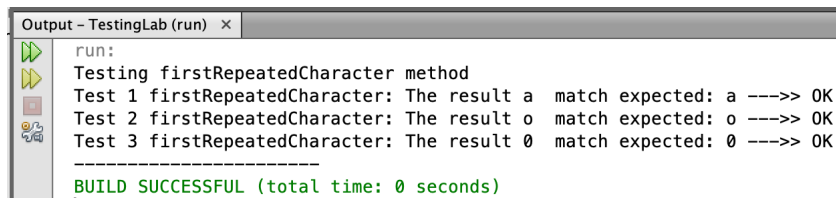
```
if (ch == aString.charAt(i + 1))
```

11) In theory, there are two different exceptions that can be thrown in this call. What are they? If you're not sure, take a look at the JavaDoc for the [charAt method](#) of the `String` class.

12) Since we know that `aString` is not null in this case then the error must be the index value we are using as the method argument.

13) Think of the nature of the bug and how to fix it.

14) Now fix this bug and run the program again. What output did you get after you fixed the bug?



```
Output - TestingLab (run) x
run:
Testing firstRepeatedCharacter method
Test 1 firstRepeatedCharacter: The result a match expected: a --->> OK
Test 2 firstRepeatedCharacter: The result o match expected: o --->> OK
Test 3 firstRepeatedCharacter: The result 0 match expected: 0 --->> OK
-----
BUILD SUCCESSFUL (total time: 0 seconds)
```

15) When your program works and you are satisfied with the result, show your work to the instructional team to get checked off and proceed to Part B.

Part B: Validating more methods

In this part we want to test the `firstMultipleCharacter` method. Notice that this method also returns a `char` type similar to the `firstRepeatedCharacter` method. So we can utilize the same validate method to check if an actual return value matches the expected value when the method is called.

- 1) Your task for this part is to validate the `firstMultipleCharacter` method. You should think of at least 4 different `String` values to that cover the following conditions:
 - A string with no multiple characters
 - A string with multiple characters appearing at the start of the string
 - A string with multiple characters appearing somewhere in the middle of the string
 - A string with multiple characters appearing at the end of the string

In the main method add statements to test the `firstMultipleCharacter` method under the 4 conditions listed above. Your code should utilize the validate method. This is similar to how it is done in the code we gave you for Part A to test the `firstRepeatedCharacter`.

- 2) When your program works and you are satisfied with the result, show your work to the instructional team to get checked off and proceed to Part B.

Part C: Using the debugger to find error source and more testing

Recall that a *debugger* is a tool that lets you execute a program in slow motion. You can observe which statements are executed and you can peek inside variables. In this lab, we will use the debugger that is a part of NetBeans to pinpoint the cause of a logical error that exists in a program.

- 1) So far we have tested two methods of the `StringAnalyzer` class (`firstRepeatedCharacter`, `firstMultipleCharacter`). This leaves the `countRepeatedCharacters` methods to test.
The `countRepeatedCharacters` returns the number of repeated character groups as an integer (`int`) value. In order to test
In order to test if the method works as expected (i.e., the logic is correct) we need to come up with a few Strings and use them to check if the method does the counting correctly. Again, this is similar to what we have done in Parts A and B. The only difference is that we need a validate method that compares the value a method returns to the expected value.

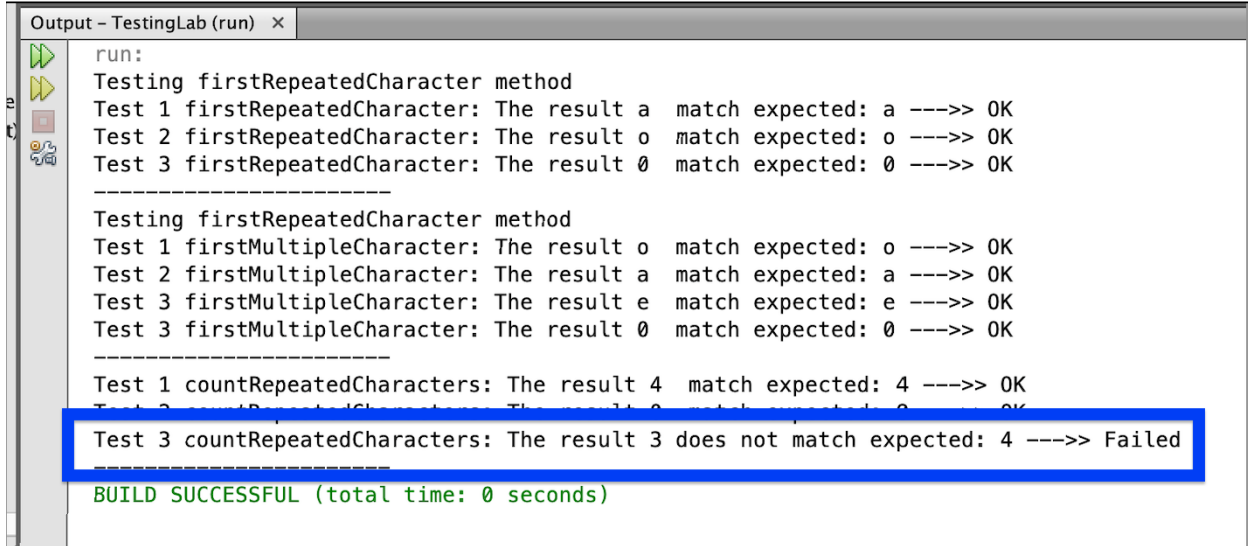
- 2) In the main class add a second (overloaded) method called `validate` as follows

```
private static String validate(int expected, int result) {  
    if (result != expected) {  
        return ("The result " + result + " does not match expected: "  
                + expected + " ----> Failed");  
    } else {  
        return ("The result " + result + " match expected: "  
                + expected + " ----> OK");  
    }  
}
```

- 3) Now let's test the `countRepeatedCharacters` using the Strings "test", "mississippiii" and "aabbcdaaaabbb". These Strings are the test cases for unit testing our method. In the main method add the statements to make these test calls. Your code might look like this

```
System.out.println("-----");  
  
s = "mississippiii";  
int intResult = wa.countRepeatedCharacters(s);  
System.out.println("Test 1 countRepeatedCharacters: " + validate(4, intResult));  
  
s = "test";  
intResult = wa.countRepeatedCharacters(s);  
System.out.println("Test 2 countRepeatedCharacters: " + validate(0, intResult));  
  
s = "aabbcdaaaabbb";  
intResult = wa.countRepeatedCharacters(s);  
System.out.println("Test 3 countRepeatedCharacters: " + validate(4, intResult));  
  
System.out.println("-----");
```

- 4) Run your test file and take a closer look at the output. Do you see anything wrong with the result?

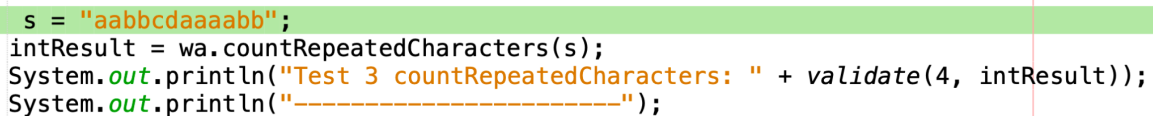


```
run:
Testing firstRepeatedCharacter method
Test 1 firstRepeatedCharacter: The result a match expected: a ---> OK
Test 2 firstRepeatedCharacter: The result o match expected: o ---> OK
Test 3 firstRepeatedCharacter: The result 0 match expected: 0 ---> OK
-----
Testing firstRepeatedCharacter method
Test 1 firstMultipleCharacter: The result o match expected: o ---> OK
Test 2 firstMultipleCharacter: The result a match expected: a ---> OK
Test 3 firstMultipleCharacter: The result e match expected: e ---> OK
Test 3 firstMultipleCharacter: The result 0 match expected: 0 ---> OK
-----
Test 1 countRepeatedCharacters: The result 4 match expected: 4 ---> OK
Test 2 countRepeatedCharacters: The result 4 match expected: 4 ---> OK
Test 3 countRepeatedCharacters: The result 3 does not match expected: 4 ---> Failed
-----
BUILD SUCCESSFUL (total time: 0 seconds)
```

You should see that the `countRepeatedCharacters` method does the right thing for the first two test cases, but it mysteriously fails on the string "aabbcdaaaabbb". It should report 4 repetitions, but it only reports 3.

We know that the first two calls to the `countRepeatedCharacters` method give the correct results. It won't be too interesting to debug them. Instead, let's go directly to the third call. We'll set a *breakpoint* at that line, so that the debugger stops as soon as it reaches the line.

- 5) Move the cursor to the third call to test. Then right click and select **Toggle Line Breakpoint** from the menu. You will see a small red square to the left of the line, indicating the breakpoint.
- 6) Now run the program in debug mode to step through it. This can be done by right clicking the file in the editor window or through the **Debug** main menu item. The debugger now runs your program. When it hits the first breakpoint, you will see that the execution stopped and the line with the breakpoint is highlighted with a green color:



```
s = "aabbcdaaaabbb";
intResult = wa.countRepeatedCharacters(s);
System.out.println("Test 3 countRepeatedCharacters: " + validate(4, intResult));
System.out.println("-----");
```

- 7) This call is boring, and we do not want to step into the constructor. We'll use the "Step Over" command instead.
- 8) Execute the "Step Over" command and see what happens. Now you are at the line `intResult = wa.countRepeatedCharacters(s);`
- 9) Now we want to step through the `countRepeatedCharacters` method in slow motion.

Execute "Step Into". Now you are inside the `countRepeatedCharacters` method. You should get to the lines

```
int c = 0;
```

10) Remember, we want to find out why we get the wrong repetition count for the third test case. Should you execute "Step Into" or "Step Over" at this point?

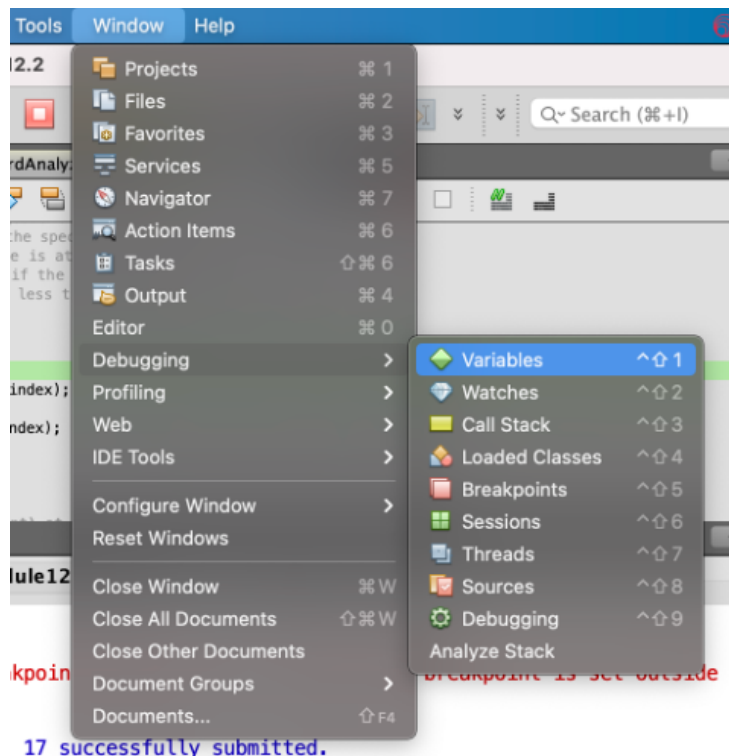
11) Execute "Step Over" a couple of times. You should get to the lines

```
if (aString.charAt(i) == aString.charAt(i + 1))
```

12) Now when you execute "Step Into" again, you will end up in the code for the `String` class.

13) That sometimes happens by accident. In this case, there is no harm done since the `charAt` method is so short. But it can be bothersome to be trapped in a long library method (such as `println`). The remedy is the "Step Out" command. It gets you out of the current method and back to the caller.

14) Look inside the Variables window from main tabs



15) You can see the contents of four variables, `this` (the implicit parameter of the call `wa.countRepeatedCharacters(s)`), `aString`, `c` and `i`.

| Variables | | | |
|-----------|-------------------|----------------|----------------|
| | Name | Type | Value |
| | <Enter new watch> | | |
| > | this | StringAnalyzer | #60 |
| | aString | String | "aabbcdaaaabb" |
| | c | int | 0 |
| | i | int | 1 |

- 16) The triangle next to `this` indicates that you can expand the variable. Click on the triangle next to `this`. What do you see? Why are there no triangles next to `c` and `i`?

| Name | Type | Value |
|----------------------|----------------|----------------|
| <Enter new watch> | | |
| ▼ this | StringAnalyzer | #60 |
| ▼ Static | | |
| \$assertionsDisabled | boolean | true |
| aString | String | "aabbcdaaaabb" |
| c | int | 0 |
| i | int | 1 |

- 17) Now keep executing the Step Over command. Watch what happens to the `c` and `i` values. You'll see `i` increase each time the loop is executed. The value of `c` increases three times. What are the values for `i` at each increase?
- 18) Look at the `aString` value. What is special about the three positions at which `c` increases? Do you see the bug in the code?
- 19) The `countRepeatedCharacters` method looks for character sequences of the form `xyy`, that is, a character followed by the same character and preceded by a different one. That is the *start* of a group. Note that there are two conditions. The condition

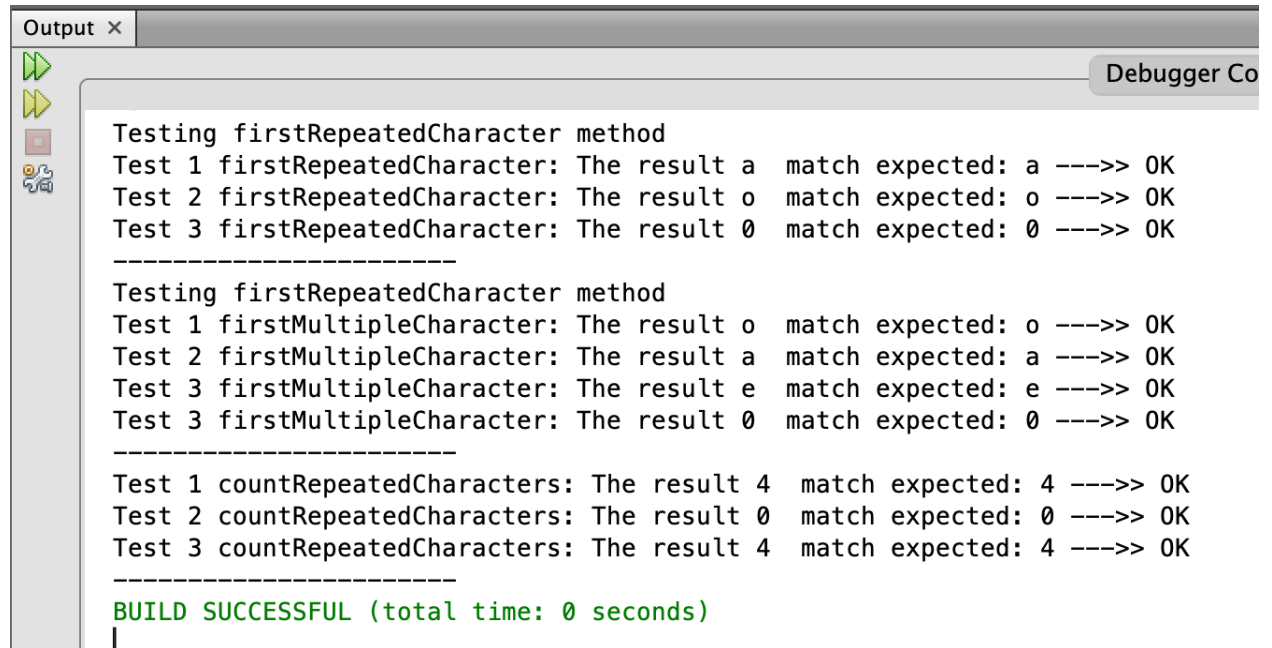
```
if (aString.charAt(i) == aString.charAt(i + 1))
```

tests for `yy`, that is, a character that is followed by another one just like it. But if we have a sequence `yyyy`, we only want it to count once. That's why we want to make sure that the preceding character is different:

```
if (aString.charAt(i - 1) != aString.charAt(i))
```

This logic works almost perfectly: it finds three group starts: `aabbcd`~~`aaa`~~`abb`

- 20) Why doesn't the method find the start of the first (`aa`) group? Why can't you simply fix the problem by letting `i` start at 0 in the `for` loop?
- 21) Go ahead and fix the bug. What is the code of your `countRepeatedCharacters` method now?
- 22) Run the main class again. What is the output now? What is output if you change the String for the last test case to `"aaaaabbcdaaaabbb"`? Is the program now free from bugs? That is not a question the debugger can answer. As the famous computer scientist [Edsger Dijkstra](#) pointed out: *"Program testing can be used to show the presence of bugs, but never to show their absence!"* As you have seen in this lab, testing and debugging is a laborious activity. In your computer science education, it pays to pay special attention to the tools and techniques that ensure correctness without testing.
- 23) When your program works and you are satisfied with the result, show your work to the instructional team to get checked off.



```
Output x
Debugger Co

Testing firstRepeatedCharacter method
Test 1 firstRepeatedCharacter: The result a match expected: a ----> OK
Test 2 firstRepeatedCharacter: The result o match expected: o ----> OK
Test 3 firstRepeatedCharacter: The result 0 match expected: 0 ----> OK
-----
Testing firstRepeatedCharacter method
Test 1 firstMultipleCharacter: The result o match expected: o ----> OK
Test 2 firstMultipleCharacter: The result a match expected: a ----> OK
Test 3 firstMultipleCharacter: The result e match expected: e ----> OK
Test 3 firstMultipleCharacter: The result 0 match expected: 0 ----> OK
-----
Test 1 countRepeatedCharacters: The result 4 match expected: 4 ----> OK
Test 2 countRepeatedCharacters: The result 0 match expected: 0 ----> OK
Test 3 countRepeatedCharacters: The result 4 match expected: 4 ----> OK
-----
BUILD SUCCESSFUL (total time: 0 seconds)
|
```

You're done with this lab!

So, what did you learn in this lab?

- How to read the stack trace
- Unit testing
- Using the debugger“

Lab Canvas submission

In this week's module, you will find the link to the lab completion submission. Submit the following files:

- a. Screenshots for parts A, B, C
- b. NetBeans project export