SEng 438 - Software Testing, Reliability and Quality

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

Group #	9
Student Names	Satyaki Ghosh
	Artin Rezaee
	Jenard Cabia

Table of Contents

1 Introduction

The main purpose of this assignment was to introduce and familiarize ourselves with automated unit testing based on the requirements for each unit of the System Under Test. We are to utilize the JUnit framework along with mock objects during the test-code development stage. The unit tests will test the methods `org.jfree.data.Range` and `org.jfree.data.DataUtilities` classes against their specifications which are documented in their Javadoc. During the design phase of the test cases, we are to apply black-box test-case design techniques discussed in the lectures such as equivalence classes, and boundary value analysis.

2 Detailed Description of Unit Test Strategy

During the design phase of our test plan, we applied the two black-box design techniques of equivalence class testing and boundary class testing. For the equivalence class testing, we identified and partitioned the classes for each method based on the expected behavior of its legal and illegal inputs. Once equivalence classes were defined, boundary testing was applied to applicable classes by having their minimum and maximum values used as input. As instructed, each test case method involves one test case only to ensure consistency of the tests.

In testing the class `org.jfree.data.DataUtilities`, we had to use Mock Objects as some of its methods uses the interfaces `Values2D` and `KeyedValues` for its inputs. Mock Objects allows the Unit Tests to focus solely on the behavior and correctness of the System Under Test, not the other systems that it utilizes or depends on.

3 Test Cases Developed

The tables below shows the equivalence classes identified for each methods being tested.

Table 1: Test Cases for `org.jfree.data.Range`

Method	**Equivalence Class**	**Selected Test Case**	**Expected**	**Description**
combine	First range is null	Range 1 = null Range 2 = (1, 2)	Range (1, 2)	If the first range is null, then the function just returns the second range
	Second range is null	Range 1 = (1, 2) Range 2 = null	Range (1, 2)	If the second range is null, then the function just returns the first range

	Both ranges are null	Range 1 = null Range 2 = null	null	If both the ranges are null, then the function returns null
	The two ranges are continuous	Range 1 = (0, 2) Range 2 = (2, 3)	Range (0, 3)	If the ranges are continuous, then the function returns their combined range
	The two ranges are not continuous	Range 1 = (0, 2) Range 2 = (10, 20)	Range (0, 20)	If the ranges are not continuous, then the function returns their combined range while including the range that is not covered by either

contains	Number less than lower bound	Range = (0,10)
		contains(-1.00)
			Range does not include -1.00	Test if a number less than lower bound is contained in the range

	Number greater than upper bound	Range = (0,10)
		contains(100000.00)
			Range does not include 100000.00	Tests if a number greater than upper bound is contained in the margin

	Number is in range	Range = (0,10)
		contains(6.0)
			Number is in range	Test if a number in range, is contained in the range

	Number is the same as lower bound	Range = (0,10)
		contains(0.0)
			Number is in range	Test if the number in the lower bound is contained in the range

	Number is the upper bound of the range	Range = (0,10)
		contains(10.0)
			Number is in range	Test if the upper bound number is contained in the range

expand	Range parameter is null	Range = null expand(range, 0.25, 0.50)	InvalidParameter Exception is thrown	expands a null Range with any valid lower and upper margins
	The lowerMargin parameter is negative and the upperMargin parameter is positive	Range = (2, 6), expand(range, -0.25, 0.50)	from (2, 6), Range becomes (3, 8)	expands a range with a negative lower margin and a positive upper margin
	The lowerMargin parameter is positive and the upperMargin parameter is negative	Range = (2, 6) expand(range, 0.75, -0.20)	from (2, 6), Range becomes (-1, 5.2)	expands a range with a positive lower margin and a negative upper margin
	Both the lowerMargin and the upperMargin are positive	Range = (2, 6) expand(range, 0.25, 0.50)	from (2, 6), Range becomes (1, 8)	expands a range with a positive lower margin and a positive upper margin
	Both the lowerMargin and the upperMargin are zero	Range = (2, 6) expand(range, 0, 0)	Range stays the same	expands a range with zero margins
expandToInclude	number less than the lower bound	Range = (2, 10), expandToInclude(range, 1.0)	Range expands to include number 1 and changes to (1,10)	Expands the range from lower bound to include the number that is less than the lower bound. The upper bound stays the same

	number more than the upper bound	Range = (2,10), expandToInclude(range, 100)	Range expands from the upper bound to include number 100. Range changes to (2,100)	Expands the range from upper bound to include the number that is greater than the upper bound. The lower bound stays the same
	number that is already in range	Range = (2,10), expandToInclude(range,5.0)	Range stays the same	Test if range stays the same when the number is already within range
	a null range	range = (2,10), expandToInclude(null, 5.0)	Returns the range as (5.0,5.0)	Tests if a null range is extended to include only the number that is passed to the function

shift (does not allow zero crossing)	Base is null	Range = null
		Shift = 2
			Throws InvalidParameter Exception	If a null object is selected as the base, then an InvalidParameter Exception is thrown

	Shifting right without crossing zero	Range (5, 10) Shift = 2	Range (7, 12)	Normal shifting right without any of the bounds crossing zero
	Shifting left without crossing zero	Range (5, 10) Shift = -2	Range (3, 8)	Normal shifting left without any of the bounds crossing zero
	Shifting right when length of range is 1	Range (5, 5) Shift = 2	Range (7, 7)	Shifting right when the length of the range is 1, without any of the bounds crossing zero

	Shifting left when length of range is 1	Range (5, 5) Shift = -2	Range (3, 3)	Shifting left when the length of the range is 1, without any of the bounds crossing zero
	Shifting right when range is from negative to positive	Range (-5, 5) Shift = 2	Range (-3, 7)	Shifting right when the lower bound is negative and the upper bound is positive
	Shifting left when range is from negative to positive	Range (-5, 5) Shift = -2	Range (-7, 3)	Shifting left when the lower bound is negative and the upper bound is positive
	Shifting right with upper bound crossing zero	Range (-20, -10) Shift = 15	Range (-5, 0)	Testing to see if the function allows shifting right when the upper bound is about the cross zero
	Shifting left with lower bound crossing zero	Range (10, 20) Shift = -15	Range (0, 5)	Testing to see if the function allows shifting left when the lower bound is about the cross zero
	Shifting right with both bounds crossing zero	Range (-20, -10) Shift = 50	Range (0, 0)	Testing to see if the function allows shifting right when the both bounds are about the cross zero
	Shifting left with both bounds crossing zero	Range (10, 20) Shift = -50	Range (0, 0)	Testing to see if the function allows shifting left when the both bounds are about the cross zero

Table 2: Test Cases for `org.jfree.data.DataUtilities`

Method	**Equivalence Class**	**Selected Test Case**	**Expected**	**Description**
calculateColumnTotal	Data is null	calculateColumnTotal(null, 0)	Throws InvalidParameterException	When an invalid data object is used, the method should throw an exception
	Values2D is empty	Values = [ ] calculateColumnTotal(values, 0)	Function returns 0	If the 2D data object is empty, then the column total should be 0
	Values2D is not empty with column = 0	Values = [1] [2] [3] Column = 0	Function returns 6	If the 2D data object has only one column and the total is being calculated for column = 0
	Values2D is not empty with column = -1	Values = [1] [2] [3] Column = -1	Function returns 0	If the 2D data object is valid and the total is being calculated for an invalid column
	Values2D is not empty with column = 1	Values = [1, 1] [2, 2] [3, 3] Column = 1	Function returns 6	If the 2D data object is valid and the total is being for a valid column
	Values2D is not empty with column = 2	Values = [1] [2] Column = 2	Function returns 0	If the 2D data object is valid and the total is being calculated for a column whose index is out of bounds
calculateRowTotal	Data is null	calculateRowTotal(null, 0)	Throws InvalidParameterException	When an invalid data object is used, the method should throw an exception

	Values2D is empty	Values = [ ] calculateRowTotal(values, 0)	Function returns 0	If the 2D data object is empty, then the row total should be 0
	Values2D is not empty with row = 0	Values = [1,2,3] row = 0	Function returns 6	If the 2D data object has only one row and the total is being calculated for row = 0
	Values2D is not empty with row = -1	Values = [1,2,3] row = -1	Function returns 0	If the 2D data object is valid and the total is being calculated for an invalid row
	Values2D is not empty with row = 1	Values = [1,2,3] [1,2,3] row = 1	Function returns 6	If the 2D data object is valid and the total is being for a valid row
	Values2D is not empty with row = 2	Values = [1,2] row = 2	Function returns 0	If the 2D data object is valid and the total is being calculated for a row whose index is out of bounds
createNumberArray	Data parameter is null	createNumberArray(null)	The test fails as null is not permitted as input	creates an array of number from a null input
	Data parameter is any object other than an array of double	data = null createNumberArray(data)	InvalidParameterException is thrown	creates an array of number from an object other than an array of double
	Data parameter is an array of double	data = {3.0, 2.0, 1.0, 4.5} createNumberArray(data)	Returns a number array that has the same size and numbers as the array of doubles	creates an array of number from an array of double

createNumberArray2D (data)	Data parameter is null	createNumberArray(null)	The test fails as null parameter is not permitted	creates a 2D array of number from a null input
	Data parameter is any object other than a 2D array of double	data = null createNumberArray(data)	InvalidParameter Exception is thrown	creates a 2D array of number from an object other than a 2D array of double
	Data parameter is a 2D array of double	data = {1.0, 2.0, 3.0, 4.0}, {2.0, 4.0, 6.0, 8.0}, {3.0, 6.0, 9.0, 12.0} createNumberArray(data)	Returns a 2D number array that has the same size and numbers as the 2D array of doubles	creates an array of number from a 2D array of double
getCumulativePercentages	A null KeyedValues	Data = null	Test should fail	Test if the method will fail upon receiving a null as an argument
	A null KeyedValues	Data = null	InvalidParameter Exception should be thrown	Test if an InvalidParameter Exception is thrown upon receiving a null as an argument
	A KeyedValues with positive numbers as its entries	KeyedValues Data = <0,1>,<1,2>,<2,3>	Return value should be: KeyedValues newData = <0,0.166>,<1,0.500>,<2,1.0>	Test if the returned cumulative percentages key-values are correct when all numbers in the original key-values are positive

	A KeyedValues with positive and negative numbers as its entries	KeyedValues Data = <0,-1>,<1,2>,<2,3>	Returned value should be: KeyedValues newData = <0,-0.250>,<1,0.250>,<2,1.0>	Test if the returned cumulative percentages key-values are correct when a mix of positive and negative numbers are supplied in the original key-values

4 How the Team Work/Effort was divided and managed

Familiarization of the System Under Test and the JUnit Framework in Eclipse was performed as a group by putting the project in a shared git repository which allows all members to work on the assignment through their own computers and in their own times. The design phase of the unit test cases were performed as a group and during its implementation, the development of all test cases was divided evenly amongst the members of the group. Once each test cases were completed individually, they were then reviewed and finalized as a group.

5 Difficulties encountered, challenges overcome, and lessons learned

One of the difficulties encountered by the team was understanding docs that were not worded properly. In addition, the team had difficulties working with jMocks as the team had no prior experience working with this technology. The lesson learned during this assignment was that making tests take an incredible amount of time and precision to ensure high quality software is produced. In addition, the team learned to not to rely solely on the javadocs and try to design tests separately so that they have the maximum possible coverage.

6 Comments/Feedback on the Lab and Lab Document Itself

Testing some of the functionalities such as CreateNumberArray are not that important since it will not fail most of the time and it is very confusing on how you would test if an specific object got created. In addition, the assignment did not have clear instruction on what to do when a test results in an error instead of a failure. In the JavaDocs some of the methods specified that they could throw a certain error, but during testing different errors were being thrown. Other methods would specify that no errors would be thrown, but some of those methods did throw errors. This made writing valid test cases much harder and resulted in errors instead of failures for some of the test cases.