Lab. Report #2

Requirements-Based Test Generation

SENG 438 - Software Testing, Reliability, and Quality

Group #:	18
Student Names:	Ashley Brown
	Nathan Godard
	Rebecca Reid
	Ines Rosito

Table of Contents

1. Introduction

This report is an overview of our approach to discovering bugs and the results of testing components of the JFreeChart program. Specifically, the functions associated with the Range and DataUtilities class were tested using the unit test approach. *[To view which specific functions were tested view Section 2.1 and 2.2 of this document]* The document will describe how the work was divided, how test cases were designed and other strategies used by members of the group to complete the testing presented in this report.

Due to the fact that only requirements of the program were provided, black-box techniques were applied to the testing. In order to achieve the task set out by the lab, automated testing frameworks and mocking was used in order to conduct the unit test on the SUT. The testing process began by having each member take responsibility for testing one method from both the Range and DataUtilities class. Due to the team being comprised of 4 members, one individual had an extra method to test from Range and another had an extra method from DataUtilities. This allowed for complete coverage to meet the requirement of testing a total of 10 methods. Through team discussion, a plan was devised to determine what should be tested within each unit test. This involved determining the input variables for methods, partitions of equivalence classes, and boundary value analysis. *[More details provided in Section 2]*

It should be noted that prior to this lab, the group had minimal experience with proper testing frameworks. For each member, it was the first time using JUnit, jMock, and mocking in general.

2. Unit Test Strategy

For each method being tested, inputs and any constraints were determined and recorded when familiarizing ourselves on the classes through the use of the JavaDocs provided. *[View Sections 2.1 and 2.2 for more details]* With this information, equivalence class were determined. For most methods, testing included a variety of expected value class, as well as, invalid input classes. When applicable and deemed relevant, boundary values were included. Few boundaries were included, due to many of the methods providing no constraints on range when inputting numeric values.

It should be noted that due to the large quantity of equivalence classes identified for the RangeTest class we created, we implemented weak ECT. Without it, there would have been approximately over 100 cases to test and that includes the boundary cases. Generic test cases were developed, and then ran multiple times with a variety of different inputs. *[View Section 2.1 for list of inputs]* Through this method we were able to implement weak ECT while still only coding one test per method.

In the case of methods contains() and constrain(), BVT was implemented. This was due to the increased importance of the boundaries of those methods, compared to other methods of the SUT. It should be noted that the tests were restricted in regards to Java not allowing us to test methods using non-numeric inputs. This is due to the JVM preventing code from writing when there is a type mismatch.

Pros vs. Cons to Mocking

Mocking is a powerful tool in testing, especially when it is impractical and/or impossible to incorporate real objects into a unit test. Mockings ability to remove external dependencies and focus upon a single unit of the SUT is amongst its key features. Mock objects are capable of mimicking behaviour in controlled ways. Therefore, once a test is finished these objects are review its interaction with the class under test, allosing testers to precisely pinpoint the source of failures.

Amongst its greater downsides, when implementing mocks the amount of code, and therefore the complexity of it, increased. It also impacts the readability of the tests, which brings along its own set of complications. Testing with real instances of objects would result in much simpler code.

Another issue encountered with mocking, or with black-box testing as a whole, is that there is a lack of clarity toward what exactly the data the mocked objects are suppose to return. Since we are not sure as to how the mock and the method being tested interact, tests can easily fail due to the lack of robustness from the mock.

2.1 Constraints and Equivalence Classes -- Range:

As discussed in the previous section of this report, our method for which to test Range was slightly different from typical weak ECT convention. As a result, the following are equivalence classes determined by the upper and lower bounds of the Range class. These classes were then used and tested for each method (Boundary cases denoted BC):

- Negative value
- Positive value
- NULL
- Double.MAX_VALUE (BC)
- negative Double.MAX_VALUE (BC)
- 0 (BC)
- Non-numeric -- considered Untestable

2.1.1 `contains(`*`double value`*`):`

Method accepts the primitive data type `double`, and should returns `true` if the specified value is within the range, otherwise it returns `false`.
Notation: `value` input is denoted with "v". Range is to be denoted as [x,y] where x and y are of type `value` and y≥ x
The following are the potential equivalence classes:

- x < v <y
- v < x < y
- x < y < v
- x = v
- y = v

2.1.2 `constrain(`*`double value`*`):`

Method accepts the primitive data type `double`, and should return the value within the range of type `double` that is closest to the specified value.
Notation: `value` input is denoted with "v". Range is to be denoted as [x,y] where x and y are of type `value` and y≥ x
The input is constrained as follows:

- x < v <y
- v < x < y
- x < y < v
- x = v
- y = v

2.1.3 `getLength():`

This method does not require an input and should return the length of the range of type `double`.

2.1.4 `getUpperBounds():`

This method does not require an input and should return the upper bound for the range of type `double`.

2.1.5 `getLowerBounds():`

This method does not require an input and should return the lower bound for the range of type `double`.

2.2 Constraints and Equivalence Classes -- DataUtilities:

2.2.1 `calculateColumnTotal(Values2D data, int column):`

Method accepts a `Values2D` object, and a column index of type `int`, and should return the sum of the values in one column defined by the arguments, in the form of a `static double`. The input is constrained as follows:

- `data` != null -- throws `InvalidParameterException` if invalid object passed
- `column` >= 0
- `column` < the number of columns in data

Therefore the equivalence classes and boundary cases were determined to be:

- *Boundary case*: `column` = Integer.MAX_VALUE
  `column` = - Integer.MAX_VALUE
- *Boundary case*: `column` = 0
- *Boundary case*: `column` = NULL
- *Boundary case*: `data` = NULL
- *Equivalence class*: `data` contains NULL
  `column` = column in data containing NULL
- *Equivalence class:* `data` contains Integer.MAX_VALUE or - Integer.MAX_VALUE
  `column` = column in data containing (+ or -) Integer.MAX_VALUE
- *Equivalence class*: `column` = (- Integer.MAX_VALUE, Integer.MAX_VALUE) -- not Inclusive

The following can be considered subsets to the equivalence class mentioned directly above:

- *Equivalence class*: `column` = [0, # of columns in data] -- inclusive
- *Equivalence class*: `column` = (# of columns in data, Integer.MAX_VALUE) -- not Inclusive

2.2.2 `calculateRowTotal(Values2D data, int row):`

Method accepts a `Values2D` object, and a row index of type `int`, and should return the sum of the values in one `column` defined by the arguments, in the form of a `static double`. The input is constrained as follows:

- `data` != null -- throws `InvalidParameterException` if invalid object passed
- `row` >= 0
- `row` < the number of rows in data

Therefore the equivalence classes and boundary cases were determined to be:

- *Boundary case*: `row` = Integer.MAX_VALUE
  `row` = - Integer.MAX_VALUE
- *Boundary case*: `row` = 0
- *Boundary case*: `row` = NULL
- *Boundary case*: `data` = NULL
- *Equivalence class*: `data` contains NULL
  `row` = row in data containing NULL

- *Equivalence class:* `data` contains Integer.MAX_VALUE or - Integer.MAX_VALUE
  `row` = row in data containing (+ or -) Integer.MAX_VALUE
- *Equivalence class*: `row` = (- Integer.MAX_VALUE, Integer.MAX_VALUE) -- not Inclusive

The following can be considered subsets to the equivalence class mentioned directly above:

- *Equivalence class*: `row` = [0, # of row in data] -- inclusive
- *Equivalence class*: `row` = (# of row in data, Integer.MAX_VALUE) -- not Inclusive

2.2.3 `createNumberArray(`*`double [] data`*`)`:

Method accepts an array of `doubles`, and should construct and return an array of `Number` objects from the provided arguments.
Assumption: array of doubles is 1:1 for the array of numbers. Meaning, same number of rows, columns, and same digits from one array to another.
The input is constrained as follows:

- `data` != null -- throws `InvalidParameterException` if invalid object passed
- 0 < `data.length` < max int value (2 147 483 647)
- Most negative `double` value < `abs(data[i])` < Most positive `double` value

Therefore the equivalence classes and boundary cases were determined to be:

- *Boundary case*: `data` = NULL
- *Boundary case*: `data` length = 0
- *Boundary case*: `data` length = Integer.MAX_VALUE
- *Equivalence class*: `data` contains values: Double.MAX_VALUE
  - Double.MAX_VALUE
- *Equivalence class*: `data` contains values from (Double.MAX_VALUE,
  - Double.MAX_VALUE)
- *Equivalence class*: `data` contains NULL

2.2.4 `createNumberArray2D(`*`double [][] data`*`)`:

Method accepts an array of `double` primitives, and should construct and return an array of `Number` objects from the provided arguments.
Assumption: Dimensions of the array labeled data is m ✕ n where m, n are integers
The input is constrained as follows:

- `data` != null -- in both m and n dimension OR in one dimension
- 0 < `data.length (both dimensions)` < max int value (2 147 483 647)
- Most negative `double` value < `abs(data[i][j])` < Most positive `double` value

Therefore the equivalence classes and boundary cases were determined to be:

- *Boundary case*: `data` = NULL
- *Boundary case*: `data` length = 0 -- in both dimensions AND in one dimension
- *Boundary case*: `data` length = Integer.MAX_VALUE
  -- in both m and n dimension AND in one dimension

- *Equivalence class*: `data` contains values: Double.MAX_VALUE
  - Double.MAX_VALUE
- *Equivalence class*: `data` contains values from (Double.MAX_VALUE,
  - Double.MAX_VALUE)
- *Equivalence class*: `data` contains NULL -- in both, m and n, dimension AND in one dimension

2.2.5 `calculateCumulativePercentages(`*`KeyedValues data`*`):`

Method accepts a `KeyedValues` object, and should return an instance of `KeyedValues` that contains the cumulative percentage values for the data provided in the argument.

`KeyedValues` object can have the variable length, L, which is an integer and is greater than 0. Assuming that there are L keys and L values within the instance, we have a domain from values: x_1, x_2, … , $x_{L.}$ Therefore the equivalence classes and boundary cases were determined to be:

- Equivalence case: $x_i \geq 0$
- Equivalence case: $x_i < 0$
- Equivalence case: L > 0
- Equivalence case: L=0, or object null
-

3. Test Cases

3.1 Range

3.1.1 `contains(double value):`

3.1.1.1 containsBLB()

- *Partition Covered:* value = lower bound - 1

Range Values	***Output (Expected)***	***Output (Actual)***
(0,0)	FALSE	FALSE
(-1,1)	FALSE	FALSE
(1,-1)	ERROR	IllegalArgumentException
(-Double,MAX_VALUE, 0)	FALSE	AssertionError
(- Double.MAX_VALUE, Double.MAX_VALUE)	FALSE	AssertionError
(NULL, NULL)	ERROR	Can't test

3.1.1.2 containsLB()

- *Partition Covered:* value = lower bound

Range Values	***Output (Expected)***	***Output (Actual)***
(0,0)	TRUE	TRUE
(-1,1)	TRUE	TRUE
(1,-1)	ERROR	IllegalArgumentException
(-Double,MAX_VALUE, 0)	TRUE	TRUE
(- Double.MAX_VALUE, Double.MAX_VALUE)	TRUE	TRUE
(NULL, NULL)	ERROR	Can't test

3.1.1.3 containsAUB()

- *Partition Covered:* value = upper bound + 1

Range Values	***Output (Expected)***	***Output (Actual)***
(0,0)	FALSE	FALSE
(-1,1)	FALSE	FALSE
(1,-1)	ERROR	IllegalArgumentException
(-Double,MAX_VALUE, 0)	FALSE	FALSE
(- Double.MAX_VALUE, Double.MAX_VALUE)	FALSE	AssertionError
(NULL, NULL)	ERROR	Can't test

3.1.1.4 containsUB()

- *Partition Covered:* value = upper bound

Range Values	***Output (Expected)***	***Output (Actual)***
(0,0)	TRUE	TRUE
(-1,1)	TRUE	TRUE
(1,-1)	ERROR	IllegalArgumentException
(-Double,MAX_VALUE, 0)	TRUE	TRUE
(- Double.MAX_VALUE, Double.MAX_VALUE)	TRUE	TRUE
(NULL, NULL)	ERROR	Can't test

3.1.1.5 containsCentral()

- *Partition Covered:* value = (lower+upper) / 2

Range Values	***Output (Expected)***	***Output (Actual)***
(0,0)	TRUE	TRUE
(-1,1)	TRUE	TRUE
(1,-1)	ERROR	IllegalArgumentException
(-Double,MAX_VALUE, 0)	TRUE	TRUE
(- Double.MAX_VALUE, Double.MAX_VALUE)	TRUE	TRUE
(NULL, NULL)	ERROR	Can't test

3.1.2 constrain(*double value*):

3.1.2.1 constrainsBLB()

- *Partition Covered:* value = lower bound - 1

Range Values	***Output (Expected)***	***Output (Actual)***
(0,0)	0	0
(-1,1)	-1	0
(1,-1)	ERROR	IllegalArgumentException
(-Double,MAX_VALUE, 0)	Not specified	- Double.MAX_VALUE
(- Double.MAX_VALUE, Double.MAX_VALUE)	Not specified	- Double.MAX_VALUE
(NULL, NULL)	ERROR	Can't test

3.1.2.2 constrainsLB()

- *Partition Covered:* value = lower bound

Range Values	***Output (Expected)***	***Output (Actual)***
(0,0)	0	0
(-1,1)	-1	-1
(1,-1)	ERROR	IllegalArgumentException
(-Double,MAX_VALUE, 0)	- Double.MAX_VALUE	- Double.MAX_VALUE
(- Double.MAX_VALUE, Double.MAX_VALUE)	- Double.MAX_VALUE	- Double.MAX_VALUE
(NULL, NULL)	ERROR	Can't test

3.1.2.3 constrainsAUB()

- *Partition Covered:* value = upper bound + 1

Range Values	***Output (Expected)***	***Output (Actual)***
(0,0)	0	0
(-1,1)	1	1
(1,-1)	ERROR	IllegalArgumentException
(-Double,MAX_VALUE, 0)	0	0
(- Double.MAX_VALUE, Double.MAX_VALUE)	Not specified	Double.MAX_VALUE
(NULL, NULL)	ERROR	

3.1.2.4 constrainsUB()

- *Partition Covered:* value = upper bound

Range Values	***Output (Expected)***	***Output (Actual)***
(0,0)	0	0
(-1,1)	1	1
(1,-1)	ERROR	IllegalArgumentException
(-Double,MAX_VALUE, 0)	0	0
(- Double.MAX_VALUE, Double.MAX_VALUE)	Not Specified	Double.MAX_VALUE
(NULL, NULL)	ERROR	Can't test

3.1.2.5 constrainsCentral()

- *Partition Covered:* value = (lower+upper) / 2

Range Values	***Output (Expected)***	***Output (Actual)***
(0,0)	0	0
(-1,1)	0	0
(1,-1)	ERROR	IllegalArgumentException
(-Double,MAX_VALUE, 0)	8.988465674311579E307	8.988465674311579E307
(- Double.MAX_VALUE, Double.MAX_VALUE)	0	0
(NULL, NULL)	ERROR	Can't test

3.1.3 getLength():

3.1.3.1 getLengthTest()

- *Partition Covered:* No inputs

Range Values	***Output (Expected)***	***Output (Actual)***
(0,0)	0	0
(-1,1)	2	-1
(1,-1)	ERROR	IllegalArgumentException
(-Double,MAX_VALUE, 0)	Double.MAX_VALUE	-Double.MAX_VALUE
(- Double.MAX_VALUE, Double.MAX_VALUE)	ERROR	-Double.MAX_VALUE
(NULL, NULL)	ERROR	Unable to test

3.1.4 getUpperBounds():

3.1.4.1 getUpperBoundsTest()

- *Partition Covered:* No inputs

Range Values	***Output (Expected)***	***Output (Actual)***
(0,0)	0	0
(-1,1)	1	-1
(1,-1)	ERROR	IllegalArgumentException
(-Double,MAX_VALUE, 0)	0	- Double.MAX_VALUE
(- Double.MAX_VALUE, Double.MAX_VALUE)	Double.MAX_VALUE	- Double.MAX_VALUE
(NULL, NULL)	ERROR	Unable to test

3.1.5 getLowerBounds():

3.1.5.1 getLowerBoundsTest()

- *Partition Covered:* No inputs

Range Values	***Output (Expected)***	***Output (Actual)***
(0,0)	0	0
(-1,1)	-1	-1
(1,-1)	ERROR	IllegalArgumentException
(-Double,MAX_VALUE, 0)	- Double.MAX_VALUE	- Double.MAX_VALUE
(- Double.MAX_VALUE, Double.MAX_VALUE)	- Double.MAX_VALUE	- Double.MAX_VALUE
(NULL, NULL)	ERROR	Unable to test

3.2 DataUtilities

3.2.1 calculateColumnTotal(*Values2D data, int column*):

2 Variables: Weak ECT
Equivalence Classes for data: All Positive, Mix of Positive and negative, high-precision values, null cells, null
Equivalence Classes for Column: Expected, Negative, Positive (doesn't exist), Very high
Boundary Cases for Column: 0

3.2.1.1 calculateColumnTotalOneNullValue()

- *Partition Covered:* data: null cells column: 0
- *Input*: {(null, 5)}, 0
- *Output*:
  - Expected: ERROR [Behaviour is not defined for this input]
  - Actual: 6

3.2.1.2 calculateColumnTotalWithPosAndNegDataValuesHighPostiveRowValue()

- *Partition Covered:* data: Mix of pos and neg column: Very high
- *Input*: {(2.5, -3.5)}, 1000000
- *Output*:
  - Expected: -1.0
  - Actual: -1.0

3.2.1.3 calculateColumnTotalForTwoPositiveValuesNegativeRowValue()

- *Partition Covered:* data: All Positive column: Negative
- *Input*: {(3.5, 2.5)}, -1
- *Output*:
  - Expected: ERROR: IndexOutofBoundsException
  - Actual: ERROR: IndexOutofBoundsException

3.2.1.4 calculateColumnTotalForTwoHighPrecisionDoubles()

- *Partition Covered:* data: High precision values column: 0
- *Input*: {(2.000000001, 2.000000001)}, 0
- *Output*:
  - Expected: 4.000000002
  - Actual: 4.000000002

3.2.1.5 calculateColumnTotalForNullValues2D()

- *Partition Covered:* data: null column: 0
- *Input*: null, 0
- *Output*:
  - Expected: ERROR: InvalidParameterException
  - Actual: ERROR: NullPointerException

3.2.2 `calculateRowTotal(`*`Values2D data, int row`*`)`:

2 Variables: Weak ECT
Equivalence Classes for data: All Positive, Mix of Positive and negative, high-precision values, null cells, null
Equivalence Classes for row: Expected, Negative, Positive (doesn't exist), Very high
Boundary Cases for row: 0

3.2.2.1 calculateRowTotalOneNullValue()

- *Partition Covered:* data: null cells row: 0
- *Input*: {(null), (5)}, 0
- *Output*:
  - Expected: ERROR [Behaviour is not defined for this input]
  - Actual: 0

3.2.2.2 calculateRowTotalWithPosAndNegDataValuesHighPostiveRowValue()

- *Partition Covered:* data: Mix of pos and neg Column: Very high
- *Input*: {(-3.5), (2.5)}, INT_MAX
- *Output*:
  - Expected: -1.0
  - Actual: -3.5

3.2.2.3 calculateRowTotalForTwoPositiveValuesNegativeRowValue()

- *Partition Covered:* data: All Positive row: Negative
- *Input*: {(3.5), (2.5)}, -1
- *Output*:
  - Expected: ERROR: IndexOutofBoundsException
  - Actual: ERROR: IndexOutofBoundsException

3.2.2.4 calculateRowTotalForTwoHighPrecisionDoubles()

- *Partition Covered:* data: High precision values row: 0
- *Input*: {(2.000000001), (2.000000001)}, 0
- *Output*:
  - Expected: 4.000000002
  - Actual: 2.000000001

3.2.2.5 calculateRowTotalForNullValues2D()

- *Partition Covered:* data: null row: 0
- *Input*: null, 0
- *Output*:
  - Expected: ERROR: InvalidParameterException
  - Actual: ERROR: NullPointerException

3.2.2.6 calculateRowTotalForTwoPositiveValuesZeroRowValue()

- *Partition Covered:* data: all positive row: 0
- *Input*: {(3.5), (2.5)}, 0
- *Output*:
    - Expected: 5.0
    - Actual: 3.5

`3.2.3 createNumberArray(double [] data):`

Equivalence Classes for double[] testInput**:** All Positive, Mix of Positive and negative, Zeros, all negatives, null

3.2.3.1 createNumberArrayBasic()

- *Partition Covered:* testInput: all positive
- *Input*: {2.0, 3.0}
- *Output*:
    - Expected: {2.0, 3.0}
    - Actual: {2.0, null}

3.2.3.2 createNumberArrayNullInput()

- *Partition Covered:* testInput: null
- *Input*: {}
- *Output*:
    - Expected: {}
    - Actual: {}

3.2.3.3 createNumberArrayZeros()

- *Partition Covered:* testInput: zeros
- *Input*: { 0.0, 0.0 }
- *Output*:
    - Expected: { 0.0, 0.0 }
    - Actual: { 0.0, null }

3.2.3.4 createNumberArrayZeros()

- *Partition Covered:* testInput: single zero
- *Input*: { 0.0 }
- *Output*:
    - Expected: { 0.0 }
    - Actual: {null }

3.2.3.5 createNumberArraySomeNegative()

- *Partition Covered:* testInput: some negative
- *Input*: { -10.0, 3.0, -2.0, 9.0 }
- *Output*:

  - Expected: { -10.0, 3.0, -2.0, 9.0 }
  - Actual: { -10.0, 3.0, -2.0, null }

3.2.3.6 createNumberArrayNegative()

- *Partition Covered:* testInput: negative
- *Input*: { -10.0 }
- *Output*:
  - Expected: { -10.0 }
  - Actual: {null }

3.2.4 `createNumberArray2D(`*`double [][] data`*`):`

Equivalence Classes for double[][] testInput**:** All Positive, Mix of Positive and negative, Zeros, all negatives, null

3.2.4.1 createNumberArray2DBasic()

- *Partition Covered:* testInput : positive
- *Input*: { { 2.0, 3.0 }, { 4.0, 5.0 } }
- *Output*:
  - Expected: { { 2.0, 3.0 }, { 4.0, 5.0 } }
  - Actual: { { 2.0, null }, { 4.0, null } }

3.2.4.2 createNumberArray2DNegative()

- *Partition Covered:* testInput : negative
- *Input*: { { -2.0, -3.0 }, { -4.0, -5.0 }, { -6.0, -7.0 } }
- *Output*:
  - Expected: { { -2.0, -3.0 }, { -4.0, -5.0 }, { -6.0, -7.0 } }
  - Actual: { { -2.0, null }, { -4.0, null }, { -6.0,null } }

3.2.4.3 createNumberArray2DSomeNegative()

- *Partition Covered:* testInput : some negative
- *Input*: { { -2.0, 3.0 }, { -4.0, -5.0 }, { 6.0, 7.0 } }
- *Output*:
  - Expected: { { -2.0, 3.0 }, { -4.0, -5.0 }, { 6.0, 7.0 } }
  - Actual: { { -2.0, null }, { -4.0, null }, { 6.0,null } }

3.2.4.4 createNumberArray2DZeros()

- *Partition Covered:* testInput : zeros
- *Input*: { { 0.0, 0.0 }, { 0.0, 0.0 } }
- *Output*:
  - Expected: { { 0.0, 0.0 }, { 0.0, 0.0 } }
  - Actual: { { 0.0, null}, { 0.0, null } }

3.2.4.5 createNumberArray2DOneZero()

- *Partition Covered:* testInput : zero
- *Input*: { { 0.0 } }

- *Output*:
    - Expected: { { 0.0 } }
    - Actual: { { null } }

3.2.4.6 createNumberArray2DNull()

- *Partition Covered:* testInput : null
- *Input*: { {} }
- *Output*:
    - Expected: { {} }
    - Actual: { {} }

3.2.5 calculateCumulativePercentages(*KeyedValues data*):

Equivalence Classes for DefaultKeyedValues input:All Positive, Mix of Positive and negative, all negative, Zeros, null

3.2.5.1 getCumulativePercentageTestEmptyList()

- *Partition Covered:* input: null
- *Input*: new DefaultKeyedValues()
- *Output*:
    - Expected: empty
    - Actual: empty

3.2.5.2 getCumulativePercentageTestNegativeValues()

- *Partition Covered:* input: negative
- *Input*: (0, 0.0), (1,-1), (2,-4)
- *Output*:
    - Expected: (0, 0.0), (1,.2), (2,1)
    - Actual:(0, -0.0), (1,.2), (2,1)

3.2.5.3 getCumulativePercentageTestExpectedValues()

- *Partition Covered:* input: positive
- *Input*: (0, 5), (1,9), (2,2)
- *Output*:
    - Expected:(0, 0.3125), (1,0.875), (2,1)
    - Actual:(0, .4545454545), (1,1.272727), (2,1.45454545)

4. Division of Teamwork

As briefly mentioned in the introduction, the group attempt to divide work as evenly as possible. Each member took one method from the Range and DataUtilities class, however, with 4 members, 2 individuals were required to test an extra method. This also ensured that the team would meet the requirement set out by the lab assignment during the Demo phase.

Following the steps discussed in class, the team determined partitions and corresponding test cases for each method, along with any special cases. Based on the documentation provided, we were attempting to find worst-case scenarios that would result in errors in the code. A skill, which some team members are more adept to than other, thus, this phase in planning was conducted as a group.

The objective of this lab was to provide individuals with hands-on experience with JUnit ***and*** jMock. Therefore, depending on the methods selected some individuals may have not required mocking in their testing. Therefore, peer-programing was implemented on one partition for one method to ensure each member was able to grasp the concept. Especially, with the Demo requirement stating each member should be able to demo mocking, even if they did not write the code for that.

Once each member finished their testing, the team re-grouped in order to evaluate each others work. Feedback was provided in order to increase the robustness of test cases, and identify any missing test cases.

5. Difficulties Encountered, Challenges Overcome, and Lessons Learned

This lab proved an effective method in which to reinforce the content discussed in lecture. However, it should be noted that a few difficulties were encountered during the process of this lab. During the "Familiarization" portion listed in the lab assignment, specifically during section "2.1.2 Add the Necessary Java Libraries," a fatal error was encountered that made it impossible to complete the lab. This afflicted 3 of the 4 team members. The majority, if not the entirety, of a complete lab session was spent trying to correct the error even with the assistance of TAs. A solution was eventually found, although one may consider it rather unconventional. Although not being able to properly run code was a hindrance, the real issue was the wasted time it caused. When each member has busy schedules, it becomes difficult to coordinate times to work on team assignments, for sessions greater than 30 minutes. Although, online communication exists, sometimes face-to-face discussion is necessary to clarify concepts or questions. However, this assignment has been completed as proven by the submission.

Amongst one of the difficulties encountered, arose from the fact that the program tested is ran from jar files. This lab specifically requests the block-box approach to testing, therefore, it makes sense that we are unable to view the actual code we are testing. However, it makes it difficult to determine whether an error encountered is based of the constructors or the method itself being tested. Therefore, reasonable assumptions were made under these circumstances.

6. Lab Feedback

Ensuring the lab is capable of running on a variety of different systems and IDEs
Referring to the issues that arose from from the Familiarization section, discussed in the previous section, If this were to happen in future version, the required step to achieve success were as follows:

1. Delete any previously existing project associated with this assignment (i.e. Delete projects title JFreeChart)
2. Create a new project, and incorrectly title it (i.e. JFreeChrt)
3. Ensure that the system is running in Java 1.8
4. Include the necessary, and correct, jar files
5. Attempt to run the program -- Should fail due to the incorrect project name
6. Refactor the project to have the correct project name (i.e. JFreeChart)
7. Re-attempt to run the project as specified by the lab document and it should work

Note: Other solutions may be found, however, these steps provided the most consistent results

Many tools are available to assist programmers in their coding and as a result of these options, people have preferences. However, this lab disregarded these potential preferences and restricted the use of other Development Environments. One team member who attempted to run the lab on intelliJ, encountered a variety of different errors. Several hours were spent trying to successfully run the program, with TA assistance, to no avail. As a result, the individual eventually ran the lab on Eclipse successfully after sorting out the errors associated with Eclipse. However, this left the team member having to learn a new environment in addition to the new tools introduced in the lab.

Repetitive tasks
There is something to be said towards repeating tasks in order to reinforce concepts associated with it. In requesting that teams provide the entire list of test cases design, it ensures that you are understanding what is being tested and how the function is being tested. It also increases the traceability of the test cases. However, some occasions it seemed a bit excessive, especially reviewing the number of pages in this report that are discussing test cases.

Providing a smaller sample set may be a reasonable solution, rather than wearing out the Ctrl+C buttons on our laptops.

Trivial Methods
Providing the limit of 5 methods from Range to reduce the workload is appreciated. However, some methods selected proved themselves to be trivial tests due to the methods lack of complexity. For example, the getLength(), getUpperBounds(), and getLowerBounds() were getter methods, therefore there was nothing in particular to test.