Group 37

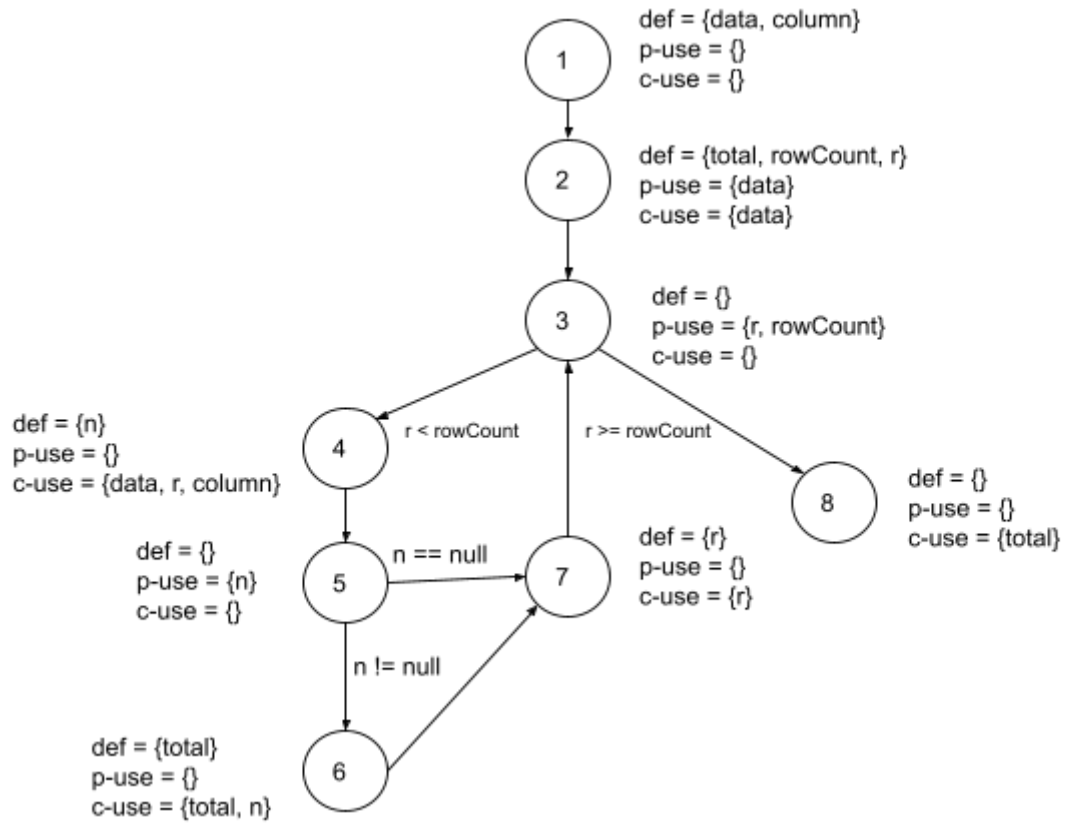
Student names:
Dominic Vandekerkhove
Alexander Varga
Ivan Tompong
John Cedric Acierto

Introduction

This lab provides an introduction to utilizing control flow and data flow coverage criteria to measure test suite adequacy. The system under test is JFreeChart, a program that simulates and displays various graphs and charts. We will be using the EclEmma code coverage tool (maybe more) to measure the adequacy of the test suite we designed in lab 2. The metrics we are measuring are statement, branch and method coverage. We will also be designing flow graphs and calculating the DU-pair coverage for 2 methods. We will use this information, along with the system's source code to generate an improved test suite.

Manual data-flow coverage calculations for X and Y methods

DataUtilities.calculateColumnTotal
Data flow graph



Def-use sets

Node (n)	def(n)	p-use(n)	c-use(n)
1	{data, column}	{}	{}
2	{total, rowCount, r}	{data}	{data}
3	{}	{r, rowCount}	{}
4	{n}	{n}	{data, r, column}
5	{}	{}	{}
6	{total}	{}	{total, n}
7	{r}	{}	{r}
8	{}	{}	{total}

All DU pairs

Variable	DU-pairs
data	(1,2) , (1, 4)
column	(1, 4)
total	(2,6) , (6, 8) , (2, 8)
rowCount	(2, 3)
r	(2, 3) , (2, 7) , (7, 3)
n	(4, 5) , (4, 6)

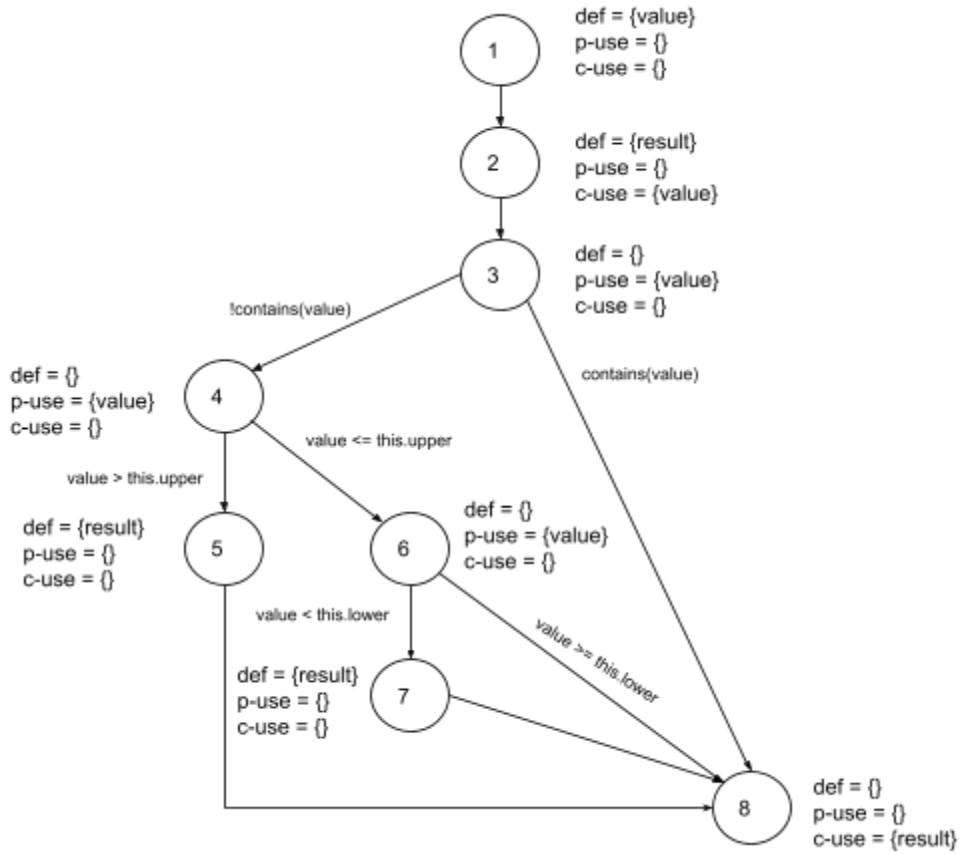
Which pairs are covered per test case

Test	DU-pairs Covered
testCalculateColumnTotalForTwoValue	All DU pairs except (2, 8, total)
testCalculateColumnTotalForTwoValueFalse	All DU pairs except (2, 8, total)
testCalculateColumnTotalNullInput	Throws exception at 2
testCalculateColumnTotalInvalidInput	Throws exception at 2

DU pair coverage = Covered DU pairs / All DU pairs = 11 / 12 = 0.9167 = **91.67%**

Range.constrain

Data flow graph



Def-use sets

Node (n)	def(n)	p-use(n)	c-use(n)
1	{value}	{}	{}
2	{result}	{}	{value}
3	{}	{value}	{}
4	{}	{value}	{}
5	{result}	{}	{}
6	{}	{value}	{}
7	{result}	{}	{}
8	{}	{}	{result}

All DU pairs

Variable	DU-pairs
value	(1,2) , (1,3) , (1,4), (1,6)
result	(2,8) , (5,8) , (7,8)

Which pairs are covered per test case

Test	DU-pairs Covered
testConstrainLessThanLowerBound	(1,2, value) , (1,3, value) , (1,4, value) , (1,6, value) , (7, 8, result)
testConstrainAtLowerBound	(1,2, value) , (1,3, value) , (1,4, value) , (1,6, value) , (2, 8, result)
testConstrainWithinRange	(1,2, value) , (1,3, value) , (1,4, value) , (1,6, value) , (2, 8, result)
testConstrainAtUpperBound	(1,2, value) , (1,3, value) , (1,4, value) , (1,6, value) , (2, 8, result)
testConstrainMoreThanUpperBound	(1,2, value) , (1,3, value) , (1,4, value) , (1,6, value) , (5, 8, result)

DU pair coverage = Covered DU pairs / All DU pairs = 7 / 7 = 1 = **100%**

A detailed description of the testing strategy for the new unit test

Our new tests were designed by analyzing the coverage of our original test suite in assignment 2. We noticed that the coverage was low for the two classes because we had only written tests for a few of the methods. So, our primary aim was to write tests for the rest of the methods in the Range and DataUtilities classes. To do this, we reviewed the method requirements in the provided documentation and established boundaries and equivalent classes similar to assignment 2. Once our tests were written, we measured the coverage again to ensure our test suite met our expectations. If any method had low coverage after tests were written for it, we reviewed the code using the EclEmma coverage tool to see which statements were not being covered. Then, we wrote additional tests to cover these situations.

Tests[1-3] Scale: The documentation details that this method scales (multiplies) the given range by a factor. Since it requires that the factor is non-negative, tests were written to cover a positive factor and a negative factor. These tests are called testPositiveScale and testNegativeScale respectively. A test was also conducted where the range argument was null, this test is called testNullScale and expects an exception to be thrown.

Tests[4-6] Shift: This method is intended to shift a given range by a certain factor. Tests were conducted on a positive factor, a negative factor, and a factor of 0 to cover all equivalent classes. These tests are called testPositiveShift, testNegativeShift and testZeroShift respectively.

Tests[7-13] Shift (overloaded): This method has the same functionality as the method above with the added option of allowing the new range to cross zero. So, tests were conducted with positive and negative shift values and a shift value of 0 while allowing the condition, and not allowing the condition. These tests are called testPositiveShiftWithZeroCross, testNegativeShiftWithZeroCross and testZeroShiftWithZeroCross for the tests allowing the zero cross. The other tests are testPositiveShiftWithoutZeroCross, testNegativeShiftWithoutZeroCross and testZeroShiftWithoutZeroCross. After executing the code coverage tool, it was noticed that a branch was not being covered. By inspection, this branch had to be covered by tests with a Range of (0,0). So, this was conducted by a test called testZeroShiftZeroRange.

Test[45] getLength(): This method returns the length of a given Range instance. One test was conducted on this method which achieves 100% statement, branch and method coverage (EclEmma). This test simply instantiates a Range object with a range of [0,10] and calls the get length function which returns the length into a double to then be asserted for the equal or expected value.

Test[46] toString(): This method returns the given range of a Range instance as a string in the format “Range[lower_bound,upper_bound]”. One test was conducted on this method which achieves 100% statement, branch and method coverage (EclEmma). This test simply

instantiates a Range object with a range of [0,10] and calls the toString function which returns a string which is then compared in an equals statement with the expected string constant as the condition in an assertTrue.

Tests[47-49] CalculateColumnTotal (overloaded): This method is intended to calculate the sum of values in a given column on selected rows. Tests were conducted where all rows are selected, no rows are selected, and some rows are selected. These tests are called testCalculateColumnTotalAll, testCalculateColumnTotalNone and testCalculateColumnTotal2.

Test[14-16] isNaNRange: This method is supposed to check if the current range contained within the Range object is NaN (Not a Number) or not. The test cases that were written for this method revolve around the three different Range objects, one containing the range (-10, 10), one with either the upper or lower bound is NaN, and one where both bounds are NaN.
We only need either the upper or lower bound to be NaN because this will achieve 100% line and branch coverage for the method

Test[22-34, 41-43]: combine(Range range1, Range range2) and combineIgnoringNaN(Range range1, Range range2): These two methods are functionally the same and return the same thing, a combination of the two ranges passed in as arguments. The difference between the two functions is that one function combines the two ranges passed in regardless of whether or not either or both of the ranges contain a NaN. The test cases for this function are meant to check if the method can return the correct output if there are no NaN's present and when there is a single or multiple NaN's present.

Test No.	Class	Function Under Test	Input Variables	Expected Outcome	Actual Outcome	Result
1	Range	scale(Range base, double factor)	base = (-1, 1) Factor = 10	Returns range (-10, 10)	Returns range (-10, 10)	PASS
2	Range	scale(Range base, double factor)	base = (-1, 1) Factor = -10	Throws illegal argument exception	Throws illegal argument exception	PASS
3	Range	scale(Range base, double factor)	base = NULL Factor = 1	Throws illegal argument exception	Throws illegal argument exception	PASS
4	Range	shift(Range base, double delta)	base = (-1, 1) Delta = 5	Returns range (4, 6)	Returns (0, 0)	FAIL
5	Range	shift(Range base, double delta)	base = (-1, 1) Delta = -5	Returns range (-6, -4)	Returns (0, 0)	FAIL
6	Range	shift(range base, double	Base = (-1, 1)	Returns range (-1, 1)	Returns (-1, 1)	PASS

		delta)	Delta = 0			
7	Range	shift(Range base, double delta, boolean allowZeroCrossing)	Base = (-1, 1) Delta = 5 allowZero Crossing = true	Returns range (4, 6)	Returns (4, 6)	PASS
8	Range	shift(Range base, double delta, boolean allowZeroCrossing)	Base = (-1, 1) Delta = -5 allowZero Crossing = true	Returns range (-6, -4)	Returns (-6, -4)	PASS
9	Range	shift(Range base, double delta, boolean allowZeroCrossing)	Base = (-1, 1) Delta = 0 allowZero Crossing = true	Returns range (-1, 1)	Returns (-1, 1)	PASS
10	Range	shift(Range base, double delta, boolean allowZeroCrossing)	Base = (-1, 1) Delta = 5 allowZero Crossing = false	Returns range (0, 6)	Returns (0, 6)	PASS
11	Range	shift(Range base, double delta, boolean allowZeroCrossing)	Base = (-1, 1) Delta = -5 allowZero Crossing = false	Returns range (-6, 0)	Returns (-6, 0)	PASS
12	Range	shift(Range base, double delta, boolean allowZeroCrossing)	Base = (-1, 1) Delta = 0 allowZero Crossing = false	Returns range (-1, 1)	Returns (-1, 1)	PASS
13	Range	shift(Range base, double delta, boolean allowZeroCrossing)	Base = (0, 0) Delta = 0 allowZero Crossing = false	Returns range (0, 0)	Returns (0, 0)	PASS
14	Range	isNaNRange()	None for isNaNRange() <u>Existing Range Object:</u> lower = -10 upper = 10	Returns false	Returns false	Pass
15	Range	isNaNRange()	None for isNaNRange() <u>Existing Range Object:</u> lower = NaN	Returns true	Returns true	Pass

			upper = NaN			
16	Range	isNaNRange()	None for isNaNRange() Existing Range Object: lower = NaN upper = 10	Return true	Return false	Fail
17	Range	intersects(double b0, double b1)	intersects() b0 = -15 b1 = 15 Existing Range Object: lower = -10 upper = 10	Return false	Return false	PASS
18	Range	intersects(double b0, double b1)	intersects() b0 = -5 b1 = -15 Existing Range Object: lower = -10 upper = 10	Return false	Return false	PASS
19	Range	intersects(double b0, double b1)	intersects() b0 = 20 b1 = 15 Existing Range Object: lower = -10 upper = 10	Return false	Return false	PASS
20	Range	intersects(Range range)	Input Range Object: lower = -5 upper = 5 Existing Range Object: lower = -10 upper = 10	Return true	Return true	PASS
21	Range	intersects(Range range)	Input Range Object: lower = 20	Return false	Return false	PASS

			upper = 30 Existing Range Object: lower = -10 upper = 10			
22	Range	combine(Range range1, Range range2)	range1 = null range2: lower = -10 upper = 10	Returns range2	Returns range2	PASS
23	Range	combine(Range range1, Range range2)	range1: lower = -10 upper = 10 range2 = null	Return range1	Returns range1	PASS
24	Range	combine(Range range1, Range range2)	range1: lower = -10 upper = 10 range2: lower = 15 upper = 30	Returns new Range(-10, 30)	Returns new Range(-10, 30)	PASS
25	Range	combineIgnoringNaN(Range range1, Range range2)	range1: lower = NaN upper = NaN range2: lower = -10 upper = 10	Returns (NaN, 10)	Returns (-10, 10)	FAIL
26	Range	combineIgnoringNaN(Range range1, Range range2)	range1: lower = 15 upper = 30 range2: lower = NaN upper = NaN	Returns (NaN, 30)	Returns (15, 30)	FAIL
27	Range	combineIgnoringNaN(Range range1, Range range2)	range1: lower = NaN upper = NaN range2: lower = NaN upper = NaN	Returns (NaN, NaN)	Returns (NaN, NaN)	PASS

28	Range	combineIgnoringNaN(Range range1, Range range2)	range1: lower = -10 upper = 10 range2: lower = 15 upper = 30	Returns (-10, 30)	Returns (-10, 30)	PASS
29	Range	combineIgnoringNaN(Range range1, Range range2)	range1 = null range2: lower = 15 upper = 30	Return (15, 30)	Return (15, 30)	PASS
30	Range	combineIgnoringNaN(Range range1, Range range2)	range2 = null range1: lower = 15 upper = 30	Return (15, 30)	Return (15, 30)	PASS
31	Range	combineIgnoringNaN(Range range1, Range range2)	range1 = null range2: lower = NaN upper = NaN	Returns null	Returns null	PASS
32	Range	combineIgnoringNaN(Range range1, Range range2)	range2 = null range1: lower = NaN upper = NaN	Returns null	Returns null	PASS
33	Range	combineIgnoringNaN(Range range1, Range range2)	range1 = null range2 = null	Returns null	Returns null	PASS
34	Range	combineIgnoringNaN(Range range1, Range range2)	range1: lower = -15 upper = NaN range2: lower = -10 upper = NaN	Returns (-15, NaN)	Returns (-15, NaN)	PASS (*shows a failure for some reason in the code)
35	Range	expand(Range range, double lowerMargin, double upperMargin)	range: lower = 10 upper = 20 lowerMargin =	Returns (5, 25)	Returns (5, 25)	PASS

			0.5 upperMargin = 0.5			
36	Range	expand(Range range, double lowerMargin, double upperMargin)	range: lower = 0 upper = 1 lowerMargin = -0.5 upperMargin = -0.7	Returns (0.4, 0.4)	Returns (0.4, 0.4)	PASS
37	Range	expandToInclude(Range range, double value)	range: null value = 1.0	Returns (1.0, 1.0)	Returns (1.0, 1.0)	PASS
38	Range	expandToInclude(Range range, double value)	range: lower = 2 upper = 3 value = 1.0	Returns (1.0, 3.0)	Returns (1.0, 3.0)	PASS
39	Range	expandToInclude(Range range, double value)	range: lower = 1 upper = 2 value = 3.0	Returns (1.0, 3.0)	Returns (1.0, 3.0)	PASS
40	Range	expandToInclude(Range range, double value)	range: lower = 3 upper = 3 value = 3	Returns (3.0, 3.0)	Returns (3.0, 3.0)	PASS
41	Range	combineIgnoringNaN(Range range1, Range range2) [Indirectly testing min/max]	range1: lower = NaN upper = 2 range2: lower = 2 upper = 3	Returns (2, 3)	Returns (2, 3)	PASS
42	Range	combineIgnoringNaN(Range range1, Range range2) [Indirectly testing min/max]	range1: lower = 2 upper = NaN range2: lower = 2 upper = 3	Returns (2,3)	Returns (2, 3)	PASS

43	Range	combineIgnoringNaN(Range range1, Range range2) [Indirectly testing min/max]	range1: lower = 1 upper = 2 range2: lower = 3 upper = 4	Returns (1,4)	Returns (1,4)	PASS
44	Range	hashCode()	range1: lower = 0 upper = 10 range2: lower = 0 upper = 10	Returns true	Returns true	PASS
45	Range	getLength()	Range1: Lower = 0 Upper = 10 Double range = range1.getLength()	Returns 10	Returns 10	PASS
46	Range	toString()	Range1: Lower = 0 Upper = 10 Double range = range1.toString()	Returns “Range[0,10]”	Returns “Range[0,10]”	PASS
47	DataUtilities	calculateColumnTotal(Values2D Data, int column, int[] validRows)	Values2D = {1}, {1}, {1} Column = 0 validRows = {0, 1, 2}	Returns 3	Returns 3	PASS
48	DataUtilities	calculateColumnTotal(Values2D Data, int column, int[] validRows)	Values2D = {1}, {1}, {1} Column = 0 validRows = {}	Returns 0	Returns 0	PASS
49	DataUtilities	calculateColumnTotal(Values2D Data, int column, int[] validRows)	Values2D = {1}, {1}, {1} Column = 0 validRows = {0, 2}	Returns 2	Returns 2	PASS
50	DataUtilities	equal(double[][]a, double[][]b)	a = { {1.0}, {1.0, 2.0}, {1.0, 2.0,	Returns true	Returns true	PASS

			3.0}, {1.0, 2.0, 3.0, 4.0} } b = { {1.0}, {1.0, 2.0}, {1.0, 2.0, 3.0}, {1.0, 2.0, 3.0, 4.0} }			
51	DataUtilities	equal(double[][]a, double[][]b)	a = null b = null	Returns true	Returns true	PASS
52	DataUtilities	equal(double[][]a, double[][]b)	a = null b = { {1.0}, {1.0, 2.0}, {1.0, 2.0, 3.0}, {1.0, 2.0, 3.0, 4.0} }	Returns false	Returns false	PASS
53	DataUtilities	equal(double[][]a, double[][]b)	a = { {1.0}, {1.0, 2.0}, {1.0, 2.0, 3.0}, {1.0, 2.0, 3.0, 4.0} } b = null	Returns false	Returns false	PASS
54	DataUtilities	equal(double[][]a, double[][]b)	a = { {1.0}, {1.0, 2.0}, {1.0, 2.0, 3.0}, {1.0, 2.0, 3.0, 4.0, 5.0} } b = { {1.0}, {1.0, 2.0}, {1.0, 2.0, 3.0}, {1.0, 2.0, 3.0, 4.0} }	Returns false	Returns false	PASS
55	DataUtilities	equal(double[][]a, double[][]b)	a = { {10.0}, {10.0, 2.0}, {10.0, 2.0, 3.0}, {10.0, 2.0, 3.0, 4.0, 5.0} } b = { {1.0}, {1.0, 2.0}, {1.0, 2.0, 3.0}, {1.0, 2.0, 3.0, 4.0} }	Returns false	Returns false	PASS
56	DataUtilities	clone(double[][] source)	source = { {1.0}, {1.0, 2.0}, {1.0, 2.0, 3.0}, {1.0, 2.0, 3.0, 4.0} }	Returns NotNull	Returns NotNull	PASS
57	DataUtilities	getCumulativePercentages(KeyedValues values)	values = Key: Value: 0 10	Returns Key: Value: 0 10/30	Returns Key: Value: 0 10/30	PASS

			1 10 2 10	1 20/30 2 30/30	1 20/30 2 30/30	
58	DataUtilities	getCumulativePercentages(KeyedValues values)	values = Key: Value: 0 10 1 null 2 10	Returns Key: Value: 0 10/20 1 10/20 2 20/20	Returns Key: Value: 0 10/20 1 10/20 2 20/20	PASS
59	DataUtilities	calculateRowTotal(Values2D data, int row, int[] validCols)	Data = Row Value: 0 35 1 8 2 90 3 45 4 5 5 10 Row = 0 validCols = {0, 2, 4, 5}	Return 140	Returns 140	PASS

A high level description of five selected test cases you have designed using coverage information, and how they have increased code coverage

Test[13] testZeroShiftWithZeroRange: Originally, this test was not included for the testing of this method. The tests were planned to cover equivalent classes for the shift value and the allowZeroCrossing option. When these tests were created and executed, it recorded 75% branch coverage for that method. So, we reviewed the method in the source code and noticed that it has conditions ensuring the range values are not 0. So, this additional test was created to cover this branch when the Range is (0,0). After executing, EclEmma reported that the method has 100% instruction, branch and method coverage.

Tests[50-55] equal(double[][]a, double[][] b): This method returns a boolean variable of whether or not the two double[][] arrays being compared are equal. For them to be considered equal, they must have the exact same dimensions and the elements must match, null permitted. Based on the documentation and observing the method and its potential coverage I wrote six test cases for the method. The six tests achieve 100% statement, branch and method coverage (EclEmma). The first test involves testing two arrays with the same length and elements expecting true, i.e. they are equal. The second test involves testing two null arrays where the expected result is true and covers a specific if branch. The third and fourth tests expect false as one tests a = null and b = valid array and the other a = valid array and b = null therefore covers two specific if branches. The fifth test involves testing two arrays with the same length but

different elements and expects false. The last test involves testing two arrays of different lengths and expects false. Together, the six test cases achieve 100% branch coverage on this method with multiple branches.

Test[56] clone(double[][] source): This method returns a clone instance of the source argument provided. One test was conducted on this method which achieves 100% statement, and method coverage (EclEmma) but only 75% branch coverage. This test initializes a double[][] array to be cloned which is then used as the argument in the clone function call that initializes a second double[][] array which is then asserted to be NotNull, i.e. the double array has been cloned, exists and is not a null object. The reason there is only 75% branch coverage is because within the clone function there is a if branch within a for loop iterating through the array that checks if the source array indexed element is null. I could cover the branch where the element was not null but to attempt to cover the branch where an element is null you must pass in an array that is either entirely null or has valid elements preceding a null element. However, in the case of an entirely null array, the clone function begins by calling ParamChecks.nullNotPermitted on the source array which will detect that the source array is null and therefore hardly any statement coverage and does not achieve our branch coverage goal. The second option is valid elements preceding null but java does not allow this therefore there is no way to achieve full branch coverage.

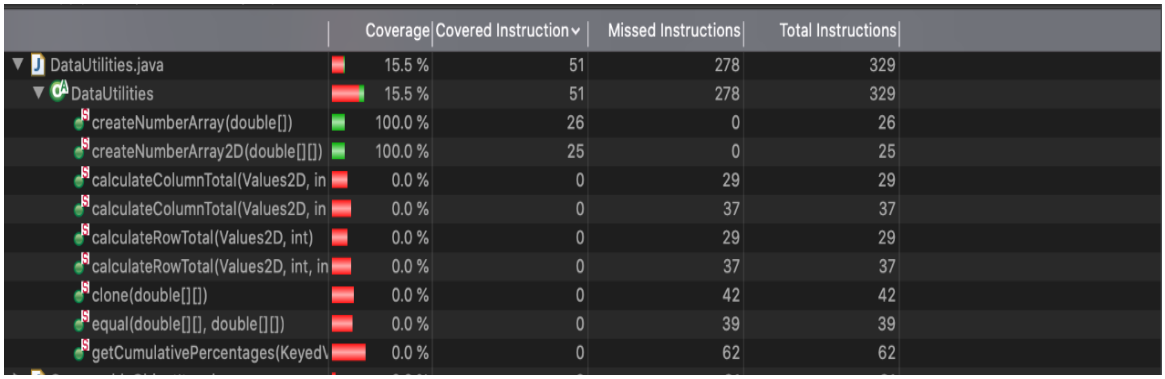
Test[36] testExpandLowerGreaterThanUpper: This method expands a Range object using the lower and upper margin arguments. Originally, this method did not have 100% statement and condition coverages because the (lower > upper) condition was not satisfied. After realizing that the margins did not have to be positive, both margin arguments were changed to negative to ensure that the *lower* variable would be greater than the *upper* variable. After doing this test, the coverage rate for all three types (condition, statement, and method) were up 100%.

Test[58] for getCumulativePercentages(KeyedValues values): This test case for getCumulativePercentages(KeyedValues values) is made to test if the method will function properly, if the KeyedValues object passed in contains a null value in one of its elements. This is a brand new test case that we would not have been able to write with black box testing alone since the documentation did not mention this specific behavior of the method. With access to the source code, we were able to determine and test this behavior, allowing us to reach over 90% line coverage and over 70% branch coverage for the method

A detailed report of the coverage achieved of each class and method (a screen shot from the code cover results in green and red color would suffice)

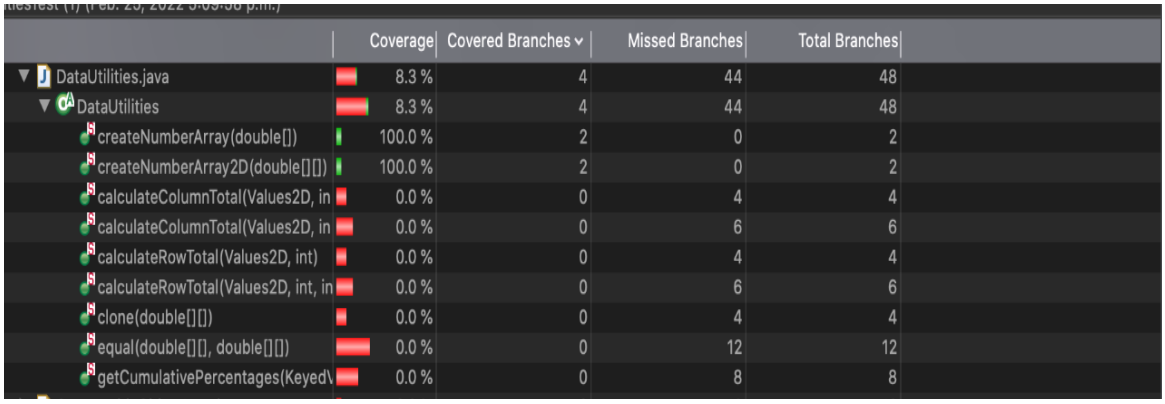
Original Test Suite

DataUtilities Statement Coverage



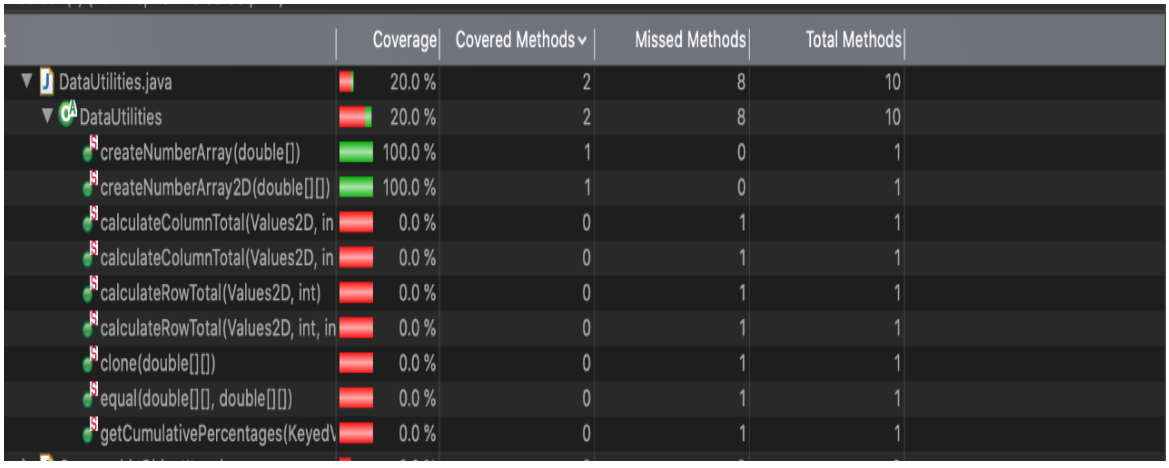
	Coverage	Covered Instruction	Missed Instructions	Total Instructions
DataUtilities.java	15.5 %	51	278	329
DataUtilities	15.5 %	51	278	329
createNumberArray(double[])	100.0 %	26	0	26
createNumberArray2D(double[][])	100.0 %	25	0	25
calculateColumnTotal(Values2D, in	0.0 %	0	29	29
calculateColumnTotal(Values2D, in	0.0 %	0	37	37
calculateRowTotal(Values2D, int)	0.0 %	0	29	29
calculateRowTotal(Values2D, int, in	0.0 %	0	37	37
clone(double[][])	0.0 %	0	42	42
equal(double[][], double[][])	0.0 %	0	39	39
getCumulativePercentages(Keyed\	0.0 %	0	62	62

DataUtilities Branch Coverage



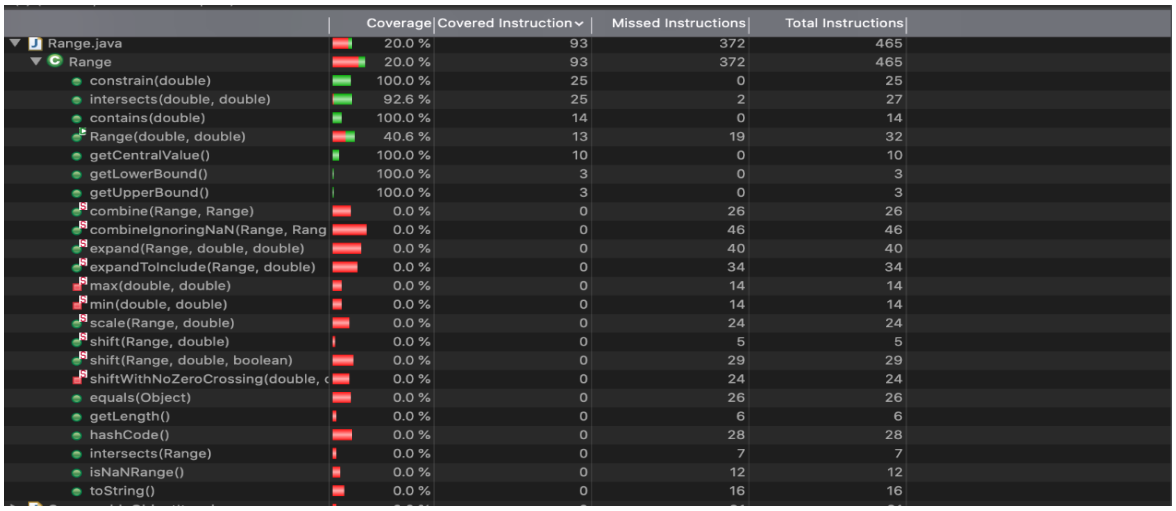
	Coverage	Covered Branches	Missed Branches	Total Branches
DataUtilities.java	8.3 %	4	44	48
DataUtilities	8.3 %	4	44	48
createNumberArray(double[])	100.0 %	2	0	2
createNumberArray2D(double[][])	100.0 %	2	0	2
calculateColumnTotal(Values2D, in	0.0 %	0	4	4
calculateColumnTotal(Values2D, in	0.0 %	0	6	6
calculateRowTotal(Values2D, int)	0.0 %	0	4	4
calculateRowTotal(Values2D, int, in	0.0 %	0	6	6
clone(double[][])	0.0 %	0	4	4
equal(double[][], double[][])	0.0 %	0	12	12
getCumulativePercentages(Keyed\	0.0 %	0	8	8

DataUtilities Method Coverage



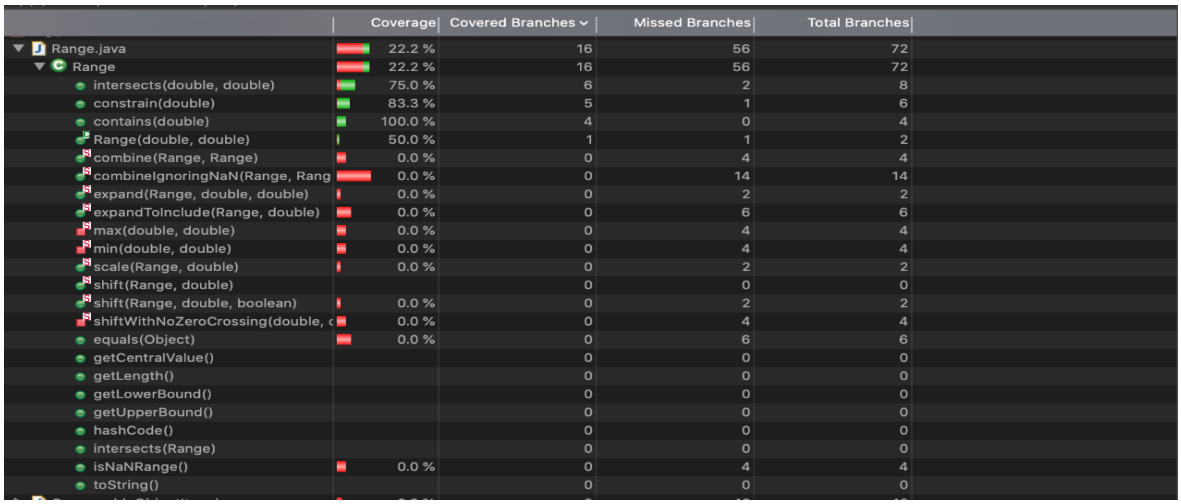
	Coverage	Covered Methods	Missed Methods	Total Methods
DataUtilities.java	20.0 %	2	8	10
DataUtilities	20.0 %	2	8	10
createNumberArray(double[])	100.0 %	1	0	1
createNumberArray2D(double[][])	100.0 %	1	0	1
calculateColumnTotal(Values2D, in	0.0 %	0	1	1
calculateColumnTotal(Values2D, in	0.0 %	0	1	1
calculateRowTotal(Values2D, int)	0.0 %	0	1	1
calculateRowTotal(Values2D, int, in	0.0 %	0	1	1
clone(double[][])	0.0 %	0	1	1
equal(double[][], double[][])	0.0 %	0	1	1
getCumulativePercentages(Keyed\	0.0 %	0	1	1

Range Statement Coverage



	Coverage	Covered Instruction	Missed Instructions	Total Instructions
Range.java	20.0 %	93	372	465
Range	20.0 %	93	372	465
constrain(double)	100.0 %	25	0	25
intersects(double, double)	92.6 %	25	2	27
contains(double)	100.0 %	14	0	14
Range(double, double)	40.6 %	13	19	32
getCentralValue()	100.0 %	10	0	10
getLowerBound()	100.0 %	3	0	3
getUpperBound()	100.0 %	3	0	3
combine(Range, Range)	0.0 %	0	26	26
combineIgnoringNaN(Range, Rang	0.0 %	0	46	46
expand(Range, double, double)	0.0 %	0	40	40
expandToInclude(Range, double)	0.0 %	0	34	34
max(double, double)	0.0 %	0	14	14
min(double, double)	0.0 %	0	14	14
scale(Range, double)	0.0 %	0	24	24
shift(Range, double)	0.0 %	0	5	5
shift(Range, double, boolean)	0.0 %	0	29	29
shiftWithNoZeroCrossing(double,	0.0 %	0	24	24
equals(Object)	0.0 %	0	26	26
getLength()	0.0 %	0	6	6
hashCode()	0.0 %	0	28	28
intersects(Range)	0.0 %	0	7	7
isNaNRange()	0.0 %	0	12	12
toString()	0.0 %	0	16	16

Range Branch Coverage



	Coverage	Covered Branches	Missed Branches	Total Branches
Range.java	22.2 %	16	56	72
Range	22.2 %	16	56	72
intersects(double, double)	75.0 %	6	2	8
constrain(double)	83.3 %	5	1	6
contains(double)	100.0 %	4	0	4
Range(double, double)	50.0 %	1	1	2
combine(Range, Range)	0.0 %	0	4	4
combineIgnoringNaN(Range, Rang	0.0 %	0	14	14
expand(Range, double, double)	0.0 %	0	2	2
expandToInclude(Range, double)	0.0 %	0	6	6
max(double, double)	0.0 %	0	4	4
min(double, double)	0.0 %	0	4	4
scale(Range, double)	0.0 %	0	2	2
shift(Range, double)		0	0	0
shift(Range, double, boolean)	0.0 %	0	2	2
shiftWithNoZeroCrossing(double,	0.0 %	0	4	4
equals(Object)	0.0 %	0	6	6
getCentralValue()		0	0	0
getLength()		0	0	0
getLowerBound()		0	0	0
getUpperBound()		0	0	0
hashCode()		0	0	0
intersects(Range)		0	0	0
isNaNRange()	0.0 %	0	4	4
toString()		0	0	0

Range Method Coverage

	Coverage	Covered Methods	Missed Methods	Total Methods
Range.java	30.4 %	7	16	23
Range	30.4 %	7	16	23
Range(double, double)	100.0 %	1	0	1
constrain(double)	100.0 %	1	0	1
contains(double)	100.0 %	1	0	1
getCentralValue()	100.0 %	1	0	1
getLowerBound()	100.0 %	1	0	1
getUpperBound()	100.0 %	1	0	1
intersects(double, double)	100.0 %	1	0	1
combine(Range, Range)	0.0 %	0	1	1
combineIgnoringNaN(Range, Rang	0.0 %	0	1	1
expand(Range, double, double)	0.0 %	0	1	1
expandToInclude(Range, double)	0.0 %	0	1	1
max(double, double)	0.0 %	0	1	1
min(double, double)	0.0 %	0	1	1
scale(Range, double)	0.0 %	0	1	1
shift(Range, double)	0.0 %	0	1	1
shift(Range, double, boolean)	0.0 %	0	1	1
shiftWithNoZeroCrossing(double,	0.0 %	0	1	1
equals(Object)	0.0 %	0	1	1
getLength()	0.0 %	0	1	1
hashCode()	0.0 %	0	1	1
intersects(Range)	0.0 %	0	1	1
isNaNRange()	0.0 %	0	1	1
toString()	0.0 %	0	1	1

After Lab Coverage Screenshots:

Range Statement Coverage:

DataUtilitiesTest.java | RangeTest.java | Coverage | DataUtilities.java | Range.java

RangeTest (1) (Mar. 3, 2022 11:49:50 a.m.)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
src	0.7 %	1,563	214,748	216,311
org.jfree.data	20.8 %	1,547	5,878	7,425
Range.java	94.6 %	440	25	465
Range	94.6 %	440	25	465
combine(Range, Range)	100.0 %	26	0	26
expand(Range, double, double)	100.0 %	40	0	40
expandToInclude(Range, double)	100.0 %	34	0	34
max(double, double)	100.0 %	14	0	14
min(double, double)	100.0 %	14	0	14
scale(Range, double)	100.0 %	24	0	24
shift(Range, double)	100.0 %	5	0	5
shift(Range, double, boolean)	100.0 %	29	0	29
shiftWithNoZeroCrossing(double,	100.0 %	24	0	24
constrain(double)	100.0 %	25	0	25
contains(double)	100.0 %	14	0	14
getCentralValue()	100.0 %	10	0	10
getLength()	100.0 %	6	0	6
getLowerBound()	100.0 %	3	0	3
getUpperBound()	100.0 %	3	0	3
hashCode()	100.0 %	28	0	28
intersects(Range)	100.0 %	7	0	7
isNaNRange()	100.0 %	12	0	12
toString()	100.0 %	16	0	16
combineIgnoringNaN(Range, Ranc	95.7 %	44	2	46
intersects(double, double)	92.6 %	25	2	27
equals(Object)	92.3 %	24	2	26
Range(double, double)	40.6 %	13	19	32

Range Branch Coverage:

DataUtilitiesTest.java | RangeTest.java | Coverage | DataUtilities.java | Range.java

RangeTest (1) (Mar. 3, 2022 11:49:50 a.m.)

Element	Coverage	Covered Branches	Missed Branches	Total Branches
org.jfree.data	11.2 %	63	499	562
RangeTest.java		0	0	0
UnknownKeyException.java		0	0	0
Range.java	87.5 %	63	9	72
Range	87.5 %	63	9	72
shift(Range, double)		0	0	0
getCentralValue()		0	0	0
getLength()		0	0	0
getLowerBound()		0	0	0
getUpperBound()		0	0	0
hashCode()		0	0	0
intersects(Range)		0	0	0
toString()		0	0	0
combine(Range, Range)	100.0 %	4	0	4
expand(Range, double, double)	100.0 %	2	0	2
expandToInclude(Range, double)	100.0 %	6	0	6
max(double, double)	100.0 %	4	0	4
min(double, double)	100.0 %	4	0	4
scale(Range, double)	100.0 %	2	0	2
shift(Range, double, boolean)	100.0 %	2	0	2
shiftWithNoZeroCrossing(double,	100.0 %	4	0	4
contains(double)	100.0 %	4	0	4
constrain(double)	83.3 %	5	1	6
equals(Object)	83.3 %	5	1	6
combineIgnoringNaN(Range, Ranç	78.6 %	11	3	14
intersects(double, double)	75.0 %	6	2	8
isNaNRange()	75.0 %	3	1	4
Range(double, double)	50.0 %	1	1	2
ComparableObjectItem.java	0.0 %	0	12	12
ComparableObjectSeries.java	0.0 %	0	60	60
DataUtilities.java	0.0 %	0	48	48

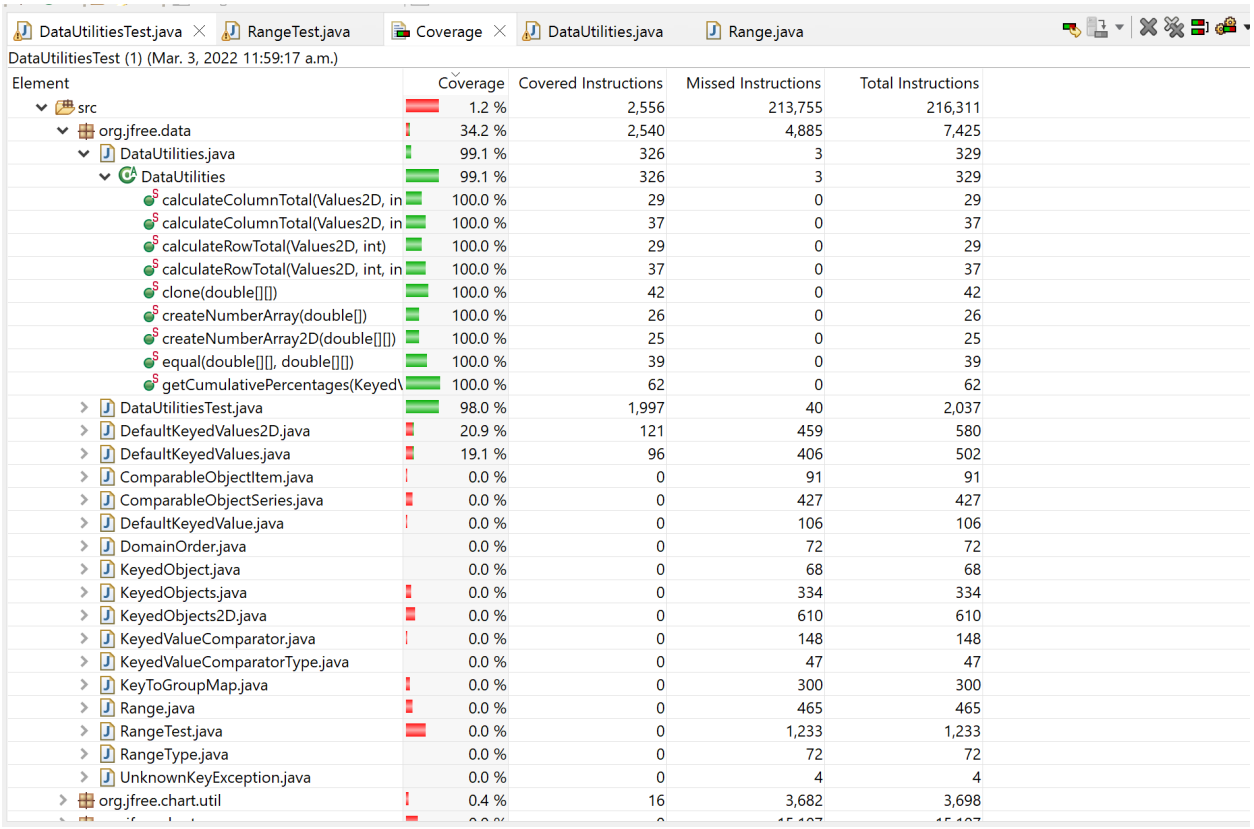
Range Method Coverage:

DataUtilitiesTest.java | RangeTest.java | Coverage | DataUtilities.java | Range.java

RangeTest (1) (Mar. 3, 2022 11:49:50 a.m.)

Element	Coverage	Covered Methods	Missed Methods	Total Methods
JFreeChar_Lab3	1.1 %	96	8,426	8,522
src	1.1 %	96	8,426	8,522
org.jfree.data	31.7 %	95	205	300
Range.java	100.0 %	23	0	23
Range	100.0 %	23	0	23
combine(Range, Range)	100.0 %	1	0	1
combineIgnoringNaN(Range, Ranç	100.0 %	1	0	1
expand(Range, double, double)	100.0 %	1	0	1
expandToInclude(Range, double)	100.0 %	1	0	1
max(double, double)	100.0 %	1	0	1
min(double, double)	100.0 %	1	0	1
scale(Range, double)	100.0 %	1	0	1
shift(Range, double)	100.0 %	1	0	1
shift(Range, double, boolean)	100.0 %	1	0	1
shiftWithNoZeroCrossing(double,	100.0 %	1	0	1
Range(double, double)	100.0 %	1	0	1
constrain(double)	100.0 %	1	0	1
contains(double)	100.0 %	1	0	1
equals(Object)	100.0 %	1	0	1
getCentralValue()	100.0 %	1	0	1
getLength()	100.0 %	1	0	1
getLowerBound()	100.0 %	1	0	1
getUpperBound()	100.0 %	1	0	1
hashCode()	100.0 %	1	0	1
intersects(double, double)	100.0 %	1	0	1
intersects(Range)	100.0 %	1	0	1
isNaNRange()	100.0 %	1	0	1
toString()	100.0 %	1	0	1
RangeTest.java	98.6 %	72	1	73
ComparableObjectItem.java	0.0 %	0	8	8
ComparableObjectSeries.java	0.0 %	0	20	20

DataUtilities Statement Coverage:

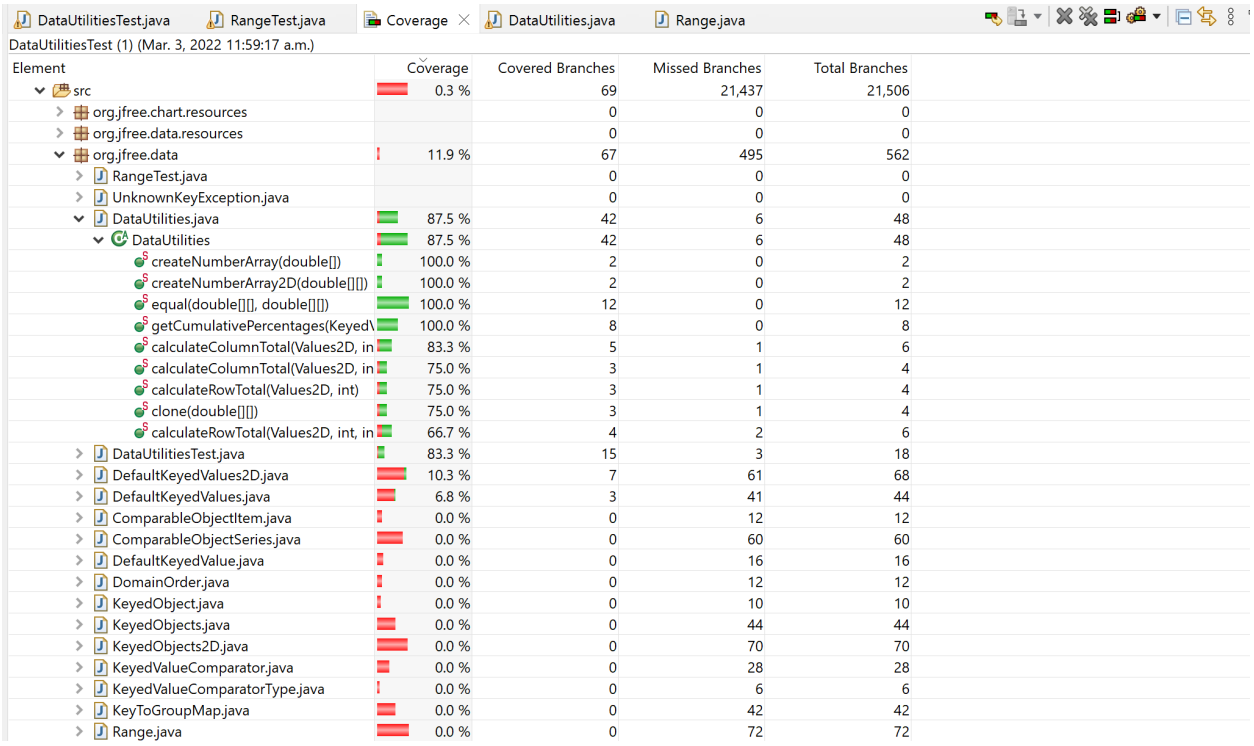


DataUtilitiesTest.java | RangeTest.java | Coverage | DataUtilities.java | Range.java

DataUtilitiesTest (1) (Mar. 3, 2022 11:59:17 a.m.)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
src	1.2 %	2,556	213,755	216,311
org.jfree.data	34.2 %	2,540	4,885	7,425
DataUtilities.java	99.1 %	326	3	329
DataUtilities	99.1 %	326	3	329
calculateColumnTotal(Values2D, in	100.0 %	29	0	29
calculateColumnTotal(Values2D, in	100.0 %	37	0	37
calculateRowTotal(Values2D, int)	100.0 %	29	0	29
calculateRowTotal(Values2D, int, in	100.0 %	37	0	37
clone(double[][])	100.0 %	42	0	42
createNumberArray(double[])	100.0 %	26	0	26
createNumberArray2D(double[][])	100.0 %	25	0	25
equal(double[][], double[][])	100.0 %	39	0	39
getCumulativePercentages(Keyed\	100.0 %	62	0	62
DataUtilitiesTest.java	98.0 %	1,997	40	2,037
DefaultKeyedValues2D.java	20.9 %	121	459	580
DefaultKeyedValues.java	19.1 %	96	406	502
ComparableObjectItem.java	0.0 %	0	91	91
ComparableObjectSeries.java	0.0 %	0	427	427
DefaultKeyedValue.java	0.0 %	0	106	106
DomainOrder.java	0.0 %	0	72	72
KeyedObject.java	0.0 %	0	68	68
KeyedObjects.java	0.0 %	0	334	334
KeyedObjects2D.java	0.0 %	0	610	610
KeyedValueComparator.java	0.0 %	0	148	148
KeyedValueComparatorType.java	0.0 %	0	47	47
KeyToGroupMap.java	0.0 %	0	300	300
Range.java	0.0 %	0	465	465
RangeTest.java	0.0 %	0	1,233	1,233
RangeType.java	0.0 %	0	72	72
UnknownKeyException.java	0.0 %	0	4	4
org.jfree.chart.util	0.4 %	16	3,682	3,698

DataUtilities Branch Coverage:

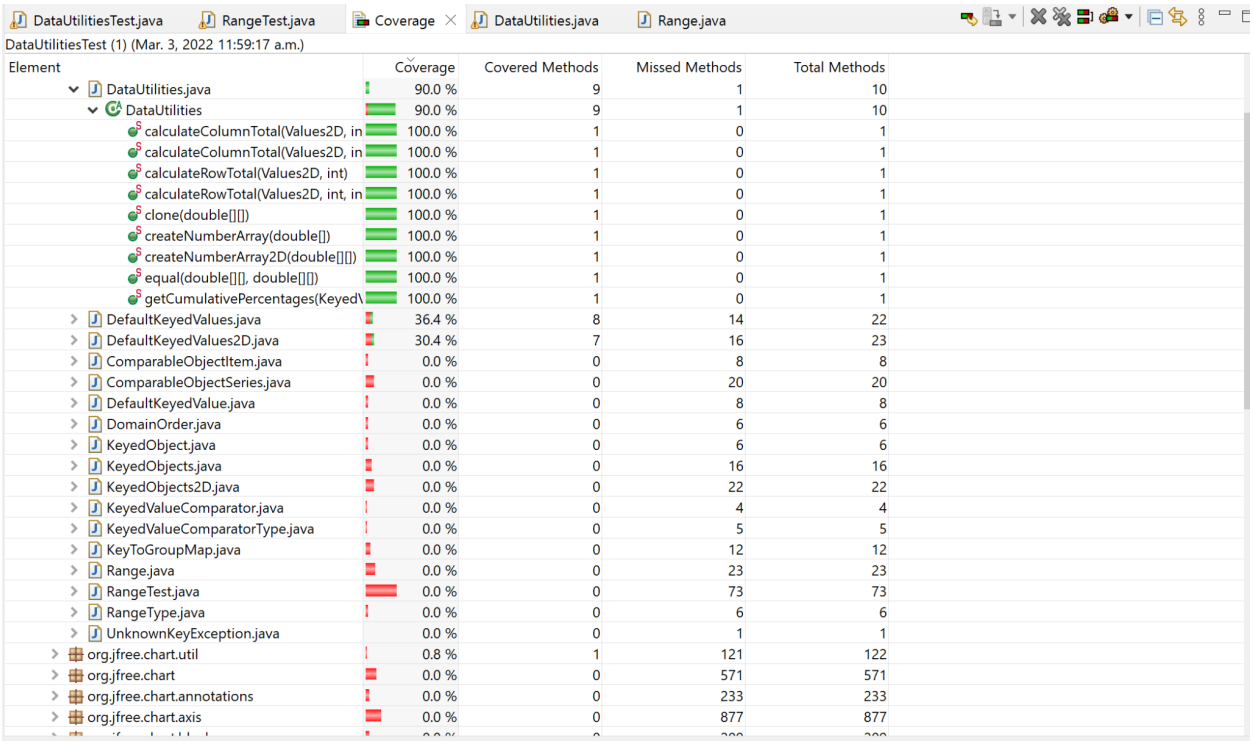


DataUtilitiesTest.java | RangeTest.java | Coverage | DataUtilities.java | Range.java

DataUtilitiesTest (1) (Mar. 3, 2022 11:59:17 a.m.)

Element	Coverage	Covered Branches	Missed Branches	Total Branches
src	0.3 %	69	21,437	21,506
org.jfree.chart.resources		0	0	0
org.jfree.data.resources		0	0	0
org.jfree.data	11.9 %	67	495	562
RangeTest.java		0	0	0
UnknownKeyException.java		0	0	0
DataUtilities.java	87.5 %	42	6	48
DataUtilities	87.5 %	42	6	48
createNumberArray(double[])	100.0 %	2	0	2
createNumberArray2D(double[][])	100.0 %	2	0	2
equal(double[][], double[][])	100.0 %	12	0	12
getCumulativePercentages(Keyed\	100.0 %	8	0	8
calculateColumnTotal(Values2D, in	83.3 %	5	1	6
calculateColumnTotal(Values2D, in	75.0 %	3	1	4
calculateRowTotal(Values2D, int)	75.0 %	3	1	4
clone(double[][])	75.0 %	3	1	4
calculateRowTotal(Values2D, int, in	66.7 %	4	2	6
DataUtilitiesTest.java	83.3 %	15	3	18
DefaultKeyedValues2D.java	10.3 %	7	61	68
DefaultKeyedValues.java	6.8 %	3	41	44
ComparableObjectItem.java	0.0 %	0	12	12
ComparableObjectSeries.java	0.0 %	0	60	60
DefaultKeyedValue.java	0.0 %	0	16	16
DomainOrder.java	0.0 %	0	12	12
KeyedObject.java	0.0 %	0	10	10
KeyedObjects.java	0.0 %	0	44	44
KeyedObjects2D.java	0.0 %	0	70	70
KeyedValueComparator.java	0.0 %	0	28	28
KeyedValueComparatorType.java	0.0 %	0	6	6
KeyToGroupMap.java	0.0 %	0	42	42
Range.java	0.0 %	0	72	72

DataUtilities Method Coverage:



Element	Coverage	Covered Methods	Missed Methods	Total Methods
DataUtilities.java	90.0 %	9	1	10
DataUtilities	90.0 %	9	1	10
calculateColumnTotal(Values2D, in	100.0 %	1	0	1
calculateColumnTotal(Values2D, in	100.0 %	1	0	1
calculateRowTotal(Values2D, int)	100.0 %	1	0	1
calculateRowTotal(Values2D, int, in	100.0 %	1	0	1
clone(double[][])	100.0 %	1	0	1
createNumberArray(double[])	100.0 %	1	0	1
createNumberArray2D(double[][])	100.0 %	1	0	1
equal(double[][], double[][])	100.0 %	1	0	1
getCumulativePercentages(Keyed\	100.0 %	1	0	1
DefaultKeyedValues.java	36.4 %	8	14	22
DefaultKeyedValues2D.java	30.4 %	7	16	23
ComparableObjectItem.java	0.0 %	0	8	8
ComparableObjectSeries.java	0.0 %	0	20	20
DefaultKeyedValue.java	0.0 %	0	8	8
DomainOrder.java	0.0 %	0	6	6
KeyedObject.java	0.0 %	0	6	6
KeyedObjects.java	0.0 %	0	16	16
KeyedObjects2D.java	0.0 %	0	22	22
KeyedValueComparator.java	0.0 %	0	4	4
KeyedValueComparatorType.java	0.0 %	0	5	5
KeyToGroupMap.java	0.0 %	0	12	12
Range.java	0.0 %	0	23	23
RangeTest.java	0.0 %	0	73	73
RangeType.java	0.0 %	0	6	6
UnknownKeyException.java	0.0 %	0	1	1
org.jfree.chart.util	0.8 %	1	121	122
org.jfree.chart	0.0 %	0	571	571
org.jfree.chart.annotations	0.0 %	0	233	233
org.jfree.chart.axis	0.0 %	0	877	877

Pros and Cons of coverage tools used and Metrics you report

The primary coverage tool we used was EclEmma. This tool was simple to use and provided fast and accurate results. When used with eclipse, code segments are highlighted in red and green to illustrate the coverage of that method. A major downside with EclEmma is that you cannot calculate the condition coverage for each class. So, we recorded the statement/instruction coverage, the branch coverage and the method coverage for the given classes. We had researched JaCoCo and realized it is the same as EclEmma.

A comparison on the advantages and disadvantages of requirements-based test generation and coverage-based test generation.

While requirement-based testing allows testers to test specific functionalities of the SUT, coverage based testing allows testers to test specific branches and instructions within the SUT. So, coverage based test generation allows for much more thorough testing, although many tests could be redundant as certain system states are unattainable through realistic use.

Compared to requirement-based testing, coverage testing is also more time consuming because, like mentioned above, we may need to write a larger number of test cases that can become redundant just to achieve a given coverage value. Although requirement based testing can be less time consuming than coverage based testing, a disadvantage to this method that we encountered is that in the documentation for certain methods,

specific functionalities and the overloaded versions of those methods were missed and not included in the documentation

A discussion on how the team work/effort was divided and managed

Each team member took responsibility for the methods they chose and wrote several unit tests for each until branches and statements were covered as much as possible.

Any difficulties encountered, challenges overcome, and lessons learned from performing the lab

Some methods had conditions that were difficult to reach due to input verification from previous functions. For example, the Range constructor does not allow the lower bound to be higher than the upper bound. This caused an issue within the expand() method since there was a condition where the lower variable had to be greater than the upper variable. We were able to overcome this by setting the margins to negative values.

There were also some issues we ran into when using JMock for some of the test cases. This issue caused Eclemma to ignore any Mockery function calls, resulting in the tool calculating the coverage metrics improperly

Comments/feedback on the lab itself