Java Arrays

Project

Table of Contents

Arrays Assignment

Objectives

The objective of the assignment is to:

- Provide practice with the basics of Java, including arrays, classes, methods and fields.
- Provide practice using the Java command line tools.

Specifications

You are going to create a class called `ArrayData` with the following fields and methods.

***** DO NOT CHANGE THE FIELD OR METHOD DEFINITIONS BELOW. *****
***** YOU ARE REQUIRED TO IMPLEMENT THEM AS GIVEN. *****
YOU CAN ADD ADDITIONAL FIELDS AND METHODS.

The ArrayData class can be found on eLearning.

Field	Description
`private int rows`	Contains the total number of rows. The initial value is 10.
`private int columns`	Contains the total number of columns. The initial value is 10.
`private int grid[][]`	This array contains integers that are used by the class. The size of the array is contained in the fields, `rows` and `columns`.
`private int rowData[]`	This array contains the sum (or other operations) of the integers in each row of the `grid` array. For example, position 0 in the `rowData` array will contain the sum of the integer values in row 0 of the `grid` array. The size of this array is contained in the field, `rows`. NOTE: The results of other operations other than summation maybe performed and placed in this array.
`private int colData[]`	This contains the sum (or other operations) of the integers in each column of the `grid` array. For example, position 0 in the `colData` array will contain the sum of the integer values in column 0 of the `grid` array. The size of this array is contained in the field, `columns`. NOTE: The results of other operations other than summation maybe performed and placed in this array.

Method	Description
`public ArrayData()`	Initialize the `rows` and `columns` fields to their default values of 10 each and create the arrays `grid`, `rowData` and `colData`. Initialize the values in the `grid`, `rowData` and `colData` arrays to the default value of 0.

	Remember to create the arrays `grid`, `rowData` and `colData` using their default size of 10 rows and 10 columns.
`public ArrayData( int size )`	Initialize the `rows` and `columns` fields to their new values of `size` and change the size of the arrays `grid`, `rowData` and `colData` to their new size. Initialize the values in the `grid`, `rowData` and `colData` arrays to the default value of 0.
`public ArrayData( int size, int startingValue )`	Initialize the `rows` and `columns` fields to their new values of `size` and change the size of the arrays `grid`, `rowData` and `colData` to their new size. Initialize the values in the `grid`, `rowData` and `colData` arrays to the new value of `startingValue`.
`public void populate( int newValue, int totalInstances )`	This method will randomly choose a position in the `grid` array and change the value to `newValue`. It will repeat this a total of `totalInstances` times. For example, the call `populate(5,10)`, will randomly choose 10 positions in the `grid` array and set their value to 5. **When randomly choosing the positions, it is possible for the same position in the `grid` array to be chosen more than once. This is allowed in this assignment.**
`public void reverse( int rowNum, int start )`	Reverse the values in the row `rowNum`. The value start is the starting position in the row from which the values are reversed. For example, for the call `reverse( 2, 3 )`, the values in row 2 (third row) are reversed starting at position 3. If the values in that row were 1 2 3 4 5 6 7 8 then this method call will change the values to 1 2 3 **8 7 6 5 4**.
`public void sum( int inc )`	This method calculates the sum of each row, in `grid`, and places it in the `rowData` array and the sum of each column and places it in the `colData` array. For example, position 0 in the `rowData` array will contain the sum of the integer values in row 0 and so on. The value `inc` is added to each even value in the row before the summation takes place. For example, if the values in the row are 1 2 3 4 and the value passed to this method is 2, then the sum is calculated as follows. Sum = 1 + (2x2) + 3 +(4x2) = 1 + 4 + 3 + 8 = 16
`public void occurrence( int num )`	This method will calculate the number of times the integer `num` or a multiple of `num` appears in each row, in `grid`, and places it in the `rowData` array. Perform the same action for each column and place the results in the `colData` array. For example, if the row contains 2, 4, 7, 8 and the number passed is 4, the answer will be 2 since 2 and 7 are not divisible by 4.
`public void standardDeviation()`	This method will calculate the standard deviation of the integers in each row, in `grid`, and place it in the `rowData` array. Note that the calculation for the standard deviation produces a floating point number. Do not worry about the loss of precision when placed in the value in the `rowData` array.

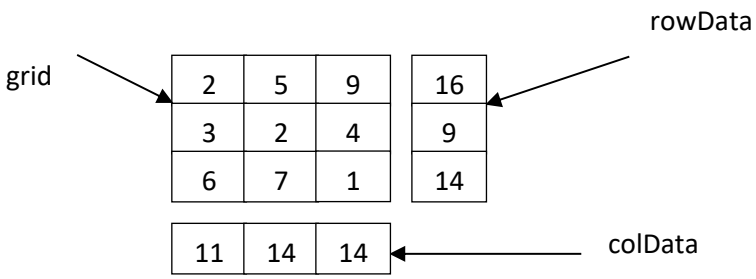
	The standard deviation for each row is calculated using the following formula. $$\sigma = \sqrt{\frac{\sum(X-\mu)^2}{N-1}}$$ Where: σ is the standard deviation for a row μ is the average/mean of the row $\sum$ is the symbol for the "the sum of" *X* represents the integer value in each position within the row *N* represents the total number of integers in the row i.e. the length of the row.
`public void swap(int rowNum1, int rowNum2 )`	This method will swap the values between the rows `rowNum1` and `rowNum2` in the `grid` array. For example, if the values in `rowNum1` are 1 2 3 and in `rowNum2` are 4 5 6. Then after the swap operation, the values in `rowNum1` become 4 5 6 and in `rowNum2` become 1 2 3.
`public int product( int rowNum, int colNum )`	This method will calculate the sum of the products of the values in the row, `rowNum` and the column, `colNum`. For example, if the values in the row are 2 4 6 and the values in the column are 1 3 5, then the result is, (2x1) + (4x3) + (6x5) = 2 + 12 + 30 = 44 Note that you must ensure that the total number of rows are the same as the total number of columns in the `grid` array. If their values are different, then adjust as follows. If rows > columns then use a temporary value rowsTemp, where rowsTemp = columns If columns > rows then use a temporary value columnsTemp, where columnsTemp = rows
`public void print()`	Prints the contents of the `grid`, `rowData` and `colData` arrays using the following format. 2 | 5 | 9 || 16 3 | 2 | 4 || 9 6 | 7 | 1 || 14 11 | 14 | 14

	Where the bottom row are the values in the `colData` array and the column on the right are the values of the `rowData` array.
`public void print( int rows, int columns )`	This method is the same as the previous `print` method but the number of rows and columns printed is set by the parameter values. For example, a call of `print( 2, 1 )` will give the output. 2 || 16 3 || 9 11

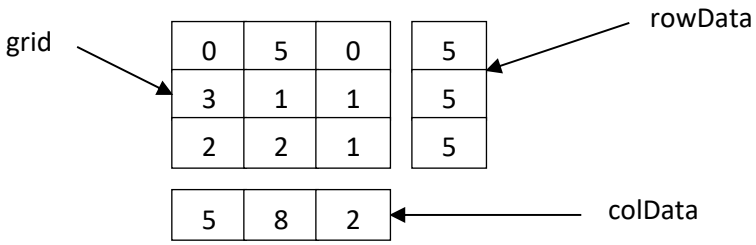
You must implement each of the methods described in the previous table. You must also use the fields described. You can create other methods and fields to complete the program. An example of how to test the `ArrayData` class is given below.

```
public class TestArrayData
{
	public static void main( String args[] )
	{
		ArrayData s = new ArrayData();

		s.populate( 5, 10 );
		s.sum();
		s.print();

		s.product( 2, 4 );
		s.sum();
		s.print();

		s.occurrence( 2 );
		s.print();

		s.print( 4, 2 );
	} // main
} // TestArrayData
```

To give you an idea of what the arrays would look like, assume that the total number of rows is 3 and the columns is 3. The arrays will have the following format.



If the rows sums must be equal to 5, the grid array may look like the following.



The row sums are now equal to 5 and the column sums adjusted to their new values. **Note that this is an example. You must follow the instructions in the table above.**

To generate a simple random value, use the following code snippet.

```
import java.util.Random;

public class ArrayData
{
	private Random rand = new Random();

	public int getRandNumber( int maxNum )
	{
		// Get a number between 0 and maxNum (excluding maxNum).
		return( rand.nextInt( maxNum ) );
	}
} // ArrayData
```

The `nextInt` method returns an integer value that lies between 0 and `maxNum` (including 0 and excluding `maxNum`).

In-Code Documentation

In-code documentation refers to the commenting of code to ensure that the functionality and purpose of the code is understood. The minimum comments that should be included are:

- At the top of each .java file, there should be a description of what the class does.
- At the top of each function/method in the .java file there should be a description of the purpose of the function, the values passed to the function and what is returned by the function.
- Next to each field there should be a description of its purpose.
- Utilise sensible names for classes, fields and functions/methods that reflect their purpose.

Grading Rubric

This project is worth 10% of the total course mark. The grading scheme is given below.

Area	Excellent	Good	Average	Unsatisfactory
Data structures and algorithms chosen (30)	Excellent use of the appropriate data structures and algorithms. Excellent coding practices have been used with no redundant or unnecessary code. Algorithms are efficient. No unnecessary data is used. **(24-30 marks)**	Good use of the appropriate data structures and algorithms. Good coding practices have been used with little redundant or unnecessary code. Algorithms are mostly efficient. Small amounts of unnecessary data are used. **(16-23 marks)**	Average use of the appropriate data structures and algorithms. Considerable redundant or unnecessary code is present. Algorithms are generally not efficient. Significant amounts of unnecessary data are used. **(8-15 marks)**	Poor use of the appropriate data structures and algorithms. Redundant and unnecessary code is largely present. Algorithms are not efficient. Most of the data that is used is unnecessary. **(0-7 marks)**
Implementation and Execution (30)	Over 80% of the required functionality has been implemented and executes correctly. **(24-30 marks)**	Between 50% - 80% of the required functionality has been implemented and executes correctly. **(16-23 marks)**	Between 25%-49% of the required functionality has been implemented and executes correctly. **(8-15 marks)**	Less than 25% of the required functionality has been implemented and executes correctly. **(0-7 marks)**
Compiling (10)	The software compiles with no errors and less than 3 warnings. **(10 marks)**	The software compiles with 1-6 errors and 4-6 warnings. **(7-9 marks)**	The software compiles with 7-12 errors and 7-10 warnings. **(4-6 marks)**	The software compiles with greater than 12 errors and greater than 10 warnings. **(0-3 marks)**
Naming Conventions (5)	Over 80% of the code uses the correct naming conventions. **(5 marks)**	Between 51%-80% of the code uses the correct naming conventions. **(4 marks)**	Between 26%-50% of the code uses the correct naming conventions. **(3 marks)**	Between 0%-25% of the code uses the correct naming conventions. **(0-2 marks)**
In-Code Documentation (5)	Over 80% of the code has been fully documented with comments	Between 50%-80% of the code has been fully documented	Between 25%-49% of the code has been fully documented	Less than 25% of the code has been fully documented

	laid out in a readable and understandable form. **(5 marks)**	with comments mostly laid out in a readable and understandable form. **(4 marks)**	with some of the comments laid out in a readable and understandable form. **(3 marks)**	with very few of the comments laid out in a readable and understandable form. **(0-2 marks)**
Total (80)				

Deliverables

The final deadline for this assignment is 10th February 2023 at midnight. It should be submitted on eLearninig.

The deliverables are:

- The Java code in the .java files. Do not include the .class files.