**Homework 1**

Chapter 1: Why do we test?

**Name:**

**What to do?**

Complete the problems below and submit this word document.

Four faulty programs are given below. Each includes test inputs that result in failure. For each of these four programs, answer the following questions **a — f**. For **“f”** copy and paste your corrected code in this document.

* 1. **findLast method (22 points)**

1. Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.
2. If possible, give a test case that does not execute the fault. If not, briefly explain why not.
3. If possible, give a test case that executes the fault, but does not result in an error state. If not, briefly explain why not.
4. If possible give a test case that results in an error, but not a failure. If not, briefly explain why not. Hint: Don’t forget about the program counter.
5. For the given test case below, describe the first error state. Be sure to describe the complete state. Hint: Don’t forget about the program counter.
6. Implement your repair and verify that the given test now produces the expected output. Submit a screenshot demonstrating your new program works.

|  |
| --- |
| */\*\*  \* Find last index of element  \*  \** ***@param x*** *array to search  \** ***@param y*** *value to look for  \** ***@return*** *last index of y in x; -1 if absent  \** ***@throws*** *NullPointerException if x is null  \*/* **private static int** findLast (**int**[] x, **int** y) {  **for** (**int** i = x.**length** - 1; i > 0; i--)  {  **if** (x[i] == y)  {  **return** i;  }  }  **return** -1; }  *// test: x = [2, 3, 5]; y = 2; Expected = 0* |

* 1. **countOccurrence method (22 points)**

1. Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.
2. If possible, give a test case that does not execute the fault. If not, briefly explain why not.
3. If possible, give a test case that executes the fault, but does not result in an error state. If not, briefly explain why not.
4. If possible give a test case that results in an error, but not a failure. If not, briefly explain why not. Hint: Don’t forget about the program counter.
5. For the given test case below, describe the first error state. Be sure to describe the complete state. Hint: Don’t forget about the program counter.

source = **"What is Software Testing? Why is Software Testing so important?"**;  
 word = **"Software"**;

Expected = 2

1. Implement your repair and verify that the given test now produces the expected output. Submit a screenshot demonstrating your new program works.

|  |
| --- |
| */\*\*  \* Counts the number of occurrence of a substring in another string  \*  \** ***@param source*** *string to use as a source of the match  \** ***@param word*** *string which is a substring of the source  \** ***@return*** *number of occurrences of word in source  \** ***@throws*** *NullPointerException if source is null  \*/* **private static int** countOccurrence(String source, String word) {  String[] temp = source.split(**" "**);  **int** count = 0;  **for** (String str : temp) {  **if** (word == str){  count++;  }  }  **return** count; }  *// Test:  // source = "What is Software Testing? Why is Software Testing so important?";  // word = "Software"; // Expected = 2* |

* 1. **countPositive method (22 points)**

1. Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.
2. If possible, give a test case that does not execute the fault. If not, briefly explain why not.
3. If possible, give a test case that executes the fault, but does not result in an error state. If not, briefly explain why not.
4. If possible give a test case that results in an error, but not a failure. If not, briefly explain why not. Hint: Don’t forget about the program counter.
5. For the given test case below, describe the first error state. Be sure to describe the complete state. Hint: Don’t forget about the program counter.

*x = [-4, 2, 0, 2]; Expected = 2*

1. Implement your repair and verify that the given test now produces the expected output. Submit a screenshot demonstrating your new program works.

|  |
| --- |
| */\*\*  \* Counts positive elements in array  \*  \** ***@param x*** *array to search  \** ***@return*** *number of positive elements in x  \** ***@throws*** *NullPointerException if x is null  \*/* **private static int** countPositive (**int**[] x) {  **int** count = 0;   **for** (**int** i = 0; i < x.**length**; i++)  {  **if** (x[i] >= 0)  {  count++;  }  }  **return** count; } *// test: x = [-4, 2, 0, 2]; Expected = 2* |

* 1. **addCommasToDigitString method (22 points)**

1. Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.
2. If possible, give a test case that does not execute the fault. If not, briefly explain why not.
3. If possible, give a test case that executes the fault, but does not result in an error state. If not, briefly explain why not.
4. If possible give a test case that results in an error, but not a failure. If not, briefly explain why not. Hint: Don’t forget about the program counter.
5. For the given test case below, describe the first error state. Be sure to describe the complete state. Hint: Don’t forget about the program counter.

*digits = "1234"; Expected = "1,234";*

1. Implement your repair and verify that the given test now produces the expected output. Submit a screenshot demonstrating your new program works.

|  |
| --- |
| */\*\*  \* Adds commas to long numbers at every third position starting from the right (e.g., "12345678" returns "12,345,678")  \*  \** ***@param digits*** *string represents a number  \** ***@return*** *a new string with a comma at every third position  \** ***@throws*** *NullPointerException if digits is null  \*/* **private static** String addCommasToDigitString(String digits) {   StringBuilder result = **new** StringBuilder();  **int** nDigits = 0;  **for** (**int** i = digits.length()-1; i >= 0; i--) {  result.insert(0, digits.charAt(i));  nDigits++;  **if** (((nDigits % 3) == 0) && (i > 1)) {  result.insert(0, **","**);  }  }  **return** result.toString(); }  *// Test: digits = "1234"; Expected = "1,234";* |

1. I'm looking for a paragraph to a page here. This is an introspective exercise without a "correct" answer, but with justification. The idea is for you to places yourself (and possibly your company) into the context described by Beizer's classification of levels of testing. You'll receive credit if you turn in something reasonable. If you have worked at a company that did software development, what level of test maturity do you think the company worked at? (0: testing=debugging, 1: testing shows correctness, 2: testing shows the program doesn’t work, 3: testing reduces risk, 4: testing is a mental discipline about quality) If you have not worked at in this scenario, think about a large project you worked on as a team and answer what level of test maturity do you think that project worked at? **(12 points)**

**Due Date**

This homework is due by **Sunday, February 6, 2022, 11:59 pm.** A penalty of 10% per day will be deducted from your grade, starting at 12:00:01 am.

**What to submit?**

Submit the following file to Blackboard:

* A word document describing your answers to the questions above.