CMSC 203, Assignment 5

Spring 2018

**Concepts Utilized in this Project**

* Creating classes based on Javadoc
* Two Dimensional Arrays
* Passing two dimensional arrays to and from methods
* Creating a Utility class (static methods)
* JUnit testing
* Reading from a file
* Writing to a file
* Using methods of the utility class within an existing GUI driver class
  + Must follow Javadoc to implement correctly

**Overview**

When GUI application starts (provided), user is shown display of Store Names and Item Names

User selects *Load Sales Data* to select the file containing the sales data. The application then displays the sales for each store and each item as well as the totals for the store and the totals for the item. The store with the highest sales for each item will be highlighted.

*Exit* will exit the application.

## Specifications & Requirements

Create a utility class that manipulates a **two-dimensional** ragged array of doubles. It will accommodate positive and negative numbers. Follow the Javadoc provided.

This utility class will be used with an existing GUI class to create a sales report.

Testing of the utility class will be done with the JUnit tests and the GUI class provided for you.

**Utility class**

The class *TwoDimRaggedArrayUtility* will follow the provided Javadoc and will contain the following methods:

1. Method **readFile** – pass in a file and return a two-dimensional ragged array of doubles
2. Method **writeToFile** – pass in a two-dimensional ragged array of doubles and a file, and writes the ragged array into the file. Each row is on a separate line and each double is separated by a space.
3. Method **getTotal** – pass in a two-dimensional ragged array of doubles and returns the total of the elements in the array.
4. Method **getAverage** – pass in a two-dimensional ragged array of doubles and returns the average of the elements in the array (total/num of elements).
5. Method **getRowTotal** – pass in a two-dimensional ragged array of doubles and a row index and returns the total of that row. Row index 0 is the first row in the array.
6. Method **getColumnTotal** - pass in a two-dimensional ragged array of doubles and a column index and returns the total of that column. Column index 0 is the first column in the array. If a row doesn’t contain that column, it is not an error, that row will not participate in this method.
7. Method **getHighestInRow** - pass in a two-dimensional ragged array of doubles and a row index and returns the largest element in that row. Row index 0 is the first row in the array.
8. Method **getLowestInRow** - a two-dimensional ragged array of doubles and a row index and returns the smallest element in that row. Row index 0 is the first row in the array.
9. Method **getHighestInColumn** - pass in a two-dimensional ragged array of doubles and a column index and returns the largest element in that column. Column index 0 is the first column in the array. If a row doesn’t contain that column, it is not an error, that row will not participate in this method.
10. Method **getLowestInColumn** - pass in a two-dimensional ragged array of doubles and a column index and returns the smallest element in that column. Column index 0 is the first column in the array. If a row doesn’t contain that column, it is not an error, that row will not participate in this method.
11. Method **getHighestInArray** - pass in a two-dimensional ragged array of doubles and returns the largest element in the array.
12. Method **getLowestInArray** - pass in a two-dimensional ragged array of doubles and returns the smallest element in the array.

**GUI Application – provided for you**

1. Uses methods of *TwoDimRaggedArrayUtility*
2. When the *Load Sales Data* button is selected the sales data is read from a file and displayed on the screen with the sales data as well as the totals for each store and the totals for each item. The largest sales for each item is highlighted.
3. The file contains a row for each store and each double in the row is separated by a space
4. Student must provide two additional input files and a screenshot of the results of each. Each file will have 6 rows and up to 6 numbers on each row. They must represent ragged arrays.

**JUnit Test**

1. Contains test for each of the methods of TwoDimRaggedArrayUtility
2. Student will fill in values for a new dataset – called dataSetSTUDENT
3. Student will implement the STUDENT test files using the dataSetSTUDENT.

**Deliverables / Submissions:**

Week 1: Initial Design – a UML class diagram with algorithm (pseudo-code)

Week 2: Submit a compressed file containing the follow (see below): The Java application (it must compile and run correctly); Javadoc files in a directory; a write-up as specified below. Be sure to review the provided project rubric to understand project expectations. The write-up will include:

* Test Cases (These are the ones you will use in your STUDENT test methods in the JUnit test)
  + Prepare a test table with a list of test cases (expected versus actual results) that you are testing the application with

**Deliverable format:** The above deliverables will be packaged as follows. Two compressed files in the following formats:

LastNameFirstName\_Assignment5.zip [a compressed file containing the following]

* + doc [a directory] *contains your Javadoc (.html) files plus supporting files*
  + Write-up with test cases
  + src [a directory] *contains your (.java) files*
    - File1.java (example)
    - File2.java (example)
    - File\_Test.java (example)

LastNameFirstName\_Assignment5\_**Moss**.zip [a compressed file containing only . java files]

*NO FOLDERS!!*

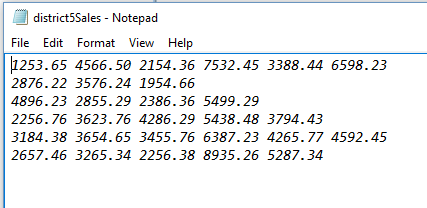
File1.java (example)

File2.java (example)

When application starts:



File containing sales data:



Result after selecting Load Sales Data:



**Grading Rubric**

**CMSC203 Project #5**

Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Overview:**

There are two parts to the rubric. First, the project is graded on whether it compiles, whether it runs without errors, and whether it satisfies the specifications. These points add up to 100. Second, the score is decremented if various requirements are not met, e.g., no Test Cases, no UML diagram, uses constructs that are not allowed, etc

**PROGRAMMING** (100 pts)

Compiles 40 pts \_\_\_\_\_

Accuracy

Public tests – JUnit test given to you 10 pts \_\_\_\_\_

Your JUnit Tests – Student tests 10 pts \_\_\_\_\_

Private tests – other tests run by the instructor 10 pts \_\_\_\_\_

Execution: runs without errors (either run-time or logic errors) 30 pts \_\_\_\_\_

Possible Sub-total 100 pts \_\_\_\_\_

**REQUIREMENTS** (Subtracts from Programming total)

**Documentation:**

Javadoc was not provided - 6 pts \_\_\_\_\_

Documentation within source code was missing or incorrect - 5 pts \_\_\_\_\_

Description of what class does was missing

Author’s Name, @author, was missing

Methods not commented properly using Javadoc @param, @return

Test cases were not provided - 5 pts \_\_\_\_\_

**Testing:**

STUDENT test cases not implemented in JUnit test - 5 pts \_\_\_\_\_

Two additional files containing a ragged array for the GUI to read from and - 5 pts \_\_\_\_\_

the screenshot showing the results of each Test File not provided

**Programming Style:**

Incorrect use of indentation, statements, structures - 4 pts \_\_\_\_\_

**Design:**

Javadoc was not followed - 6 pts \_\_\_\_\_

Doesn’t use a 2-dim array - 6 pts \_\_\_\_\_

Doesn’t work as a ragged array - 8 pts \_\_\_\_\_

Doesn’t work correctly with positive and negative numbers - 5 pts \_\_\_\_\_

Methods are not implemented correctly - 5 pts \_\_\_\_\_

Possible decrements: -60 pts \_\_\_\_\_

Possible total grade: 100 pts \_\_\_\_\_