**Course: Software Testing**

**Lab. Report #2 – Automated Requirements-Based API Unit Testing using JUnit**

|  |  |
| --- | --- |
| Group #: | G59 |
| Student Names: | Patrick Hamill |
| James Cassidy |

# Unit testing plan

**Patrick’s Unit Testing Plan**

Going forward with lab 2, I’ve decided to use Equivalence class testing, or Equivalence class partitioning as my Black Box Testing Technique. Equivalence partitioning is an important testing technique used by testers to group different input data deriving tests cases that offer a wide level of coverage for the system under test. It is an important and useful foundation for many other techniques and approaches in software testing, therefore I thought it an appropriate candidate for this lab.

From the lectures and lecture slides, I plan on deriving my test Cases in the following way;

* Firstly, I will use the Javadoc for JFree Chart to ascertain 5 suitable methods/ functions to test within the Range Class, identifying the appropriate input domains for each.
* Next, I will partition the test data into equivalent classes depending on the input data, for instance, using the example from the lecture slides, partitioning the test data into what can be inputted for data and column
* Finally, using Strong Equivalence class testing, I will combine each class into multi-dimensional test cases.

Lastly, I will organize my Junit test suite based on Appendix C, splitting each class into its constituent test class, refraining from writing multiple classes in the same file. I also intend to stick to the naming convention of prefixing each test with the word “test”, to read and understand the test code more quickly and accurately. Furthermore, when it is required, I plan on using alternative 1 when asserting for expected exceptions in Junit. I hope to stay as consistent as possible, ensuring a high level of understanding and readability throughout.

**James’ Unit Testing Plan**

Before I start my black box testing on the Data Utilities Class, equivalence partitioning will need to be undertaken. This technique involves partitioning the input domain of each method and deriving test cases from the wide range of values that can be input into those methods. From this technique I can derive test cases using Weak Equivalence-Class Testing (WECT) and Strong Equivalence-Class Testing (SECT). This ensures the system under test has a wide array of tests available that cover all aspects of a method.

With WECT there can be too few test cases and SECT there can be too many test cases. To find a middle ground for deriving test cases, pair-wise testing will be undertaken. This ensures maximum coverage of the system under test whilst minimizing the time spent on each method. This allows me to efficiently tackle each of the five methods in the Data Utilities Class.

Following Appendix C, each test method will begin with ‘test’. This will remain constant though out my entire testing. Test methods will also be grouped together by the method tested. For example, every calculateColumnTotal test method will be grouped together within DataUtilitiesTest.java instead of one test method beginning at the top whilst another is at the very bottom of the java file. Comments will be used to separate each test method. Allows the user to clearly see what test method belongs to the respective method.

# Designing the unit test-cases using black-box test-case design techniques

## Patrick’s Test Cases

### public Boolean contains(double value)

Input Domain: value, bounds

Value:

|  |  |  |
| --- | --- | --- |
| value < lower bound | lower bound < value < upper bound | value > upper bound |

Bounds:

|  |  |  |  |
| --- | --- | --- | --- |
| LB < 0 & UB >0 | LB > 0 & UB > 0 | LB < 0 & UB < 0 | LB = UB |

|  |  |  |  |
| --- | --- | --- | --- |
| TC | Bounds | Value | Expected output |
| 1 | -5 & 5 | -6 | False |
| 2 | -5 & 5 | 1 | True |
| 3 | -5 & 5 | 6 | False |
| 4 | 1 & 5 | -6 | False |
| 5 | 1 & 5 | 3 | True |
| 6 | 1 & 5 | 6 | False |
| 7 | -5 & -1 | -6 | False |
| 8 | -5 & -1 | -3 | True |
| 9 | -5 & -1 | 6 | False |
| 10 | 1 & 1 | -6 | False |
| 11 | 1 & 1 | 1 | True |
| 12 | 1 & 1 | 6 | False |

### public double getLength()

Input Domain: bounds

Bounds :

|  |  |  |  |
| --- | --- | --- | --- |
| LB < 0 & UB >0 | LB > 0 & UB > 0 | LB < 0 & UB < 0 | LB = UB |

|  |  |  |
| --- | --- | --- |
| TC | Bounds | Expected output |
| 1 | -5 & 5 | 10 |
| 2 | 1 & 5 | 4 |
| 3 | -5 & -1 | 4 |
| 4 | 1 & 1 | 0 |

### public Boolean intersects(double b0, double b1)

Input Domain: Range 1 bounds, Range 2 Bounds

Range 1 Bounds :

|  |  |  |  |
| --- | --- | --- | --- |
| LB < 0 & UB >0 | LB > 0 & UB > 0 | LB < 0 & UB < 0 | LB = UB |

Range 2 Bounds :

|  |  |  |  |
| --- | --- | --- | --- |
| LB < 0 & UB >0 | LB > 0 & UB > 0 | LB < 0 & UB < 0 | LB = UB |

|  |  |  |  |
| --- | --- | --- | --- |
| TC | Range 1 Bounds | Range 2 Bounds | Expected output |
| 1 | -5 & 5 | -3 & 10 | True |
| 2 | -5 & 5 | 3 & 10 | True |
| 3 | -5 & 5 | -12 & - 7 | False |
| 4 | -5 & 5 | 1 & 1 | True |
| 5 | 1 & 5 | -3 & 10 | True |
| 6 | 1 & 5 | 3 & 10 | True |
| 7 | 1 & 5 | -12 & - 7 | False |
| 8 | 1 & 5 | 1 & 1 | True |
| 9 | -5 & -1 | -3 & 10 | True |
| 10 | -5 & -1 | 3 & 10 | True |
| 11 | -5 & -1 | -12 & - 7 | False |
| 12 | -5 & -1 | 1 & 1 | True |
| 13 | 1 & 1 | -3 & 10 | False |
| 14 | 1 & 1 | 3 & 10 | False |
| 15 | 1 & 1 | -12 & - 7 | False |
| 16 | 1 & 1 | 1 & 1 | True |

### public double constrain(double value)

Input Domain: Range bounds, value

Range Bounds :

|  |  |  |  |
| --- | --- | --- | --- |
| LB < 0 & UB >0 | LB > 0 & UB > 0 | LB < 0 & UB < 0 | LB = UB |

Value :

|  |  |  |
| --- | --- | --- |
| Value < LB | Value > UB | LB <= Value <=UB |

|  |  |  |  |
| --- | --- | --- | --- |
| TC | Bounds | Value | Expected output |
| 1 | -5 & 5 | -6 | -5 |
| 2 | -5 & 5 | 6 | 5 |
| 3 | -5 & 5 | 3 | 3 |
| 4 | 1 & 5 | -6 | 1 |
| 5 | 1 & 5 | 6 | 5 |
| 6 | 1 & 5 | 3 | 3 |
| 7 | -5 & -1 | -6 | -5 |
| 8 | -5 & -1 | 6 | -1 |
| 9 | -5 & -1 | -3 | -3 |
| 10 | 1 & 1 | -6 | 1 |
| 11 | 1 & 1 | 1 | 1 |
| 12 | 1 & 1 | 6 | 1 |

### public Range expandToInclude(Range range, double Value)

Input Domain: Range bounds, value

Range Bounds :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LB < 0 & UB >0 | LB > 0 & UB > 0 | LB < 0 & UB < 0 | LB = UB | null |

Value :

|  |  |  |
| --- | --- | --- |
| Value < LB | Value > UB | LB <= Value <=UB |

|  |  |  |  |
| --- | --- | --- | --- |
| TC | Range Bounds | Value | Expected output |
| 1 | -5 & 5 | -6 | Range expanded to include -6 |
| 2 | -5 & 5 | 10 | Range expanded to include 10 |
| 3 | -5 & 5 | 1 | Original Range |
| 4 | 1 & 5 | -6 | Range expanded to include -6 |
| 5 | 1 & 5 | 10 | Range expanded to include 10 |
| 6 | 1 & 5 | 1 | Original Range |
| 7 | -5 & -1 | -6 | Range expanded to include -6 |
| 8 | -5 & -1 | 10 | Range expanded to include 10 |
| 9 | -5 & -1 | -3 | Original Range |
| 10 | 1 & 1 | -6 | Range expanded to include -6 |
| 11 | 1 & 1 | 10 | Range expanded to include 10 |
| 12 | 1 & 1 | 1 | Original Range |
| 13 | Null | -6 | New Range(-6, -6) |
| 14 | Null | 10 | New Range(10 ,10) |

## James’ Test Cases

### calculateColumnTotal

**Input domain:** calculateColumnTotal(Values2D data, int column)

Data:

|  |  |
| --- | --- |
| Valid 2D matrices  Includes decimal, negative and whole numbers | Everything else (invalid inputs) |

Col:

|  |  |  |
| --- | --- | --- |
| Negative number  (invalid) | Valid: 0<= col <= # of columns in data matrix-1 | col > # of columns in data matrix-1  (invalid) |

|  |  |  |  |
| --- | --- | --- | --- |
| **TC#** | **Data** | **col** | **Expected output:** |
| 1 | [1 0 0  4 1 0] | Valid  0 | 5 |
| 2 | [3 4 8  2 5 11] | Valid  1 | 9 |
| 3 | [3 4 8  2 5.4 11] | Valid  1 | 9.4 |
| 4 | [3 4 8  -2 5.4 11] | Invalid  0 | 1 |
| 5 | [3 4 8  -2 5.4 11] | Invalid  55 | 0 per Javadoc specification |
| 6 | [3 4 8  -2 5.4 11] | Invalid  -1 | 0 per Javadoc specification |
| 7 | null | Valid  1 | InvalidParameterException |
| 8 | [3.4 null null  null null null] | Valid  0 | IllegalArguementExcepetion |
| 9 | [null null null  null null null] | Invalid  11 | IllegalArguementException |
| 10 | [null null 3  null null 15] | Valid  2 | IllegalArguementException |

### calculateRowTotal

**Input domain** calculateRowTotal(Values2D data, int row)

Data:

|  |  |
| --- | --- |
| Valid 2D double matrices  Includes decimal, negative and whole numbers | Everything else (**Invalid)** |

Row:

|  |  |  |
| --- | --- | --- |
| Negative number  **(invalid)** | Valid: 0<= row <= # of columns in data matrix-1 | col > # of rows in data matrix-1  (invalid) |

|  |  |  |  |
| --- | --- | --- | --- |
| **TC#** | **Data** | **col** | **Expected output:** |
| 1 | [3 4 8  2 5 11] | Valid  0 | 15 |
| 2 | [3 4 8  2 5 11] | Valid  1 | 18 |
| 3 | [null null null  -2 5.4 11] | Valid  0 | IllegalArguementException |
| 4 | [null null null  -2 5.4 11] | Valid  1 | IllegalArguementException |
| 5 | [3 4 8  -2 5.4 11] | Invalid  55 | 0 per specification |
| 6 | [3 4 8  -2 5.4 11] | Invalid  -1 | 0 per specification |
| 7 | null | Valid  1 | InvalidParameterException |
| 8 | [3.4 null null  null null 11] | Valid  0 | IllegalArguementExcepetion |
| 9 | [3.4 null null  null null 11l] | Invalid  22 | IllegalArguementException |
| 10 | [3 4 8  -2 -5 -11] | Valid  1 | -18 |
| 11 | [3.3 4.4 8.3  2 5 11] | Valid  0 | 15 |

### createNumberArray

**Input Domain** createNumberArray(double[] data)

**Data:**

|  |  |
| --- | --- |
| **Valid Double Array**  Includes decimal, negative and whole numbers | Everything else (invalid inputs) |

|  |  |  |
| --- | --- | --- |
| **TC#** | **Data** | **Expected Result** |
| 1 | **Expected**  {2.0}  **Actual**  {2.0} | {2.0} |
| 2 | **Expected**  {5.0, 10.0, -15.0, 3.0}  **Actual**  {5.0, 10.0, -15.0, 3.0} | {5.0, 10.0, -15.0, 3.0} |
| 3 | **Expected**  {5.0, 10.0, -15.0, 3.0}  **Actual**  {5, 10, 15, 20} | {5, 10, 15, 20} |
| 4 | **Expected**  {5.0, 10.0, -15.0, 3.0}  **Actual**  {5, 10, 15} | {5, 10, 15} |
| 5 | **Expected**  {5.0, 10.0, -15.0, 3.0}  **Actual**  {5.0, 10.0, -15.0, 3.0} | {5.0, 10.0, -15.0} |
| 6 | **null** | Invalid Parameter Exception |

### createNumberArray2D

**Input Domain** createNumberArray2D(double[][] data)

**Data:**

|  |  |
| --- | --- |
| Valid 2D Double Array  Includes decimal, negative, and whole numbers | Everything else (invalid inputs) |

|  |  |  |
| --- | --- | --- |
| **TC#** | **Data** | **Expected Result** |
| 1 | **Expected**  {1.0}  **Actual**  {1.0} | {1.0} |
| 2 | **Expected**  {1.0, 2.11, -1.4, 0.0} ,{4.0, -8.0, 16.0, -24.0}  **Actual**  {1.0, 2.11, -1.4, 0.0} ,{4.0, -8.0, 16.0, -24.0} | {1.0, 2.11, -1.4, 0.0} {4.0, -8.0, 16.0, -24.0} |
| 3 | **Actual**  {5, 10, 15, 20},{ 25, 30, 35, 40 }  **Expected**  {1.0, 2.11, -1.4, 0.0} ,{4.0, -8.0, 16.0, -24.0} | {5, 10, 15, 20},{ 25, 30, 35, 40 } |
| 4 | **Actual**  {5, 10, 15, 20},{ 25, 30, 35 }  **Expected**  {1.0, 2.11, -1.4, 0.0} ,{4.0, -8.0, 16.0, -24.0} | {5, 10, 15, 20},{ 25, 30, 35 } |
| 5 | **null** | Invalid Parameter Exception per Javadoc specification |

### getCumulativePercentages

**Input Domain**: getCumulativePercenatges(KeyedValues data)

**Data**

|  |  |
| --- | --- |
| **Key** | **Value** |
| **Valid**  Comparable | **Valid Double input**  Includes decimal, negative, and whole numbers |
| **Invalid**  Anything else | **Invalid**  Anything Else |

|  |  |  |
| --- | --- | --- |
| **TC#** | **Key | Data** | **Expected Result** |
| 1 | 0 5  1 9  2 2 | 1.0  getValue(2) |
| 2 | 0 5  1 9  2 -15 | 0.875  getValue(1) |
| 3 | 0 2.5  1 5.7  2 1.8 | 0.82  getValue(1) |
| 4 | 0 5  1 9  2 -15 | Invalid Parameter Exception per Javadoc specifications  getValue(1) |
| 5 | 0 5  1 9  2 -15 | Invalid Parameter Exception per Javadoc specifications  getValue(3) |
| 6 | 0 -5  1 -9  2 -15 | 1.0  getValue(2) |
| 7 | Null | Invalid Parameter Exception per Javadoc specifications |
| 8 | Null 5  Null 9  Null 2 | IllegalArguementException |
| 9 | 0 null  1 null  2 null | IllegalArguementException  getValue(2) |

# Output of test suite execution

## Screenshot of test-suite execution in JUnit (showing the names of test methods)

### Patrick’s Test Suite Execution

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

### James’ Test Suite Execution

Text

Description automatically generated

## List of failed test cases, and the possible defects based on that information

### Patrick’s failed test Cases and possible Defects

Intersects

* testRangeNegativeThreeToPositiveTenDoesNotIntersectsRangeFour
* testRangeNegativeTwelveToNegativeSevenDoesNotIntersectRangeThree
* testRangeNegativeThreeToPositiveTenIntersectsRangeOne
* testRangeNegativeTwelveToNegativeSevenDoesNotIntersectRangeOne
* testRangePositiveThreeToPositiveTenIntersectsRangeOne
* testRangePositiveThreeToPositiveTenIntersectsRangeTwo
* testRangeNegativeTwelveToNegativeSevenDoesNotIntersectRangeTwo
* testRangeNegativeThreeToPositiveTenIntersectsRangeThree
* testRangePositiveThreeToPositiveTenIntersectsRangeThree
* testRangePositiveOneToPositiveOneIntersectsRangeThree
* testRangeNegativeTwelveToNegativeSevenDoesNotIntersectRangeFour

The Javadoc entry for the Intersects method doesn’t provide much detail into what exactly is achieved only stating that it “*Returns true if the range intersects with the specified range, and false otherwise*.”. As the ranges are directionless, I interpreted this to mean that as long as the 2 ranges overlapped each other in some way, they would ‘*Intersect’.* For example, if Range1 = -5 - 5 and Range2 = 0 - 10 these two ranges would intersect from 0-5 as there was some overlap. This interpretation could be completely false, resulting in the failed tests, however with the information I was given this was the approach I took.

From the failed and positive test cases, I’ve come to have a deeper understanding of what the intersect method is checking for. This may have been an intentional defect, or just a lack of detail in the definition of the method but intersects appears to return true if one range is entirely within the other. For Example, the test case “testRangeNegativeThreeToPositiveTenIntersectsRangeTwo” returns true, with Range2 equaling 1-5 while the test case “testRangePositiveThreeToPositiveTen IntersectsRangeTwo” returns false. With my initial interpretation of the intersects method, both these test cases should return true, with the first overlapping completely Rang2 and the second test case overlapping from 3 – 5.

Constrain

* testConstrainNegativeSixReturnsPositiveOneForRangeTwo
* testConstrainNegativeSixReturnsNegativeFiveForRangeThree
* testConstrainNegativeSixReturnsNegativeFiveForRangeOne

The constrain method should return the value within the range that is closest to the value given to the method. I have taken this and derived 3 conditions; the lower bound is returned if the value is less than the lower bound, the upper bound is returned if the value is greater than the upper bound, or the value is returned if it is within the Range. After writing tests to accommodate these conditions I have found 3 test cases that fail. All 3 are testing a negative value that is less than the lower bound, this would indicate that there is a fault in the code that isn’t testing negative numbers unless they are within the range, however if this was the case, test case “testConstrainNegativeSixReturnsPositiveOneForRangeFour” should also return a failure, however this is not the case. Without looking into the source code, I can’t definitively say the reason why those 3 test cases failed.

Expand To Include

* testValueAlreadyInTheRangeForRangeFour
* testValueAlreadyInTheRangeForRangeTwo
* testValueLessThanLBForRangeOne
* testValueLessThanLBForRangeThree
* testValueLessThanLBForRangeTwo
* testValueLessThanLBForRangeFour

The Java doc entry for expand to include was a bit confusing, however, I elected to go with this return statement “A range which spans over the input range and has been expanded to contain the input value” and developed my test cases accordingly. All 6 failed test cases appear to indicate a fault when the method interacts with the lower bound of the away. It fails to alter the lower bound to encompass the new value wishing to be entered, as well as failing to recognize that the value is already in the range if it equals the lower bound. Examination of the source code would confirm this or not, but without doing so, this appears to be the most logical option.

### James’ Failed Test Cases and Possible Defects

### calculateColumnTotal

* testValidDataTotalColumnTwo
* testValidDataTotalColumnTwoDecimalNumber
* testValidDataTotalColumnOneNegativeNumber
* testValidDataTotalColumnInvalid
* testValidDataTotalColumnInvalidNegativeNumber
* testNullInputCalulateColumnTwoTotal

Several defects were discovered in testing the calculateColumnTotal method. Column 2 does not function as intended as seen from testValidDataTotalColumnTwo. The answer should be 9 but the test fails to return this. When testing decimal numbers in column two there is also a failure as seen from the testValidDataTotalColumnTwoDecimalNumber. In retrospect, this may be because there is an issue with column two itself and not with decimal numbers. In the test case testValidDataTotalColumnOneNegativeNumber there is a failure. This may be due to a negative number being present within the test case. I come to this conclusion as in the test case testValidDataAndColumnColumnTotal there is no negative number in column one and the test case passes. This can therefore be seen as a defect. Test case testValidDataTotalColumnInvalid fails as there is an invalid column number of 55 within the test case. In the setup of this method, only 3 columns exist and therefore the test fails. The Javadoc states ‘With invalid input, a total of zero will be returned.’ This was not the case. When attempting to create this test case to return zero, the complier only returned an error. A similar conclusion can be drawn with the test case testValidDataTotalColumnInvalidNegativeNumber as a negative column number can’t exist. A zero could not be returned as Eclipse returned an error. Finally, test case testValidDataTotalColumnInvalidNegativeNumber fails as a null input has been passed through the method. A null value is not permitted per the specifications of the Javadoc. The Javadoc states a InvalidParameterException should be returned is a null value is passed so the test should fauls. Therefore, this must be a defect that exists within the calculateColumnTotal method.

### calculateRowTotal

* testValidInputTotalRowOne
* testValidInpuTotaltRowTwo
* testRowTotalInvalidRowNumber
* testRowTotalNegativeRowNumber
* testRowTotalNullInput
* testValidInputRowTwoTotalNegativeNumbers
* testValidInputRowOneTotalDecimalNumber

Several defects were also detected on calculateRowTotal. Test cases testValidInputTotalRowOne and testValidInpuTotaltRowTwo both failed. Correct row numbers were entered along with the correct sum for that respective row. Both failed, however. Because of this, it is most likely test cases with negative and decimal numbers will fail, but not because they are negative and decimal numbers. This is likely why test cases testValidInputRowTwoTotalNegativeNumbers and testValidInputRowOneTotalDecimalNumber fail. The Javadoc states ‘With invalid input, a total of zero will be returned.’ However, when test cases were initially created in Eclipse, the complier returned an error, stating that this was not possible. This was initially done on test cases testRowTotalInvalidRowNumber and testRowTotalNegativeRowNumber which both failed. The test case testRowTotalNullInput also failed. The Javadoc states that an invalid parameter exception should be thrown as null input is not permitted.

### createNumberArray

* testArraysDontMatchDifferentLength
* testThrowsExpectionWhenInputArrayIsNull

Less issues appeared for createNumberArray method. The test case testArraysDontMatchDifferentLength failed. This was expected as the expected array contains only 3 values while the actual array contains 4. And the test case testThrowsExpectionWhenInputArrayIsNull also failed as null values are not permitted per the Javadoc specifications.

### createNumberArray2D

* testThrowsExpectionWhenInput2DArrayNull

The test case testThrowsExpectionWhenInput2DArrayNull failed as null values are not permitted per the Javadoc specifications.

Upon further inspection, the test case test2DArraysDontMatchDifferentLength passed. I expected this to fail as both 2D arrays have different lengths compared to each other. This can therefore be seen as a defect.

### getCumulativePercentages

* testReturnKeyValueTwo
* testReturnKeyValueOne
* testReturnKeyValueOneDecimalValue
* testReturnKeyValueOneDividedByNegativeNumber
* testReturnInvalidKeyValue
* testNegativeValueLastCumulativePercentageShouldBeOne
* testNullInputInvalidParameterType
* testNullValues

This method experienced the most the failures out of all 5 of the DataUtilies methods. The test case testNullKeyValuesIllegalExperssion was the only test case that passed. The key values were all null and therefore were illegal arguments. The test case was then tailored accordingly to this. The failures in the getCumulativePercentages may be attributed to the fact I had to use ‘@SuppressWarnings("rawtypes")’ before getting any test case to be able to run. Although the Javadoc makes it clear as to what values are legal for KeyedValues, in practice, Eclipse would not allow me to enter the specific values I wanted. I understood how the method worked, but it was not clear how exactly they should be entered. For example, for test case testReturnKeyValueTwo, the value returned should be 1 as getValue is used on the last value in the method, therefore it is the total divided by the total which should return 1. This failed however due to a defect in the method or my lack of understanding of the method. I expected test case testReturnKeyValueOneDividedByNegativeNumber to fail as 14/-1 should equal -14 which is not permitted as the value returned should be between 0.0 and 1.0.

# How the TEAMWORK/effort was divided and managed

## How the TEAMWORK/effort of the lab was managed and divided

In undertaking this task, we agreed to each take 5 methods. Patrick designed a test case plan for Range and James designed test cases for all of 5 methods of Data Utilities.

With other deadlines taking place for other modules, we decided it was best to undertake one meeting a week. These were usually done in person, but a couple of short meetings were arranged on Teams.

When implementing our test cases, we did run into a few issues.

For the Data Utilities class, there were a couple of the methods that proved quite difficult to implement test methods. James had more issues with implementing his Data Utilities test methods compared with Patrick implanting Range Test Methods. This might have been the case due to the Data Utilities only having 5 methods. If James struggled with implementation of test method, he had to push on through and try and find a solution. In Patrick’s case, if he struggled with implementing test methods for a method within the Range class, he could simply pick another method from the 15 available.

To combat this, we collaborated several times during our time working on this lab. As Patrick struggled less with the 5 methods he was working with and was able to complete them quicker, he was able to help James in understanding the Javadoc for Data Utilities methods. Having a second pair of eyes look at Data Utilities and get a fresh perspective helped in completing the test cases quicker.

## Writing the lab report

Fill up the following table to specify who wrote what part of the lab document:

|  |  |
| --- | --- |
| **Lab-report section** | **Written by** |
| 1- Unit Testing Plan | James Cassidy and Patrick Hamill |
| 2-Designing the unit test-cases using black-box test-case design techniques | James Cassidy and Patrick Hamill |
| 3-Output of Test Suite Execution | James Cassidy and Patrick Hamill |
| 4-Hows the Teamwork was Divided and Managed | James Cassidy |
| 5-Difficulties/ challenges encountered, overcoming them, and lessons learned | Patrick Hamill |
| 6- Difficulties/ challenges encountered, overcoming them, and lessons learned | Patrick Hamill. |

## Lessons learned from your teamwork in this lab

Although we went our separate ways in completing this lab with Patrick completing test cases for the Range class and James completing test cases for the Data Utilities, there were several times when teamwork helped complete our tasks more effectively.

As Patrick is only undertaking 2 modules this semester and James is taking 3, we agreed that Patrick could work on one more section of the report to lessen the workload that James already had during the time this lab took place.

Once we had looked over and derived a test plan from the Javadoc, we were able to meet and discuss with each other to make sure what we had devised was sufficient for what was being asked of us in the Lab document.

And finally, when either of us had trouble developing our test cases in eclipse, we were able to arrange a Teams and meet in the CSB labs to solve each other’s issues.

# Difficulties/ challenges encountered, overcoming them, and lessons learned

This section has the following sub-sections.

## Difficulties/ challenges encountered

As the semester carried on, other modules began releasing assignments that coincided with Software testing. Luckily, Labs 0 and 1 took place before other class assessments, which allowed us to focus our time on those Labs, however, as we both took Network Security, we began devoting some time to completing those assignments and to a small degree, fell behind with our work on this Lab.

In terms of difficulties encountered while completing the Lab work, we were fortunate to only run into a few issues, centering on setting up the project. The main issue here revolved around adding the correct referenced libraries to the project. For some reason we struggled to get the project to work after following the steps in the lab document, even after following the videos provided, we tried several times on our local machines to get it working, uninstalling, and reinstalling eclipse renaming files but to no avail.

## How did you overcome the above difficulties/ challenges?

After completing work for our other module, we realized that we had fallen significantly behind for Lab 2, which quickly led to issues with Lab 3 due to them being linked. As a result, we got together to work out a plan of action for how to catch up. We decided that communication was key, and that we should stay in touch when it came to our progress with completing the lab report, even if we were doing our individual tasks separately. We decided on some short-term goals and divided the lab work accordingly to ensure an efficient work balance, not only for this lab, but for our other assignments as well.

We still aren’t clear on what issue we were having trying to add the referenced libraries to our project. After one of the attempts to reinstall eclipse and download a fresh artifacts file from canvas, the program began working, we tried our best to figure out what had changed, but to our knowledge nothing of significance had been altered. Our best guess was that we created the java project in a higher-level file, not buried into sub files, this had caused issues in the past with previous projects, so I assumed a similar problem had occurred here.

## Lessons learned

From completing Lab 2, we both took away the importance of time management, and communication. It’s very easy and quick for one task being late to have a knock-on effect. For Lab 3, and other assignments going forward, this is an area that we have learned to put a lot of focus and attention to detail to ensure the smooth completion of our work, while also putting aside the necessary time to complete all of our assignments.

# Comments/feedback on the lab and lab document itself

This section has the following sub-sections.

## Did you find the lab a useful learning experience? How it helped you learn the new testing topics

We both agree that we found the Lab to be an extremely useful learning experience. Its one thing to learn about the different testing techniques and another to actually put into practice what was taught. From the lectures, Equivalence class testing is set out as the foundation used for computer scientists to develop their skills in other black box testing techniques, and as such, we decided it would be an appropriate starting point for this lab.

## Was the lab document easy to follow?

Overall, the lab document was straight forward and reasonably easy to follow. The nature of the lab required a high level of text written instructions which made the document appear very wordy, this made it slightly difficult to follow due to easily getting lost and mixing up steps. The inclusion of images and videos helped to make more sense of the lengthy directions providing more in-depth explanations for each step.

## About time budget? (Was there too much/too little time for this lab?)

In terms of the budget for this Lab, there was more than enough time allocated to complete the work well before the submission deadline, however, as previously mentioned, the extra time proved to be extremely useful as it gave us the flexibility to work on other assignments alongside this one.

We initially didn’t budget out time well and took advantage of the long deadline to fall behind, however this was quickly rectified when we realized that it was having a knock-on effect with Lab 3, and we didn’t want to be struggling the last few weeks of the semester.

## Please provide your comments on how to improve the lab work and lab document

Although we found the lab document relatively straightforward and easy to follow, the sea of text made it more difficult to follow some instructions as it was easy to lose our place the document. The images proved helpful in demonstrating what to do, however, a slight criticism would be that the images didn’t always line up with what we were experiencing. This mis alignment could have been a result of using an older version of eclipse. To help improve the lab document for the future, it might be useful to go through the story boards and examples, ensuring they match what a student will see when trying to follow them, and with the nature of the lab document, the chunks of text for now appear unavoidable