Detailed description of unit test strategy

Lana Hawa

- My unit test strategy is based on black-box testing techniques, to verify that each method behaves according to its specifications. Tests are designed to cover normal cases, edge cases, and error conditions, ensuring thorough coverage and early defect detection. This approach is applied to both the 1D and 2D conversion methods, with tests automated using JUnit.
- Input partitions
    - createNumberArray
        - The input partitions for this method include a valid partition containing typical nonzero positive values, which confirms that each element is correctly converted into a Number. A negative partition ensures that the method properly handles sign conversion and arithmetic on negative values. A zero partition tests that the method does not inadvertently produce nulls or errors when all inputs are zero. Lastly, an edge partition consisting of an empty array verifies that the method perfectly handles the absence of data without throwing exceptions.
    - createNumberArray2D
        - The input partitions for the 2D conversion method include a valid partition where a typical 2D array of positive nonzero values is used to verify standard behavior. A negative partition ensures that negative numbers in each row are handled correctly across both dimensions. A zero partition tests that all zero values are converted accurately without producing null elements. Additionally, a mixed partition challenges the method with a variety of numeric values in one array, while an edge partition of an empty 2D array confirms that the method can handle cases with no input data.

John T

- My unit test strategy followed a weak-equivalence class testing process. This allowed me to test the general functionality of the method and how the method interacted with different input variables values.

- Input partitions
    - combine
        - Since a range can contain either only positive values, only negative values, or a mix of these values, this would be the first equivalence class. The second equivalence class would be the similarity between the two ranges. Therefore, we would also test when the ranges are separated, similar in range, and equal to each other. In addition to these tests, there are edge cases that also need to be tested such as when one or both ranges are null.

- constrain
    - Since constrain uses a test value to a specific range, we would be mainly focussed on the properties of the test value. The first equivalence class would be the sign of the value, being either negative or positive. The second equivalence class is where the test value is located with respect to the range, be it less than the lower bound, equal to the lower bound, within bounds, equal to the upper bound, and greater than the upper bound.

- getCentralValue
    - The two equivalence classes in this method would be the length of the range and the type of range. Expanding on the type of range, this refers to what values the range contains. Be it purely negative, positive, or a mix of both. The length of the range can either be even or odd which is why it's another equivalence class.

- calculateRowTotal
    - This method primarily tests the addition of values in the data table's row. Therefore, there are two equivalence classes which are the values used and the length of the row. The values used can be either negative or positive and the length of the row can be one or two and more.

Ron Ibe

- The unit test strategy for the toString and getCumulativePercentages methods follows a black-box testing approach combined with an equivalence class and boundary value testing.

Input partitions

toString:

- The toString method ensures that numerical ranges are correctly formatted. The first equivalence class considers whether the range consists of only positive values, negative values, or both. Handles cases where lower and upper bounds are the same, such as zero range. This includes edge cases like very large numbers or small, ensuring correct formatting in all scenarios.

getCumulativePercentages:

- The getCumulativePercentages method revolves around the distribution and nature of dataset values. Used many different numbers of elements in the dataset, covering single values, multiple values, or mix values. Handled the presence of zero values in the

dataset, ensuring it can handle zeros appearing in any place in the dataset. Another equivalence class is an edge case where all values are zero, which results in a NaN.

Mark jimenez

My unit strategy for the testing intersects and calculateColumnTotal methods is the black-box testing with the addition of boundary testing for the intersects method.

Intersects

- For the intersects, the upper and lower bounds were modified to fit and test certain parts of the example range. This is done by taking into account multiple possibilities. For example, if the testRange is within, outside, equal, near lower or upper and completely outside the lower and upper bounds of the example range.

calculateColumnTotal

- For calculateColumnTotal, making sure the method functions as intended is the most important part. Testing positive and negative values on a specific column is significant to gauge the accuracy of the method. As well as testing a table consisting of zero entries, this ensures that the testing is consistent whether the table has entries or not.

Test Cases Developed

classes	test method	test case creator
org.jfree.data.Range	combine	John
org.jfree.data.Range	constrain	John
org.jfree.data.Range	getCentralValue	John
org.jfree.data.Range	intersects	Mark
org.jfree.data.Range	toString	Ivan
org.jfree.data.DataUtilities	calculateColumnTotal	Mark
org.jfree.data.DataUtilities	calculateRowTotal	John
org.jfree.data.DataUtilities	createNumberArray	Lana
org.jfree.data.DataUtilities	createNumberArray2D	Lana
org.jfree.data.DataUtilities	getCumulativePercentages	Ivan

Input Partitions

method tested	test case name	input partitions
combine	oneNullRange	edge case, one input is null and the other is a valid range
	bothNullRange	edge case, both input variables are null
	combineSeperateRanges	One input is positive range, second input is negative range, there is a gap between positive range lower bound and the negative range lower bound
	combineSimilarRanges	Both inputs are positive, there are overlapping values between ranges
	combineEqualRanges	Both ranges are positive and are equivalent to each other
constrain	valueLessThanRange	The input range is a fixed range containing both negative and positive values, the test value is negative and less than the lower bound of the range
	valueEqualsLowerBound	The input range is a fixed range containing both negative and positive values, test value is equal to the lower bound of the range
	valueWithinRangeExcludingBounds	The input range is a fixed range containing both negative and positive values, the test value is positive and within bounds

	valueEqualsUpperBound	The input range is a fixed range containing both negative and positive values, test value is positive and greater than the upper bound of the range
	valueGreaterThanRange	The input range is a fixed range containing both negative and positive values, test value is positive and greater
getCentralValue	centralValueShouldBeZero	This test was given as an example from the assignment document
	positiveRangeEvenLengthCentralValue	The target range contained only positive values and was of even length
	positiveRangeOddLengthCentralValue	The target range contained only positive values and was of odd length
	mixedRangeEvenLengthCentralValue	The target range contained mixed values and was of even length
	mixedRangeOddLengthCentralValue	The target range contained mixed values and was of odd length
	negativeRangeEvenLengthCentralValue	The target range contained only negative values and was of even length
	negativeRangeOddLengthCentralValue	The target range contained only negative values and was of old length
calculateRowTotal	simpleCalculateRowTotal	The table was created with two rows, each containing one item, one positive value and one tested value, the method

		was then called on the positive value
	negativeValueCalculateRowTotal	The table was created with two rows, each containing one item, one positive value and one tested value, the method was then called on the negative value
	calculateRowTotalOf2x2Table	The table was created with two rows, each containing two items, all positive values, the method was then called on the second row
toString	positiveRange	Tests formatting when both bounds are positive
	negativeRange	Tests formatting when both bounds are negative
	negativePositiveRange	Tests formatting when one bound is positive and one bound is negative
	zeroRange	Tests formatting when both bounds are zero
getCumulativePercentages	dataHasSingleValue	Tests a single value in a keypair data results in a 100% cumulative percentage
	dataHasFiveValues	Tests cumulative percentages for uniform keypair data of five values
	dataHasFirstTwoValuesZero	Tests cumulative percentage of 3 values when first 2 values are zero
	dataHasLastTwoValuesZero	Tests cumulative percentage of 3 values when the last 2 values are zero
	dataHasAllZero	Tests cumulative

		percentage of 3 values when all values are zero
createNumberArray	createNumberArrayWithValidValues	consists of a non-empty array of positive, nonzero doubles to test normal conversion.
	createNumberArrayWithEmptyArray	an empty array, verifying that the method returns an empty Number array when no data is provided.
	createNumberArrayWithegativeNumbers	includes an array of negative doubles to ensure that negative values are correctly converted.
	createNumberrrayWithZeroValues	uses an array composed entirely of zeros, ensuring that zeros are handled properly without producing null elements.
	createNumberArrayAssertArrayEquals	applies a typical positive nonzero array for direct comparison of expected and actual results using assertArrayEquals.
createNumberAray2D	create NumberArray2DWithValidValues	consists of a standard 2D array with positive nonzero values to test proper two-dimensional conversion.
	createNumberArray2DWithEmptyArray	an empty 2D array, verifying that the method correctly returns an empty 2D Number array when no data is provided.
	createNumberArray2DNegativeNumbers	uses a 2D array composed entirely of negative double values to verify that the conversion method correctly processes and converts all negative numbers into the corresponding 2D

		Number array.
	createNumberArray2DZeroValues	includes a 2D array where every element is zero to ensure that zeros are accurately converted in every row and column.
	createNumberArray2DMixedValues	consists of a 2D array with mixed values to validate that the method handles positive, negative, and zero values across different rows and columns.
Intersects	testRangeIsBigger	One input is a positive range and the second is negative, both inputs covers the range of the exampleRange
	testRangeWithinExampleRange	One input is fully inside the exampleRange and the second range is also within the bounds of example Range
	testrRangeTouchingLower	The testRange touches the lower bound/ it is equal to the lower bound and the upper bound is within exampleRange
	testRangeTouchingUpper	The testRange touches the upper bound/ it is equal to the lower bound and the lower bound is within exampleRange
	testRangeSameSize	Both inputs are equal to the upper and lower bound of the example Range
	testRangeBefore	The testRange is completely before the example range/ lower bound of the example Range with no overlap

	testRangeAfter	The testRange is completely after the example range/ upper bound of the example Range with no overlap
	testRangeAroundLowerBound	The test range is touching the lower bound of the example range.
	testRangeAroundUpperBound	The test range is touching the upper bound of the example range
calculateColumnTotal	sumColumnWithpositiveNumbers	A table of four positive entries on column 2 is mocked. All the entries were then added to get the sum
	sumColumnWithNegativeNumbers	A table of three negative entries on column 1 is mocked. All the entries were then added to get the sum
	sumColumnWithNoValues	A table with no entries was inputted, then added the column 0.