**Assignment 5**

Josue Ponce

Montgomery College

4/2/2018

Author’s Note

This project report was prepared for CMSC 203 CRN #30672, taught by professor Ahmed Tarek

**Table of Contents**

TwoDimRaggedArrayUtility.Java Pseudo Code………………………………………………….2

Copy of TwoDimRaggedArrayUtility.Java Code…………………………………………………8

Test Case 1: Hand Calculated Results Using Original Input File………………………..………14

Test Case 1: Hand Calculated Results Using Original Input File Program Output……………...15

Test Case 2: Hand Calculated Results Using Custom Made Text File…………………………..18

Test Case 2: Program’s Sample Output With Custom Text File…………………………...……20

Test Case 3: Hand Calculated Results With Fully Customized Text File……………………….21

Test Case 3: Program’s Sample Output With New Custom Text File…………………………..23

TwoDimRaggedArrayUtility.Java Using Junit Test Program…………………………………..24

Copy of Junit Test Code…………………………………………………………………………25

**TwoDimRaggedArrayUtility.Java Pseudo Code**

Import Java Scanner to scan files

Import Java file class.

public final class TwoDimRaggedArrayUtility{

public static double[][] readFile(File file){

double[][] data = null

try{

Scanner in = new Scanner(file);

Initialize integer rows and set it equal to 0.

while(in.hasNextLine()){

++rows;

in.nextLine();

}END WHILE

in.close();

in = new Scanner(file);

String line;

data = new double[rows][];

for(int i = 0; i < data.length; i++){

line = in.nextLine();

String[] nums = line.split(" ");

data[i] = new double[nums.length];

for(int j = 0; j < data[i].length; j++){

data[i][j] = Double.valueOf(nums[j]);

}END FOR

}END FOR

In.close() to close scanner

}END TRY

catch (IOException e) {

e.printStackTrace();

}END CATCH

Return data

}END READFILE METHOD

public static void writeToFile(double[][] data, File outputFile){

try{

BufferedWriter writeData = new BufferedWriter(new FileWriter(outputFile));

for(int i = 0; i < data.length; i++){

for(int j = 0; j < data[i].length; j++){

writeData.write(data[i][j] + " ");

}END FOR

Declare a newline within the copied text file for rows and columns.

}END FOR

Close writeData which is the data being written to copy ragged array to new text file.

}END TRY

catch (IOException e) {

e.printStackTrace();

}END CATCH

}END writeToFile method

public static double getTotal(double[][] data){

initialize double total and set it equal to 0.

for(int i = 0; i < data.length; i++){

for(int j = 0; j < data[i].length; j++){

total += data[i][j];

}END FOR

}END FOR

return total;

}END getTotal METHOD.

public static double getAverage(double[][] data){

Initialize double total and set it equal to 0.

Initialize integer count and set it equal to 0.

for(int i =0; i<data.length;i++){

for(int j=0;j<data[i].length;j++){

total += data[i][j];

count++;

}END FOR

}END FOR

Return (total/count);

} END getAverage METHOD

public static double getRowTotal(double[][] data, int row){

Initialize double rowTotal and set it equal to 0.

for(int i = 0; i < data.length; i++){

if(col < data[i].length)

rowTotal += data[i][col];

}END FOR

return rowTotal;

}END getRowTotal METHOD

public static double getColumnTotal(double[][] data, int col){

initialize double columnTotal and set it equal to 0.

for(int i = 0; i < data.length; i++){

if(col < data[i].length)

columnTotal += data[i][col];

}END FOR

return columnTotal;

}END getColumnTotal METHOD

public static double getHighestInRow(double[][] data, int row){

Initialize double max and set it equal to 0.

for(int i =0; i < data[row].length;i++){

if(max < data[row][i])

max = data[row][i];

}END FOR

return max;

}END getHighestInRow MEHTOD

public static double getLowestInRow(double[][] data, int row){

initialize double min and set it equal to Double.MAX\_VALUE.

for(int i =0; i < data[row].length;i++){

if(min > data[row][i]){

min = data[row][i];

}END IF

}END FOR

return min;

}END getLowestInRow Method.

public static double getHighestInColumn(double[][] data, int col){

initialize double max and set it equal to 0.

for(int i = 0; i < data.length; i++){

if(data[i].length>col){

if(data[i][col] > max){

max = data[i][col];

}END IF

}END IF

}END FOR

return max;

}END getHighestInColumn MEHTOD

public static double getLowestInColumn(double[][] data, int col){

initialize double min and set it equal to Double.MAX\_VALUE.

for(int i = 0; i < data.length; i++){

if(data[i].length>col){

if(data[i][col] < min){

min = data[i][col];

}END IF

}END IF

}END FOR

return min;

}END getLowestInColum METHOD

public static double getHighestInArray(double[][] data){

initialize double max and set it equal to data[0][0]

for(int i = 0; i < data.length;i++){

for(int j = 0; j<data[i].length;j++){

if(data[i][j] > max)

max = data[i][j];

}END IF

}END FOR

}END FOR

return max;

}END getHighestInArray METHOD.

public static double getLowestInArray(double[][] data){

Initialize double min and set it equal to data[0][0].

for(int i = 0; i <data.length;i++){

for(int j = 0; j<data[i].length;j++){

if(data[i][j] < min)

min = data[i][j];

}END IF

}END FOR

}END FOR

return min;

}END getLowestInArray METHOD

**Copy of TwoDimRaggedArrayUtility.Java code**

**import** **java.io.\***;

**import** **java.util.Scanner**;

/\*\*

\* This utility works with 2 dimensional ragged arrays with a maximum of 10 rows and 10 columns

\* This utility also works with negative and positive numbers

\* There are no private data members in the utility class!

\* @Author Josue Ponce

\* @Version 1.0

\* @Date 4/3/18

\*

\*/

**public** **final** **class** **TwoDimRaggedArrayUtility** {

/\*\*

\* Reads data from the text file.

\* @param file - the file to read from.

\* @return data - which is a two dimensional ragged (depending on data) array of doubles if the file is not empty, returns a null if file is empty

\*/

**public** **static** **double**[][] **readFile**(File file) {

**double**[][] data = **null**;

**try** {

Scanner in = **new** Scanner(file);

**int** rows = **0**;

**while** ( in .hasNextLine()) {

++rows; in .nextLine();

} in .close(); in = **new** Scanner(file);

String line;

data = **new** **double**[rows][];

**for** (**int** i = **0**; i < data.length; i++) {

line = in .nextLine();

String[] nums = line.split(" ");

data[i] = **new** **double**[nums.length];

**for** (**int** j = **0**; j < data[i].length; j++) {

data[i][j] = Double.valueOf(nums[j]);

}

} in .close();

}

**catch** (IOException e) {

e.printStackTrace();

}

**return** data;

}

/\*\*

\* Writes the ragged array of doubles into a file.

\* @param data - two dimensional ragged array of doubles

\* @param outputFile - The output file that will be written.

\*/

**public** **static** **void** **writeToFile**(**double**[][] data, File outputFile) {

**try** {

BufferedWriter writeData = **new** BufferedWriter(**new** FileWriter(outputFile));

**for** (**int** i = **0**; i < data.length; i++) {

**for** (**int** j = **0**; j < data[i].length; j++) {

writeData.write(data[i][j] + " ");

}

writeData.newLine();

}

writeData.close();

}

**catch** (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Returns the total of all the elements of the two dimensional array

\* @param data- The two dimensional arrays that will be calculated to get the total.

\* @return total - The sum of all the elements in the two dimensional array

\*/

**public** **static** **double** **getTotal**(**double**[][] data) {

**double** total = **0**;

**for** (**int** i = **0**; i < data.length; i++) {

**for** (**int** j = **0**; j < data[i].length; j++) {

total += data[i][j];

}

}

**return** total;

}

/\*\*

\* Returns the average of the elements in the two dimensional array

\* @param data - The two dimensional arrays that will be calculated to get the average.

\* @return (total/count) - the average of the elements in the two dimensional array

\*/

**public** **static** **double** **getAverage**(**double**[][] data) {

**double** total = **0**;

**int** count = **0**;

**for** (**int** i = **0**; i < data.length; i++) {

**for** (**int** j = **0**; j < data[i].length; j++) {

total += data[i][j];

count++;

}

}

**return** (total / count);

}

/\*\*

\* Returns the total of the selected row in the two dimensional array index 0 refers to the first row

\* @param data - The two dimensional array

\* @param row - The row index to take the total of (0 refers to the first row)

\* @return rowTotal - The total of the row

\*/

**public** **static** **double** **getRowTotal**(**double**[][] data, **int** row) {

**double** rowTotal = **0**;

**for** (**int** i = **0**; i < data[row].length; i++) {

rowTotal += data[row][i];

}

**return** rowTotal;

}

/\*\*

\* Returns the total of the selected column in the two dimensional array index 0 refers to the first column.

\* If a row in the two dimensional array doesn't have this column index,

\* it is not an error, it doesn't participate in this method.

\* @param data - The two dimensional array

\* @param col - the column index to take the total of (0 refers to the first column)

\* @return columnTotal - The total of the column

\*/

**public** **static** **double** **getColumnTotal**(**double**[][] data, **int** col) {

**double** columnTotal = **0**;

**for** (**int** i = **0**; i < data.length; i++) {

**if** (col < data[i].length)

columnTotal += data[i][col];

}

**return** columnTotal;

}

/\*\*

\* Returns the largest element of the selected row in the two dimensional array index 0 refers to the first row.

\* @param data - the two dimensional array

\* @param row - the row index to find the largest element of (0 refers to the first row)

\* @return max - the largest element of the row

\*/

**public** **static** **double** **getHighestInRow**(**double**[][] data, **int** row) {

**double** max = **0**;

**for** (**int** i = **0**; i < data[row].length; i++) {

**if** (max < data[row][i])

max = data[row][i];

}

**return** max;

}

/\*\*

\* Returns the smallest element of the selected row in the two dimensional array index 0 refers to the first row.

\* @param data - the two dimensional array

\* @param row - the row index to find the smallest element of (0 refers to the first row)

\* @return min - the smallest element of the row

\*/

**public** **static** **double** **getLowestInRow**(**double**[][] data, **int** row) {

**double** min = Double.MAX\_VALUE;

**for** (**int** i = **0**; i < data[row].length; i++) {

**if** (min > data[row][i])

min = data[row][i];

}

**return** min;

}

/\*\*

\* Returns the largest element of the selected column in the two dimensional array index 0 refers to the first column.

\* @param data - The two dimensional array

\* @param col - The column index to find the largest element of (0 refers to the first column)

\* @return max - The largest element of the column

\*/

**public** **static** **double** **getHighestInColumn**(**double**[][] data, **int** col) {

**double** max = **0**;

**for** (**int** i = **0**; i < data.length; i++) {

**if** (data[i].length > col) {

**if** (data[i][col] > max) {

max = data[i][col];

}

}

}

**return** max;

}

/\*\*

\* Returns the smallest element of the selected column in the two dimensional array index 0 refers to the first column.

\* @param data - The two dimensional array

\* @param col - The column index to find the smallest element of (0 refers to the first column)

\* @return min - The smallest element of the column

\*/

**public** **static** **double** **getLowestInColumn**(**double**[][] data, **int** col) {

**double** min = Double.MAX\_VALUE;

**for** (**int** i = **0**; i < data.length; i++) {

**if** (data[i].length > col) {

**if** (data[i][col] < min) {

min = data[i][col];

}

}

}

**return** min;

}

/\*\*

\* Returns the largest element in the two dimensional array

\* @param data - The two dimensional array

\* @return max - The largest element in the two dimensional array

\*/

**public** **static** **double** **getHighestInArray**(**double**[][] data) {

**double** max = data[**0**][**0**];

**for** (**int** i = **0**; i < data.length; i++) {

**for** (**int** j = **0**; j < data[i].length; j++) {

**if** (data[i][j] > max)

max = data[i][j];

}

}

**return** max;

}

/\*\*

\* Returns the smallest element in the two dimensional array

\* @param data - The two dimensional array

\* @return min - The smallest element in the two dimensional array

\*/

**public** **static** **double** **getLowestInArray**(**double**[][] data) {

**double** min = data[**0**][**0**];

**for** (**int** i = **0**; i < data.length; i++) {

**for** (**int** j = **0**; j < data[i].length; j++) {

**if** (data[i][j] < min)

min = data[i][j];

}

}

**return** min;

}

}

**Test Case 1: Hand Calculated Results Using Original Input File**

Once the program was completed, the results were hand calculated to ensure that the program was free of any logical errors. The revenue for each column and row were obtained from the text file provided which was used to calculate the total amount of revenue for each row and column. The total amount of revenue for each row was added to get the total amount of revenue for each row. The columns were also added to get the total amount of revenue for each column. The table is similar to the GUI integrated in the program. Additionally, the highest and lowest amount of revenue per column have been highlight in the table down below. The highlighted sections provide a visual representation of the highest and lowest amount of revenue that is expected from the program’s output. The green highlighted sections represent the highest amount of revenue per column and the red highlighted sections represent the lowest amount of revenue expected for each column.

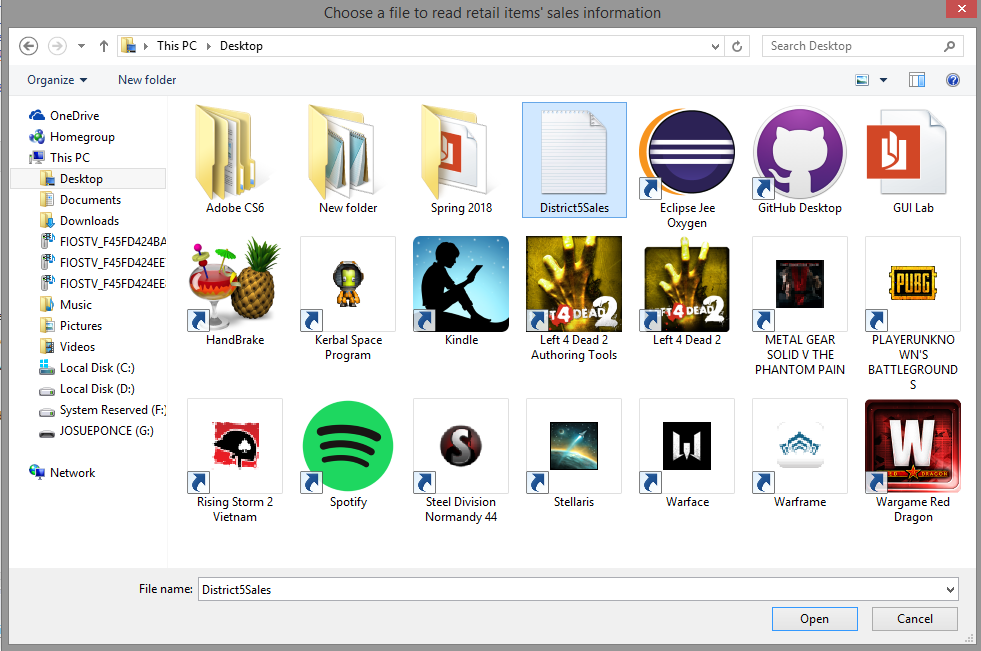
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Books** | **Tsum Tsum** | **Trading Pins** | **Star Wars** | **Lego** | **Marvel** | **Total** |
| **Emporium** | $1,253.65 | $4,566.50 | $2,154.36 | $7,532.45 | $3,388.44 | $6,598.23 | $25,493.63 |
| **World Traveler** | $2,876.22 | $3,576.24 | $1,954.66 |  |  |  | $8,407.12 |
| **Discovery Trading** | $4,896.23 | $2,855.29 | $2,386.36 | $5,499.29 |  |  | $15,637.17 |
| **Merchant of Venus** | $2,256.76 | $3,623.76 | $4,286.29 | $5,438.48 | $3,794.43 |  | $19,399.72 |
| **Once Upon a Toy** | $3,184.38 | $3,654.65 | $3,455.76 | $6,387.23 | $4,265.77 | $4,592.45 | $25,540.24 |
| **Tatooine Traders** | $2,657.46 | $3,265.34 | $2,256.38 | $8,935.26 | $5,287.34 |  | $22,401.78 |
| **Total** | $17,124.70 | $21,541.78 | $16,493.81 | $33,792.71 | $16,735.98 | $11,190.68 |  |

*Table 1.* This table represents the expected program’s output. All sales information in this table have been hand calculated to ensure that the program is free of logical errors and the total amount of revenue is mathematically correct.

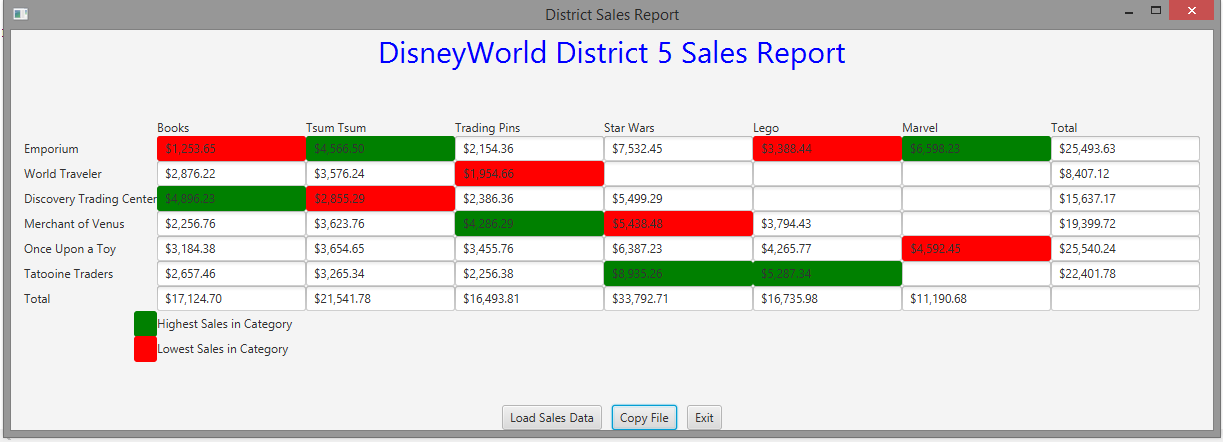
**Test Case 1: Hand Calculated Results Using Original Input File Program Output**

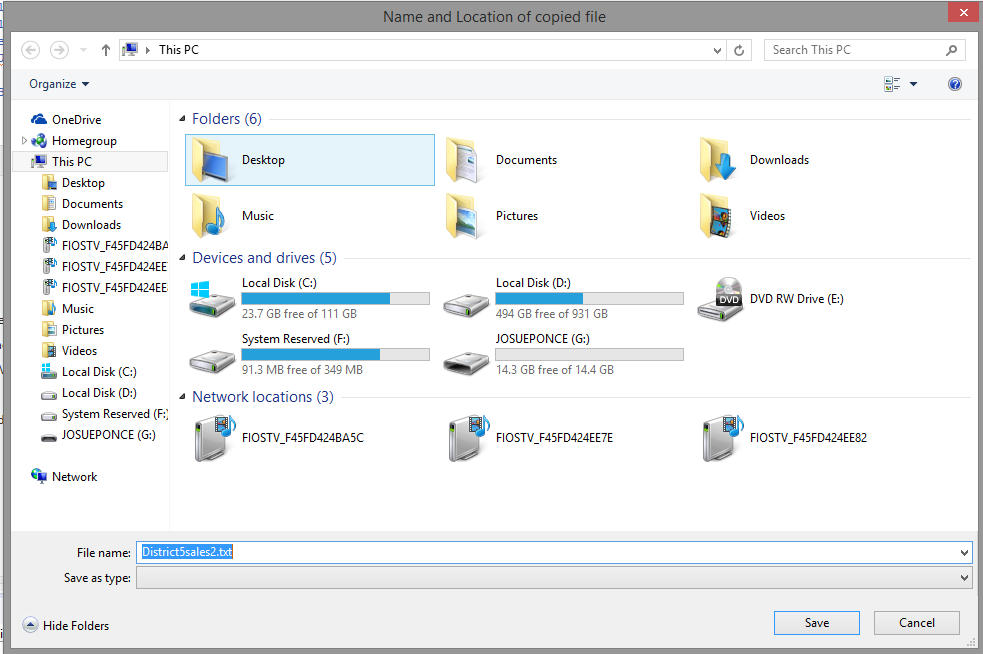
After hand calculating the results, the program was compiled and the original input text file was input into the program. All of the hand calculated results matched the program’s output which ensures that this program is free of logical errors and the total amount of revenue is mathematically correct. In addition, the program was also able to determine the lowest and highest amount of revenue per column correctly. Moreover, the original text file was copied using the program to ensure that all the data from the original text file could be written onto a new text file without error. The screenshots down below are a visual representation of the running application without error.

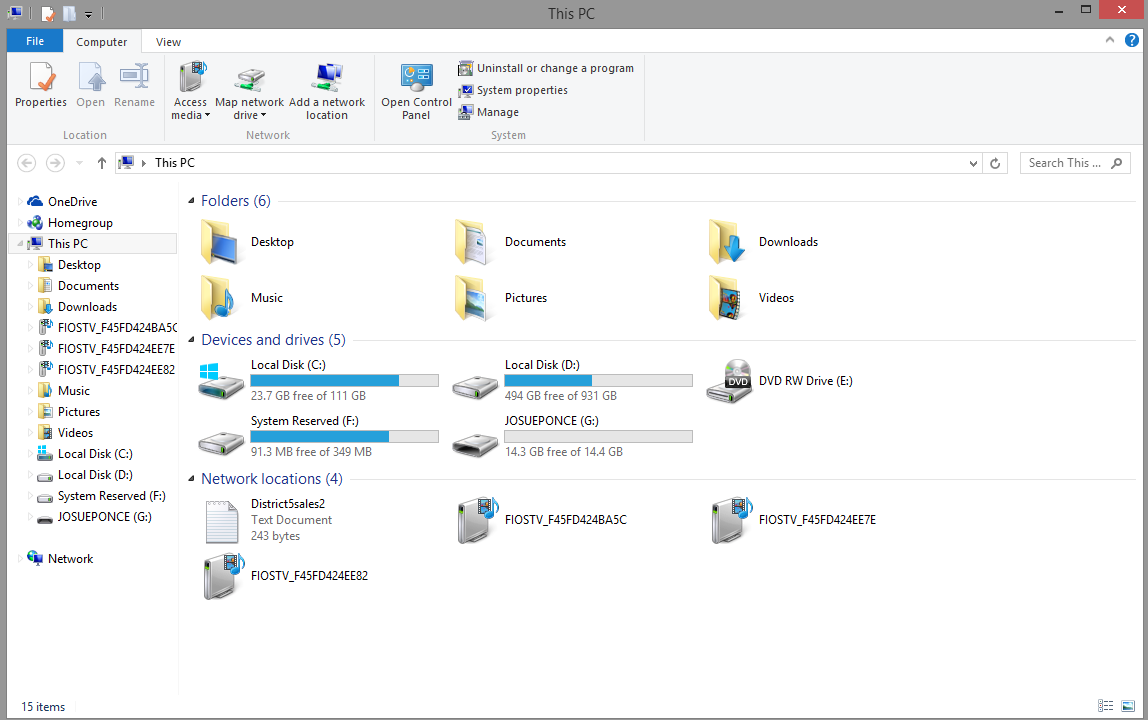


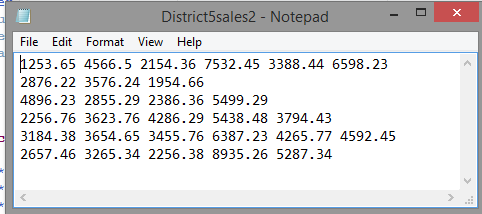






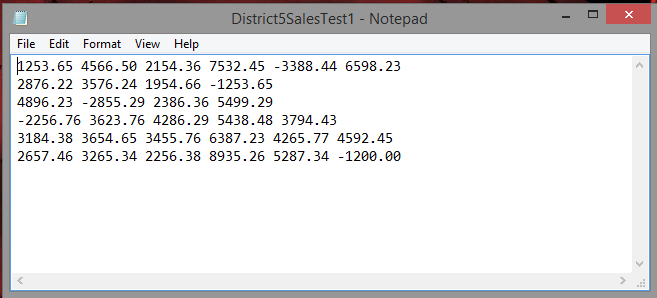






**Test Case 2: Hand Calculated Results Using Custom Made Text File**

A custom made text file was created to test if the program will calculate negative and positive numbers correctly. Figure 1 shows all the numbers from the new text file that will be tested. However, before compiling the program with the new text file, the results were first hand calculated to ensure that the program is free of logical errors and displays mathematically correct results. The table down below is a representation of the expect program’s sample output just like the first test case. The total amount of revenue for each column and row were obtained the same way they were in test case 1. Simply by adding all the revenue together to get the total amount of revenue for each column and row. Once again, the highest and lowest amount of revenue per column have been highlight in the table down below. The highlighted sections provide a visual representation of the highest and lowest amount of revenue that is expected from the program’s output. The green highlighted sections represent the highest amount of revenue per column and the red highlighted sections represent the lowest amount of revenue expected for each column.



*Figure 1.* Screenshot displays new text file that will be tested in the program.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Books** | **Tsum Tsum** | **Trading Pins** | **Star Wars** | **Lego** | **Marvel** | **Total** |
| **Emporium** | $1,253.65 | $4,566.50 | $2,154.36 | $7,532.45 | ($3,388.44) | $6,598.23 | $18,716.75 |
| **World Traveler** | $2,876.22 | $3,576.24 | $1,954.66 | ($1,253.65) |  |  | $7,153.47 |
| **Discovery Trading** | $4,896.23 | ($2,855.29) | $2,386.36 | $5,499.29 |  |  | $9,926.59 |
| **Merchant of Venus** | ($2,256.76) | $3,623.76 | $4,286.29 | $5,438.48 | $3,794.43 |  | $14,886.20 |
| **Once Upon a Toy** | $3,184.38 | $3,654.65 | $3,455.76 | $6,387.23 | $4,265.77 | $4,592.45 | $25,540.24 |
| **Tatooine Traders** | $2,657.46 | $3,265.34 | $2,256.38 | $8,935.26 | $5,287.34 | ($1,200.00) | $21,201.78 |
| **Total** | $12,611.18 | $15,831.20 | $16,493.81 | $32,539.06 | $9,959.10 | $9,990.68 |  |

*Table 1.* This table represents the expected program’s output. All sales information in this table have been hand calculated to ensure that the program is free of logical errors and the total amount of revenue is mathematically correct. The numbers in parenthesis represent a negative number or in this case negative revenue.

**Test Case 2: Program’s Sample Output With Custom Text File**

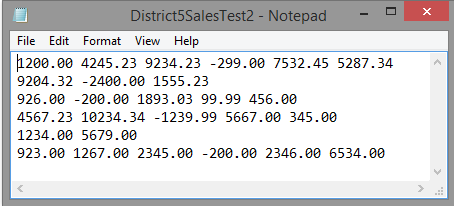
The program’s output matched the expected output in the hand calculated results. The new text file with negative numbers was read by the program and calculated the results correctly. Thus, the test was successful because the program was able to calculate both positive and negative numbers correctly and free of mathematical errors. The screenshot down below displays the running program with the correct results.



*Note.* Screenshot of the program’s results with the new text file.

**Test Case 3: Hand Calculated Results With Fully Customized Text File**

A new text file was created with complete different numbers. The numbers mixed into the new text file includes both negative and positive numbers. Figure 1 shows all the numbers from the new text file that will be tested after hand calculating the results . Once again, the highest and lowest amount of revenue per column have been highlight in the table down below. The highlighted sections provide a visual representation of the highest and lowest amount of revenue that is expected from the program’s output. The green highlighted sections represent the highest amount of revenue per column and the red highlighted sections represent the lowest amount of revenue expected for each column.



*Figure 1.* New custom text file that will be tested in the program.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Books** | **Tsum Tsum** | **Trading Pins** | **Star Wars** | **Lego** | **Marvel** | **Total** |
| **Emporium** | $1,200.00 | $4,245.23 | $9,234.23 | ($299.00) | $7,532.45 | $5,287.34 | $27,200.25 |
| **World Traveler** | $9,204.32 | ($2,400.00) | $1,555.23 |  |  |  | $8,359.55 |
| **Discovery Trading** | $926.00 | ($200.00) | $1,893.03 | $99.99 | $456.00 |  | $3,175.02 |
| **Merchant of Venus** | $4,567.23 | $10,234.34 | ($1,239.99) | $5,667.00 | $345.00 |  | $19,573.58 |
| **Once Upon a Toy** | $1,234.00 | $5,679.00 |  |  |  |  | $6,913.00 |
| **Tatooine Traders** | $923.00 | $1,267.00 | $2,345.00 | ($200.00) | $2,346.00 | $6,534.00 | $13,215.00 |
| **Total** | $18,054.55 | $18,825.57 | $13,787.50 | $5,267.99 | $10,679.45 | $11,821.34 |  |

*Table 1.* This table represents the expected program’s output. All sales information in this table have been hand calculated to ensure that the program is free of logical errors and the total amount of revenue is mathematically correct. The numbers in parenthesis represent a negative number or in this case negative revenue.

**Test Case 3: Program’s Sample Output With New Custom Text File**

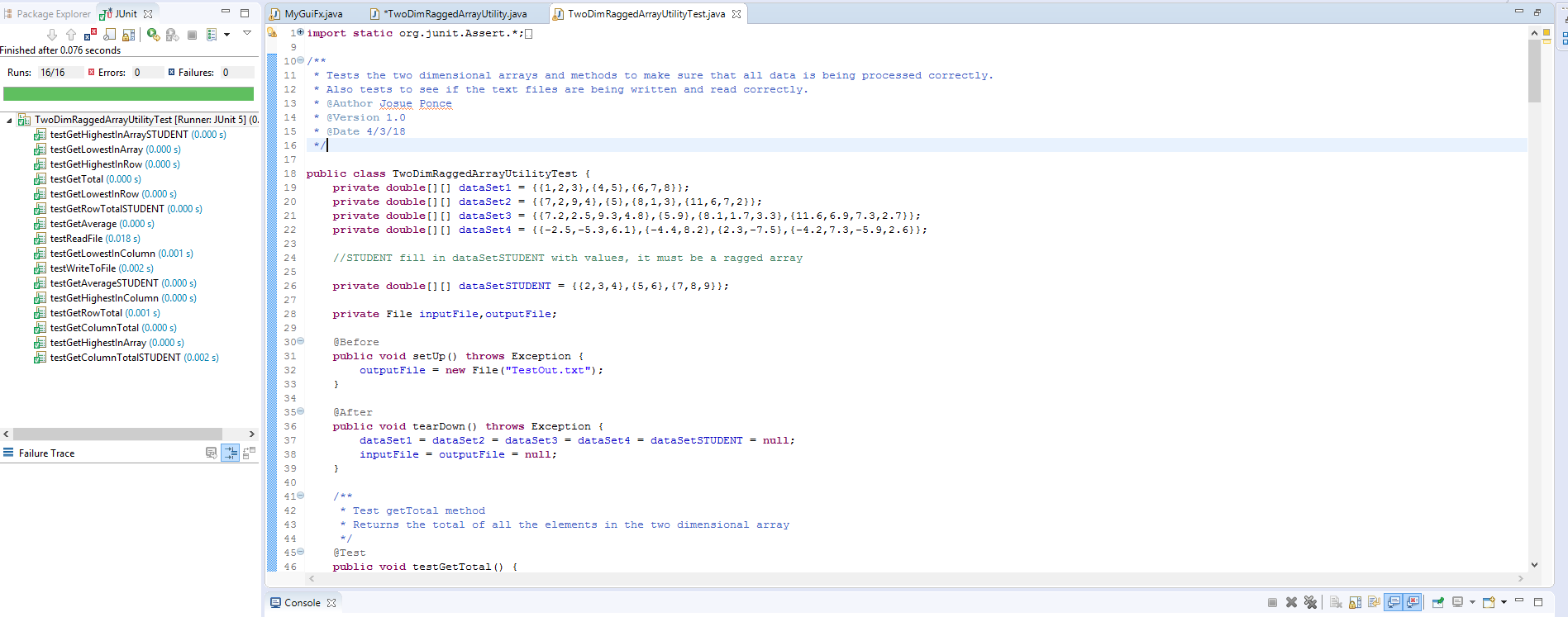
The program’s output matched the expected output in the hand calculated results. The new text file with negative numbers was read by the program and calculated the results correctly. Thus, the test was successful because the program was able to calculate both positive and negative numbers correctly and free of mathematical errors. The new text file worked as intended and the program was able to read this new text file without error. The screenshot down below displays the running program with the correct results.



*Note.* Program’s sample output with the newly created text file.

**TwoDimRaggedArrayUtility.Java Using Junit Test Program**

A test program was made to ensure that all the methods implemented into the application worked as intended. The test program also tests the two dimensional arrays and methods to make sure that all data is being processed correctly. The test program also ensures that the text files are being written and read correctly by the program being tested. After running the test program on the application being tested, all the results returned green which means that all methods have been implemented correctly. Additionally, the positive results also show that the ragged arrays are processing data correctly in the application being tested. Thus, the program passed all tests and works as intended. The screenshot down below shows the results of the tested application.



*Note.* Screenshot displays positive results after a Junit test was conducted on TwoDimRaggedArrayUtility.Java

**Copy of Junit Test Code**

**import** **static** org.junit.Assert.\*;

**import** **java.io.File**;

**import** **java.io.FileNotFoundException**;

**import** **java.io.PrintWriter**;

**import** **java.util.Scanner**;

**import** **org.junit.After**;

**import** **org.junit.Before**;

**import** **org.junit.Test**;

/\*\*

\* Tests the two dimensional arrays and methods to make sure that all data is being processed correctly.

\* Also tests to see if the text files are being written and read correctly.

\* @Author Josue Ponce

\* @Version 1.0

\* @Date 4/3/18

\*/

**public** **class** **TwoDimRaggedArrayUtilityTest** {

**private** **double**[][] dataSet1 = {{**1**,**2**,**3**},{**4**,**5**},{**6**,**7**,**8**}};

**private** **double**[][] dataSet2 = {{**7**,**2**,**9**,**4**},{**5**},{**8**,**1**,**3**},{**11**,**6**,**7**,**2**}};

**private** **double**[][] dataSet3 = {{**7.2**,**2.5**,**9.3**,**4.8**},{**5.9**},{**8.1**,**1.7**,**3.3**},{**11.6**,**6.9**,**7.3**,**2.7**}};

**private** **double**[][] dataSet4 = {{-**2.5**,-**5.3**,**6.1**},{-**4.4**,**8.2**},{**2.3**,-**7.5**},{-**4.2**,**7.3**,-**5.9**,**2.6**}};

//STUDENT fill in dataSetSTUDENT with values, it must be a ragged array

**private** **double**[][] dataSetSTUDENT = {{**2**,**3**,**4**},{**5**,**6**},{**7**,**8**,**9**}};

**private** File inputFile,outputFile;

**@Before**

**public** **void** **setUp**() **throws** Exception {

outputFile = **new** File("TestOut.txt");

}

**@After**

**public** **void** **tearDown**() **throws** Exception {

dataSet1 = dataSet2 = dataSet3 = dataSet4 = dataSetSTUDENT = **null**;

inputFile = outputFile = **null**;

}

/\*\*

\* Test getTotal method

\* Returns the total of all the elements in the two dimensional array

\*/

**@Test**

**public** **void** **testGetTotal**() {

assertEquals(**36.0**,TwoDimRaggedArrayUtility.getTotal(dataSet1),.**001**);

assertEquals(**65.0**,TwoDimRaggedArrayUtility.getTotal(dataSet2),.**001**);

assertEquals(**71.3**,TwoDimRaggedArrayUtility.getTotal(dataSet3),.**001**);

assertEquals(-**3.3**,TwoDimRaggedArrayUtility.getTotal(dataSet4),.**001**);

}

/\*\*

\* Test getAverage method

\* Returns the average of all the elements in the two dimensional array

\*/

**@Test**

**public** **void** **testGetAverage**() {

assertEquals(**4.5**,TwoDimRaggedArrayUtility.getAverage(dataSet1),.**001**);

assertEquals(**5.417**,TwoDimRaggedArrayUtility.getAverage(dataSet2),.**001**);

assertEquals(**5.942**,TwoDimRaggedArrayUtility.getAverage(dataSet3),.**001**);

assertEquals(-.**3**,TwoDimRaggedArrayUtility.getAverage(dataSet4),.**001**);

}

/\*\*

\* Student test for getAverage method

\* Use the dataSetSTUDENT

\* Returns the average of all the elements in the two dimensional array

\*/

**@Test**

**public** **void** **testGetAverageSTUDENT**() {

assertEquals(**5.5**,TwoDimRaggedArrayUtility.getAverage(dataSetSTUDENT),.**001**);

}

/\*\*

\* Test getRowTotal method

\* Returns the total of all the elements of the row.

\* Row 0 refers to the first row in the two dimensional array

\*/

**@Test**

**public** **void** **testGetRowTotal**() {

assertEquals(**9.0**,TwoDimRaggedArrayUtility.getRowTotal(dataSet1,**1**),.**001**);

assertEquals(**5.0**,TwoDimRaggedArrayUtility.getRowTotal(dataSet2,**1**),.**001**);

assertEquals(**22.0**,TwoDimRaggedArrayUtility.getRowTotal(dataSet2,**0**),.**001**);

assertEquals(**28.5**,TwoDimRaggedArrayUtility.getRowTotal(dataSet3,**3**),.**001**);

assertEquals(**5.9**,TwoDimRaggedArrayUtility.getRowTotal(dataSet3,**1**),.**001**);

assertEquals(**3.8**,TwoDimRaggedArrayUtility.getRowTotal(dataSet4,**1**),.**001**);

assertEquals(-.**2**,TwoDimRaggedArrayUtility.getRowTotal(dataSet4,**3**),.**001**);

}

/\*\*

\* StudenttTest for getRowTotal method

\* Use the dataSetSTUDENT

\* Returns the total of all the elements of the row.

\* Row 0 refers to the first row in the two dimensional array

\*/

**@Test**

**public** **void** **testGetRowTotalSTUDENT**() {

assertEquals(**11.0**,TwoDimRaggedArrayUtility.getRowTotal(dataSetSTUDENT,**1**),.**001**);

}

/\*\*

\* Test getColumnTotal method

\* Returns the total of all the elements in the column. If a row in the two dimensional array

\* doesn't have this column index, it is not an error, it doesn't participate in this method.

\* Column 0 refers to the first column in the two dimensional array

\*/

**@Test**

**public** **void** **testGetColumnTotal**() {

assertEquals(**11.0**,TwoDimRaggedArrayUtility.getColumnTotal(dataSet1,**0**),.**001**);

assertEquals(**19.0**,TwoDimRaggedArrayUtility.getColumnTotal(dataSet2,**2**),.**001**);

assertEquals(**11.1**,TwoDimRaggedArrayUtility.getColumnTotal(dataSet3,**1**),.**001**);

assertEquals(-**8.8**,TwoDimRaggedArrayUtility.getColumnTotal(dataSet4,**0**),.**001**);

assertEquals(**2.7**,TwoDimRaggedArrayUtility.getColumnTotal(dataSet4,**1**),.**001**);

}

/\*\*

\* Student test for the getColumnTotal method

\* Use dataSetSTUDENT

\* Returns the total of all the elements in the column. If a row in the two dimensional array

\* doesn't have this column index, it is not an error, it doesn't participate in this method.

\* Column 0 refers to the first column in the two dimensional array

\*/

**@Test**

**public** **void** **testGetColumnTotalSTUDENT**() {

assertEquals(**17**,TwoDimRaggedArrayUtility.getColumnTotal(dataSetSTUDENT,**1**),.**001**);

}

/\*\*

\* Test getHighestInRow method

\* Returns the largest of all the elements in the row.

\* Row 0 refers to the first row in the two dimensional array

\*/

**@Test**

**public** **void** **testGetHighestInRow**() {

assertEquals(**3.0**,TwoDimRaggedArrayUtility.getHighestInRow(dataSet1,**0**),.**001**);

assertEquals(**8.0**,TwoDimRaggedArrayUtility.getHighestInRow(dataSet2,**2**),.**001**);

assertEquals(**5.9**,TwoDimRaggedArrayUtility.getHighestInRow(dataSet3,**1**),.**001**);

assertEquals(**6.1**,TwoDimRaggedArrayUtility.getHighestInRow(dataSet4,**0**),.**001**);

assertEquals(**8.2**,TwoDimRaggedArrayUtility.getHighestInRow(dataSet4,**1**),.**001**);

}

/\*\*

\* Test getLowestInRow method

\* Returns the smallest of all the elements in the row.

\* Row 0 refers to the first row in the two dimensional array

\*/

**@Test**

**public** **void** **testGetLowestInRow**() {

assertEquals(**6.0**,TwoDimRaggedArrayUtility.getLowestInRow(dataSet1,**2**),.**001**);

assertEquals(**5.0**,TwoDimRaggedArrayUtility.getLowestInRow(dataSet2,**1**),.**001**);

assertEquals(**2.5**,TwoDimRaggedArrayUtility.getLowestInRow(dataSet3,**0**),.**001**);

assertEquals(-**4.4**,TwoDimRaggedArrayUtility.getLowestInRow(dataSet4,**1**),.**001**);

assertEquals(-**7.5**,TwoDimRaggedArrayUtility.getLowestInRow(dataSet4,**2**),.**001**);

}

/\*\*

\* Test getHighestInColumn method

\* Returns the largest of all the elements in the column. If a row in the two dimensional array

\* doesn't have this column index, it is not an error, it doesn't participate in this method.

\* Column 0 refers to the first column in the two dimensional array

\*/

**@Test**

**public** **void** **testGetHighestInColumn**() {

assertEquals(**8.0**,TwoDimRaggedArrayUtility.getHighestInColumn(dataSet1,**2**),.**001**);

assertEquals(**6.0**,TwoDimRaggedArrayUtility.getHighestInColumn(dataSet2,**1**),.**001**);

assertEquals(**11.6**,TwoDimRaggedArrayUtility.getHighestInColumn(dataSet3,**0**),.**001**);

assertEquals(**8.2**,TwoDimRaggedArrayUtility.getHighestInColumn(dataSet4,**1**),.**001**);

assertEquals(**6.1**,TwoDimRaggedArrayUtility.getHighestInColumn(dataSet4,**2**),.**001**);

}

/\*\*

\* Test getLowestInColumn method

\* Returns the smallest of all the elements in the column. If a row in the two dimensional array

\* doesn't have this column index, it is not an error, it doesn't participate in this method.

\* Column 0 refers to the first column in the two dimensional array

\*/

**@Test**

**public** **void** **testGetLowestInColumn**() {

assertEquals(**2.0**,TwoDimRaggedArrayUtility.getLowestInColumn(dataSet1,**1**),.**001**);

assertEquals(**3.0**,TwoDimRaggedArrayUtility.getLowestInColumn(dataSet2,**2**),.**001**);

assertEquals(**5.9**,TwoDimRaggedArrayUtility.getLowestInColumn(dataSet3,**0**),.**001**);

assertEquals(-**4.4**,TwoDimRaggedArrayUtility.getLowestInColumn(dataSet4,**0**),.**001**);

assertEquals(-**7.5**,TwoDimRaggedArrayUtility.getLowestInColumn(dataSet4,**1**),.**001**);

}

/\*\*

\* Test getHighestInArray method

\* Returns the largest of all the elements in the two dimensional array.

\*/

**@Test**

**public** **void** **testGetHighestInArray**() {

assertEquals(**8.0**,TwoDimRaggedArrayUtility.getHighestInArray(dataSet1),.**001**);

assertEquals(**11.0**,TwoDimRaggedArrayUtility.getHighestInArray(dataSet2),.**001**);

assertEquals(**11.6**,TwoDimRaggedArrayUtility.getHighestInArray(dataSet3),.**001**);

assertEquals(**8.2**,TwoDimRaggedArrayUtility.getHighestInArray(dataSet4),.**001**);

}

/\*\*

\* Student test for the getHighestInArray method

\* Use the dataSetSTUDENT

\* Returns the largest of all the elements in the two dimensional array.

\*/

**@Test**

**public** **void** **testGetHighestInArraySTUDENT**() {

assertEquals(**9.0**,TwoDimRaggedArrayUtility.getHighestInArray(dataSetSTUDENT),.**001**);

}

/\*\*

\* Test getLowestInArray method

\* Returns the smallest of all the elements in the two dimensional array.

\*/

**@Test**

**public** **void** **testGetLowestInArray**() {

assertEquals(**1.0**,TwoDimRaggedArrayUtility.getLowestInArray(dataSet1),.**001**);

assertEquals(**1.0**,TwoDimRaggedArrayUtility.getLowestInArray(dataSet2),.**001**);

assertEquals(**1.7**,TwoDimRaggedArrayUtility.getLowestInArray(dataSet3),.**001**);

assertEquals(-**7.5**,TwoDimRaggedArrayUtility.getLowestInArray(dataSet4),.**001**);

}

/\*\*

\* Test the writeToFile method

\* write the array to the outputFile File

\* then read it back to make sure formatted correctly to read

\*

\*/

**@Test**

**public** **void** **testWriteToFile**() {

**double**[][] array=**null**;

**try** {

TwoDimRaggedArrayUtility.writeToFile(dataSet4, outputFile);

} **catch** (Exception e) {

fail("This should not have caused an Exception");

}

//now read from the output file

**try** {

array = TwoDimRaggedArrayUtility.readFile(outputFile);

assertEquals(-**2.5**, array[**0**][**0**],.**001**);

assertEquals(-**5.3**, array[**0**][**1**],.**001**);

assertEquals(**6.1**, array[**0**][**2**],.**001**);

assertEquals(-**4.4**, array[**1**][**0**],.**001**);

assertEquals(**8.2**, array[**1**][**1**],.**001**);

assertEquals(**2.3**, array[**2**][**0**],.**001**);

assertEquals(-**7.5**, array[**2**][**1**],.**001**);

assertEquals(-**4.2**, array[**3**][**0**],.**001**);

assertEquals(**7.3**, array[**3**][**1**],.**001**);

assertEquals(-**5.9**, array[**3**][**2**],.**001**);

assertEquals(**2.6**, array[**3**][**3**],.**001**);

// Due to the structure of my code I changed FileNotFoundException e to Exception e

} **catch** (Exception e) {

fail("This should not have caused an Exception");

}

}

/\*\*

\* Test the readFile method

\* reads from a file and then test that the returned two dimensional array of doubles

\* is ragged.

\*/

**@Test**

**public** **void** **testReadFile**() {

**double**[][] array=**null**;

**try** {

inputFile = **new** File("District5Sales.txt");

PrintWriter inFile = **new** PrintWriter(inputFile);

inFile.print("-2.5 -5.3 6.1\n" +

"-4.4 8.2\n" +

"2.3 -7.5\n" +

"-4.2 7.3 -5.9 2.6");

inFile.close();

array = TwoDimRaggedArrayUtility.readFile(inputFile);

assertEquals(-**2.5**, array[**0**][**0**],.**001**);

assertEquals(-**5.3**, array[**0**][**1**],.**001**);

assertEquals(**6.1**, array[**0**][**2**],.**001**);

assertEquals(-**4.4**, array[**1**][**0**],.**001**);

assertEquals(**8.2**, array[**1**][**1**],.**001**);

assertEquals(**2.3**, array[**2**][**0**],.**001**);

assertEquals(-**7.5**, array[**2**][**1**],.**001**);

assertEquals(-**4.2**, array[**3**][**0**],.**001**);

assertEquals(**7.3**, array[**3**][**1**],.**001**);

assertEquals(-**5.9**, array[**3**][**2**],.**001**);

assertEquals(**2.6**, array[**3**][**3**],.**001**);

} **catch** (FileNotFoundException e) {

fail("This should not have caused an FileNotFoundException");

}

//testing that the array is a ragged array

**try**{

assertEquals(**0.0**, array[**1**][**2**],.**001**);

fail("This should have caused a ArrayIndexOutOfBoundsException");

} **catch** (ArrayIndexOutOfBoundsException e){

assertTrue("Correctly threw ArrayIndexOutOfBoundsException", **true**);

}

**catch** (Exception e) {

fail("This should not have caused an Exception");

}

//testing that the array is a ragged array

**try**{

assertEquals(**0.0**, array[**2**][**2**],.**001**);

fail("This should have caused a ArrayIndexOutOfBoundsException");

} **catch** (ArrayIndexOutOfBoundsException e){

assertTrue("Correctly threw ArrayIndexOutOfBoundsException", **true**);

}

**catch** (Exception e) {

fail("This should not have caused an Exception");

}

}

}