**CMSC203 Assignment 5 Implementation (Documentation)**

Class: CMSC203 CRN XXXX

 Program: Assignment # 5

Instructor: Professor Grinberg: Summary of Description: Create a utility class that manipulates a **two-dimensional** ragged array of doubles. This utility class will be used to create a Sales Report for Retail District #5. It will accommodate positive and negative numbers. Follow the Javadoc provided. Create a utility class that will calculate holiday bonuses given a ragged array of doubles which represent the sales for each store in each category. It will also take in bonus amount for the store with the highest sales in a category, the bonus amount for the store with the lowest sales in a category and the bonus amount for all other stores.

 Due Date: 04/18/2022

 Integrity Pledge: I pledge that I have completed the programming assignment independently.

 I have not copied the code from a student or any source.

**Part1: Pseudo Code:** Here is a pseudo code for Assignment 4 program:

ManagementCompany.java:

**DECLARE public constructor** TwoDimRaggedArrayUtility()

DECLARE **public** **static** **double**[][] method readFile with parameter java.io.File file with **throws** java.io.FileNotFoundException

DECLARE **double** data[][] to **null**

DECLARE **integer** maxRow to 10

DECLARE **integer** maxCol to 10

DECLARE integer rowCount to 0

DECLARE integer currentRow to 0

SET tempArray to **new** **double object**[maxRow][maxCol]

DECLARE Scanner variable inputArray1 to **new** Scanner object with parameter file

WHILE (inputArray1 has next)

INCREMENT rowCount

SET inputArray.hasNext()

CLOSE inputArray1

SET data to new double[rowCount][]

DECLARE Scanner variable inputArray2to **new** Scanner object with parameter file

WHILE (inputArray2 has next)

DECLARE String variable a to inputArray2.nextLine

DECLARE String column [] to a.split(" ")

DECLARE data[currentRow] to **new** **double**[column.length]

**FOR** (DECLARE **integer** variable c to 0; REPEAT WHILE c is less than column.length; INCREMENT c)

DECLARE data[currentRow][c] to Double.*parseDouble*(column[c])

INCREMENT currentRow

CLOSE inputArray2

**IF** inputArray2 equals **null**

**RETURN** **null**

**RETURN** data

DECLARE **public** **static** **void** method writeToFile with parameters **double**[][] data and java.io.File outputFile with **throws** java.io.FileNotFoundException

DECLARE PrintWriter outputData to **new** PrintWriter object with parameter (outputFile)

//Sets up a for loop that goes through all the rows

**FOR** (DECLARE **integer variable** r to 0; REPEAT WHILE r is less than data.length; INCREMENT r)

//Sets up a for loop that goes through all the columns inside the rows

FOR (DECLARE **integer variable** c to 0; REPEAT WHILE c is less than data[r].length; INCREMENT c)

IF(data[r][c] is less than 0)

SET data[r][c] to data[r][c] \* -1

PRINT ("( " +data[r][c]+ ") ") to ouputData

PRINT (data[r][c]+ " ") to ouputData

PRINT ("\n") to outputData

CLOSE outputData

DECLARE **public** **static** **double method** getTotal with parameter **double**[][] data

DECLARE **double variable** total to 0

//Sets up a for loop that goes through all the rows

FOR (DECLARE **integer** variable r to 0; REPEAT WHILE r is less than data.length;

INCREMENT r)

//Sets up a for loop that goes through all the columns inside the rows

FOR (DECLARE **integer** variable c to 0; REPEAT WHILE c is less than data[r].length; INCREMENT c)

SET total to add equal to data [r][c]

//Return

**RETURN** total

**DECLARE public** **static** **double method** getAverage with parameter **double**[][] data

**DECLARE double variable** total to 0

DECLARE **double** variable num to 0

DECLARE **double variable** average to 0

//Sets up a for loop that goes through all the rows

FOR (DECLARE **integer variable** r to 0; REPEAT WHILE r is less than data.length; INCREMENT r)

//Sets up a for loop that goes through all the columns inside the rows

FOR (DECLARE **integer variable** c to 0; REPEAT WHILE c is less than data[r].length; INCREMENT c)

SET total to add equal to data [r][c]

INCREMENT num

SET average equal to total divided by num

//Return

**RETURN** average

DECLARE **public** **static** **double** method getRowTotal with parameters **double**[][] data and **integer** row

DECLARE **double** variable totalR to 0

//Sets up a for loop that goes through all the columns inside the rows

FOR (DECLARE **integer variable** c to 0; REPEAT WHILE c is less than data[row].length; INCREMENT c)

SET totalR add equal to data[row][c]

//Return

**RETURN** totalR

DECLARE **public** **static** **double method** getColumnTotal with parameters **double**[][] data and **integer** col

DECLARE **double** variable totalC to 0

//Sets up a for loop that goes through all the rows

FOR (SET **integer variable** r to 0; REPEAT WHILE r is less than data.length; INCREMENT r)

**IF** (data[r].length is greater than or equal to col AND data[r] is not equal to **null**)

CONTINUE

SET totalC add equal to data[r][col]

//Return

**RETURN** totalC

**DECLARE public** **static** **double** method getHighestInRow with parameters **double**[][] data and **integer variable** row

DECLARE **double** variable highestR to 0

//Loop//Sets up a for loop that goes through all the rows

FOR (DECLARE **integer variable** c to 0; REPEAT WHILE c is less than data[row].length; INCREMENT c)

//Sets up a for loop that goes through all the columns inside the rows

**IF** highestR is less than data[row][c]

SET highestR to data[row][c]

**RETURN** highestR

**DECLARE public** **static** **integer method** getHighestInRowIndex with parameters **double**[][] data and **integer variable** row

DECLARE **double variable** highestR to 0

DECLARE **integer variable** hrIndex to 0

//Loop sets up a for loop that goes through all the columms

FOR (DECLARE **integer variable** c to 0; REPEAT WHILE c is less than data[row].length; INCREMENT c)

//Sets up a for loop that goes through all the columns inside the rows

IF highestR is less than data[row][c]

SET highestR to data[row][c]

SET hrIndex to row

RETURN hrIndex

DECLARE **public** **static** **double method** getLowestInRow with parameters **double**[][] data and **integer variable** row

DECLARE **double** variable lowestR to 1000000

//Loop sets up a for loop that goes through all the columns

FOR (DECLARE **integer variable** c to 0; REPEAT WHILE c is less than data[row].length; INCREMENT c)

//Sets up a for loop that goes through all the columns inside the rows

**IF** lowestR is greater than data[row][c]

SET lowestR to data[row][c]

**RETURN** lowestR

DECLARE **public** **static** **integer method** getLowestInRowIndex with parameters **double**[][] data and **integer variable** row

DECLARE **double** variable lowestR to 1000000

DECLARE **integer variable** lrIndex to 0

//Loop sets up a for loop that goes through all the columns

FOR (DECLARE **integer variable** c to 0; REPEAT WHILE c is less than data[row].length; INCREMENT c)

//Sets up a for loop that goes through all the columns inside the rows

**IF** lowestR is greater than data[row][c]

SET lowestR to data[row][c]

SET lrIndex to c

RETURN lrIndex

DECLARE **public** **static** **double method** getHighestInColumn with parameters **double**[][] data and **integer variable** col

DECLARE **double variable** highestC to 0

//Sets up a for loop that goes through all the rows

FOR (DECLARE **integer variable** r to 0; REPEAT WHILE r is less than data.length; INCREMENT r)

**IF** data[r].length is less than or equal to col

CONTINUE

IF highestC is less than data[r][col]

SET highestC to data[r][col]

RETURN highestC

DECLARE **public** **static** **integer method** getHighestInColumnIndex with parameters **double**[][] data and **integer variable** col

DECLARE **double variable** highestC to 0

DECLARE **integer variable** hcIndex to 0

//Sets up a for loop that goes through all the rows

FOR (DECLARE **integer variable** r to 0; REPEAT WHILE r is less than data.length; INCREMENT r)

//Sets up an if statement continues the for loop if the data[r] length is less than or equal to col

**IF** data[r].length is less than or equal to col CONTINUE

IF highestC is less than data[r][col]

SET highestC to data[r][col]

SET hcIndex to col

RETURN hcIndex

DECLARE **public** **static** **double method** getLowestInColumn with parameters **double**[][] data and **integer variable** col

**DECLARE double** variable lowestC to 1000000

//Sets up a for loop that goes through all the rows

FOR (DECLARE **integer variable** r to 0; REPEAT WHILE r is less than data.length; INCREMENT r)

**IF** data[r].length is less than or equal to col CONTINUE

//Sets up a for loop that goes through all the columns inside the rows

IF lowestC is greater than data[r][col]

SET lowestC to data[r][col]

**RETURN** lowestC

DECLARE **public** **static** **integer method** getLowestInColumnIndex with parameters **double**[][] data and **integer variable** col

**DECLARE double** variable lowestC to 1000000

DECLARE i**nteger variable** lcIndex to 0

//Sets up a for loop that goes through all the rows

FOR (DECLARE **integer variable** r to 0; REPEAT WHILE r is less than data.length; INCREMENT r)

**IF** data[r].length is less than or equal to col CONTINUE

//Sets up a for loop that goes through all the columns inside the rows

IF lowestC is greater than data[r][col]

SET lowestC to data[r][col]

SET lcIndex to col

RETURN lcIndex

DECLARE **public** **static** **double method** getHighestInArray with parameter **double**[][] data

DECLARE **double variable** highestA to 0

//Sets up a for loop that goes through all the rows

FOR (DECLARE **integer variable** r to 0; REPEAT WHILE r is less than data.length; INCREMENT r)

//Loop sets up a for loop that goes through all the columms

FOR (DECLARE **integer variable** c to 0; REPEAT WHILE c is less than data[r].length; INCREMENT c)

IF highestA is less than data[r][c]

SET highestA to data [r][c]

//Return

**RETURN** highestA

DECLARE **public** **static** **double method** getLowestInArray with parameter **double**[][] data

DECLARE **double** variable lowestA to 1000000

//Sets up a for loop that goes through all the rows

FOR (DECLARE **integer variable** r to 0; REPEAT WHILE r is less than data.length; INCREMENT r)

//Loop sets up a for loop that goes through all the columms

FOR (DECLARE **integer variable** c to 0; REPEAT WHILE c is less than data[row].length; INCREMENT c)

IF lowestA is greater than data[r][c]

SET lowestA to data [r][c]

//Return

**RETURN** lowestA

**HolidayBonus.java:**

**DECLARE public constructor** HolidayBonus()

DECLARE **public** **static** **double**[] method calculateHolidayBonus with parameters **double**[][] data, **double** variable high, **double variable** low, and **double variable** other

**DECLARE double**[] variable result to **null**

//Sets up a for loop that goes through all the rows

FOR (DECLARE **integer variable** r to 0; REPEAT WHILE r is less than data.length; INCREMENT r)

//Loop sets up a for loop that goes through all the columms

FOR (DECLARE **integer variable** c to 0; REPEAT WHILE c is less than data[row].length; INCREMENT c)

**DECLARE lowestI as a CALL to** TwoDimRaggedArrayUtility.*getLowestInColumnIndex*(data, c)

//If lowestI is not null

IF array[lowestI] does not equal **null**

//If lowest index is less than or equal to 0, set data element to 0

**IF** data[r][c] is less than or equal to 0

SET array[lowestI][c] to 0

**else** //If lowest index is greater than 0, set data element to low

SET array[lowestI][c] to low

//Calls getHighestInColumnIndex

**DECLARE highestI as a CALL to** T woDimRaggedArrayUtility.*getHighestInColumnIndex*(data, c)

**IF** array[highestI] does not equal **null**

//If lowest index is less than or equal to 0, set array element to 0

IF data[r][c] less than or equal to 0

SET array[highestI][c] to 0

**else** //Else if lowest index is greater than 0, set array element to low

SET array[highestI][c] to high

//If array is not null and

IF array[r] does not equal **null**

//If row does not equal highestI and does not equal lowestI, set array element to other

IF r does not equal highestI AND r does not equal lowestI

SET array[r][c] to other;

//add high to result

**IF** highestI equals r

SET result[r] to add equal high

CONTINUE

//add low to result

IF lowestI equals r

SET result[r] to add equal low

CONTINUE

//add other to result

IF r does not equal highestI AND r does not equal lowestI

SET result[r] add equal to other

CONTINUE

**RETURN** result

DECLARE **public** **static** **double method** calculateTotalHolidayBonus with parameters **double**[][] data, **double** variable high, **double variable** low, and **double variable** other

DECLARE **double variable** totalHB to 0

CALL HolidayBonus.*calculateHolidayBonus* with parameters data, high, low, and other and set it as bonus[]

FOR (DECLARE **integer variable** i to 0; REPEAT WHILE i is less than bonus.length; INCREMENT i)

SET totalHB to add equal to bonus[i]

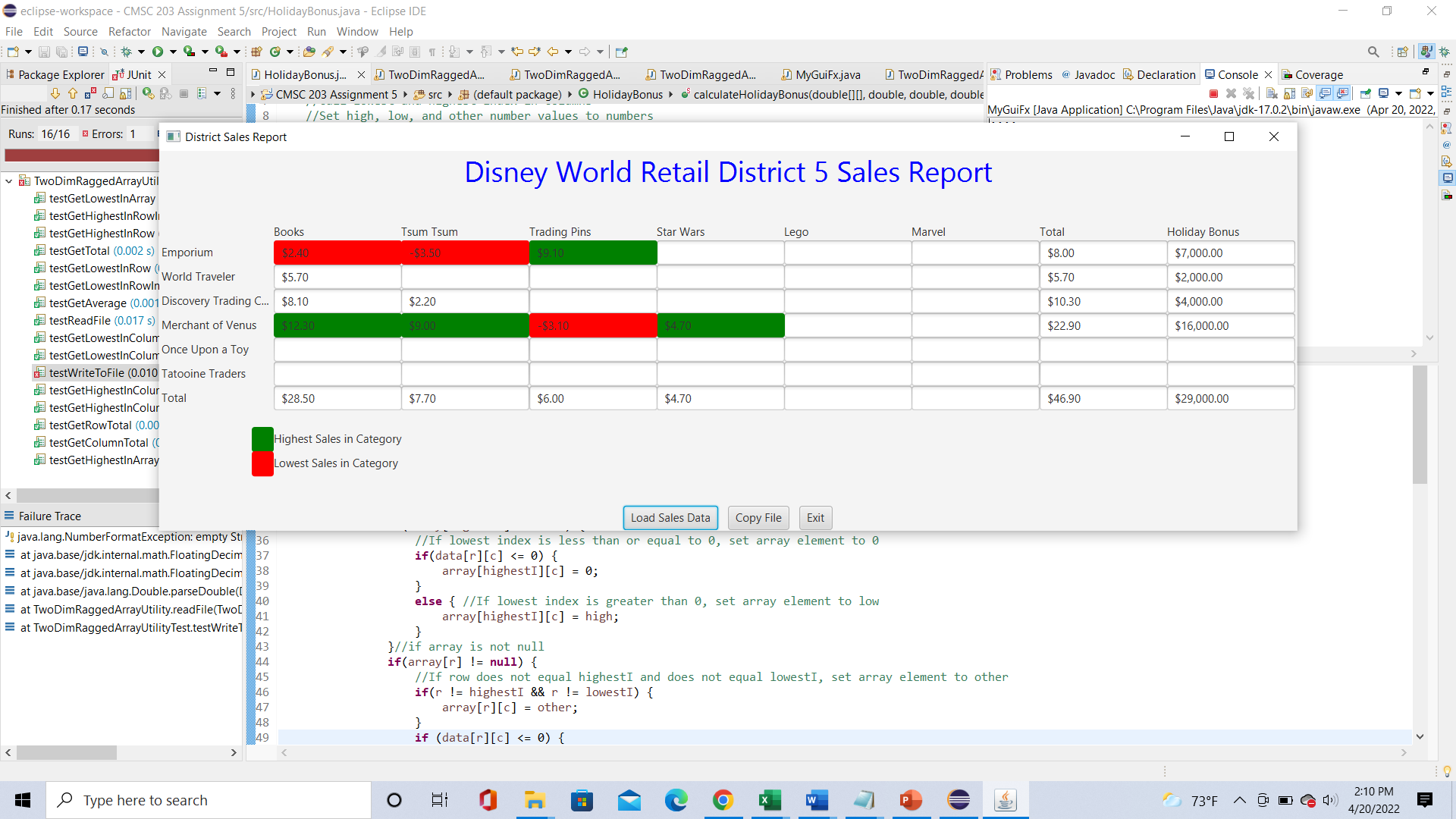
RETURN totalHB

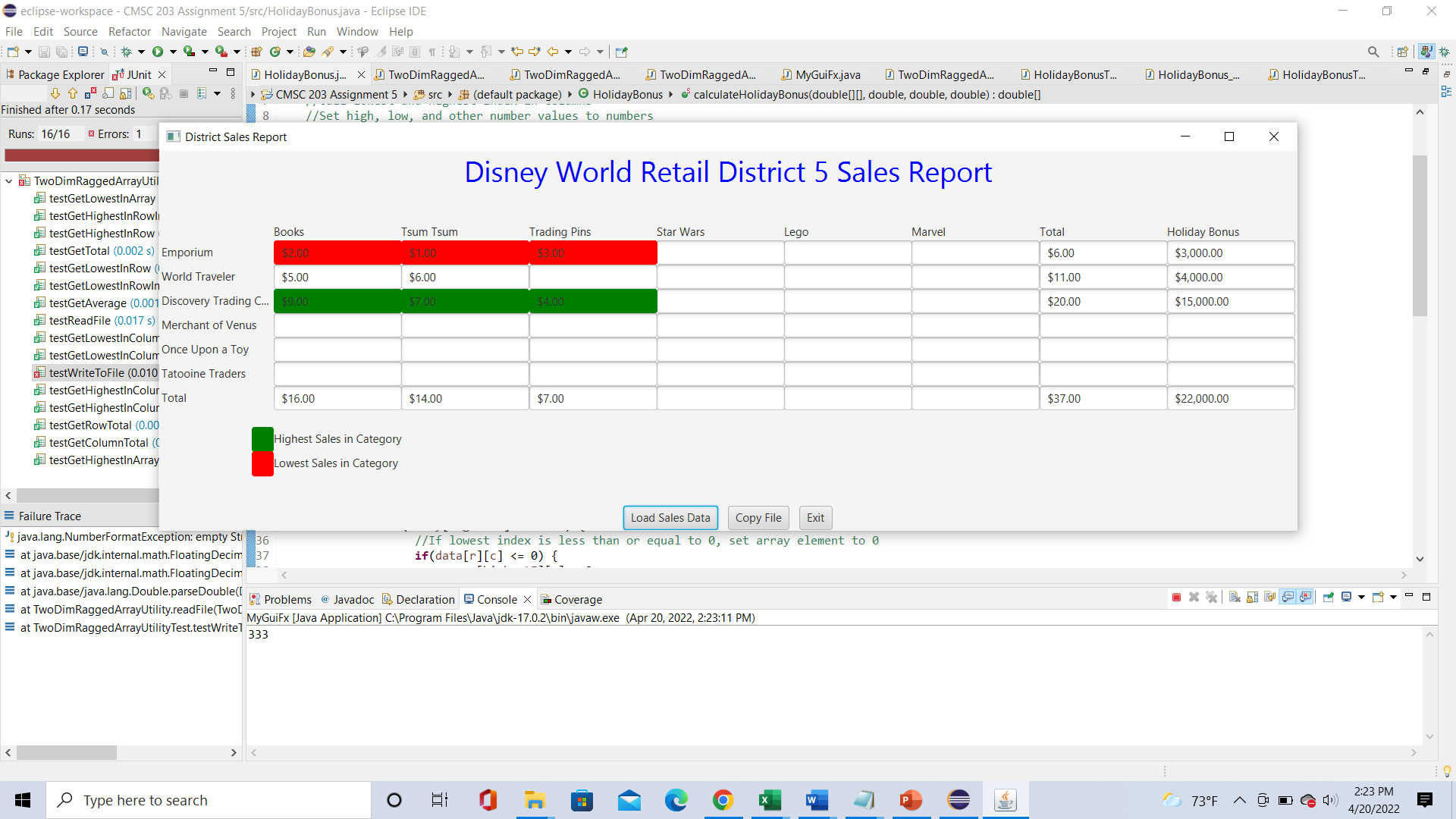
**Part2: Comprehensive Test Plan**

A good test plan should be comprehensive. This means you should have a few test cases that test when the input is in and out of range, division by 0, incorrect Data type, etc. (Provide valid and invalid input)

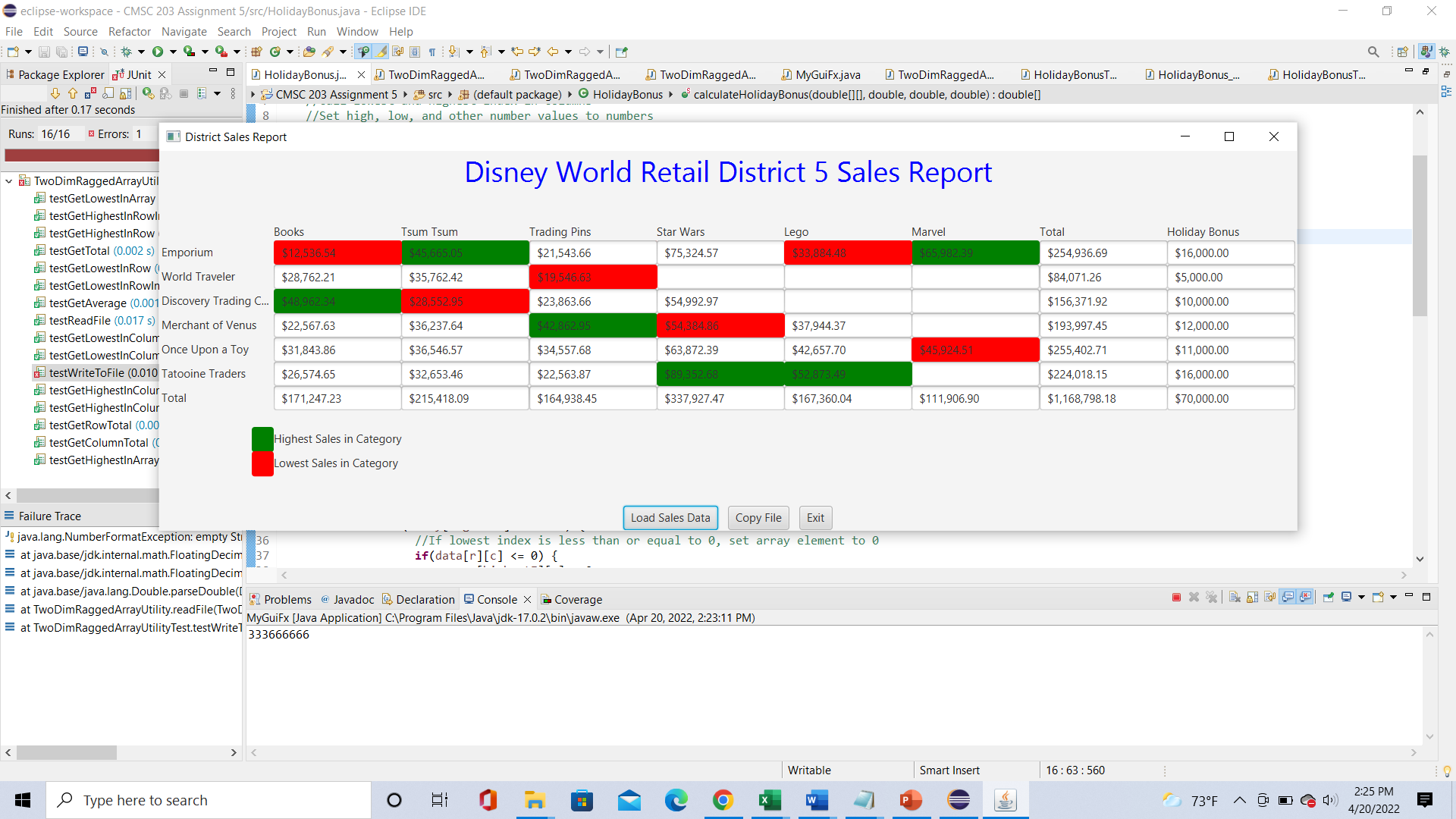
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cases | Input | Expected Output | Actual Output | Did Test Pass? |
| Case 1 | 2 1 3  5 6  9 7 4 | 2 1 3 6 3000  5 6 11 4000  9 7 4 20 15000  16 14 7 37 22000 | 2 1 3 6 3000  5 6 11 4000  9 7 4 20 15000  16 14 7 37 22000 | Yes |
| Case 2 | 2.4 -3.5 9.1  5.7  8.1 2.2  12.3 9.0 -3.1 4.7 | 2.4 -3.5 9.1 8 7,000  5.7 5.7 2,000  8.1 2.2 10.3 4,000  12.3 9.0 -3.1 4.7 22.9 16,000  28.5 7.7 6.0 4.7 46.9 29,000 | 2.4 -3.5 9.1 8 7,000  5.7 5.7 2,000  8.1 2.2 10.3 4,000  12.3 9.0 -3.1 4.7 22.9 16,000  28.5 7.7 6.0 4.7 46.9 29,000 | Yes |

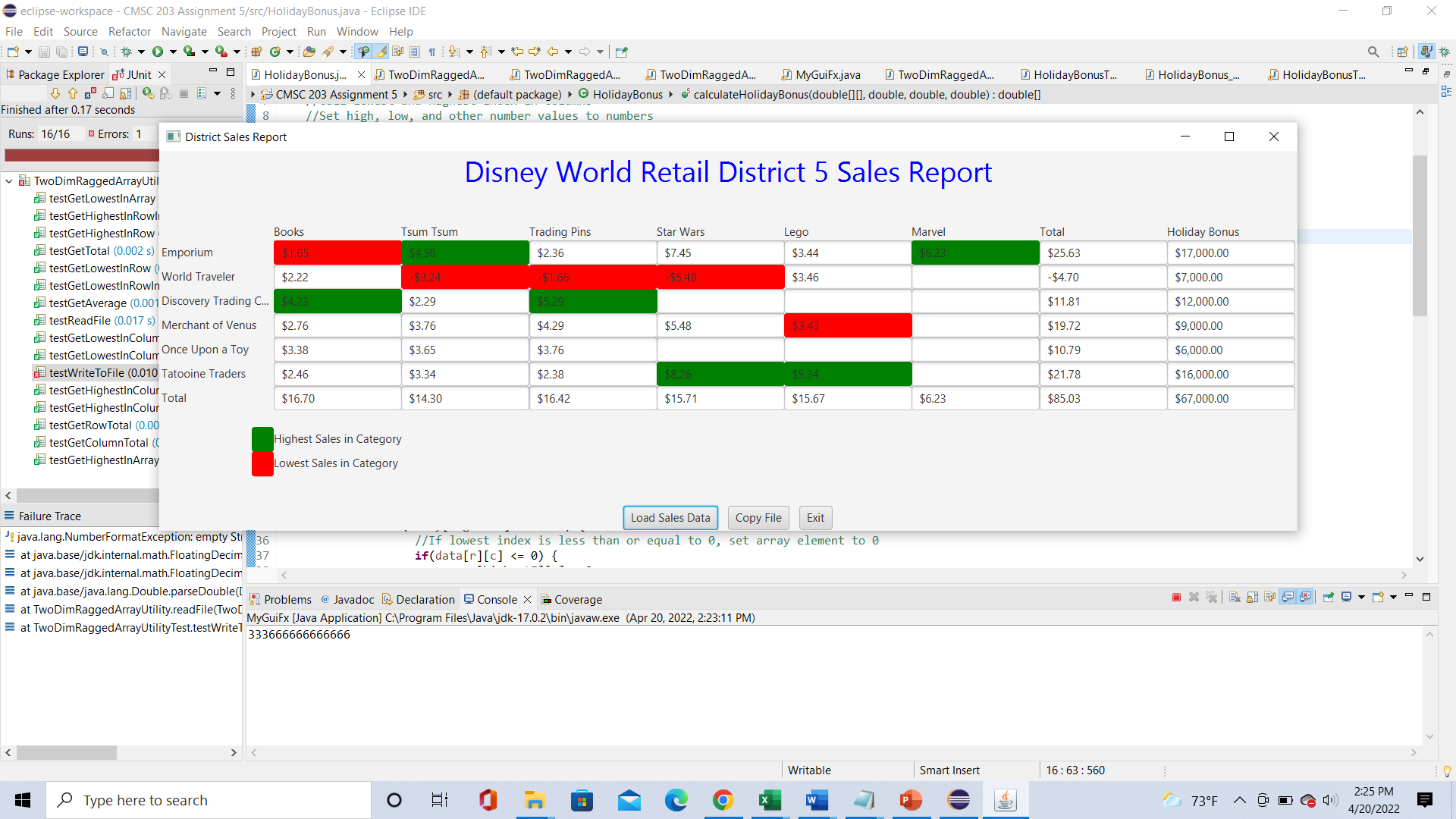
**Screenshots related to the Test Plan:**

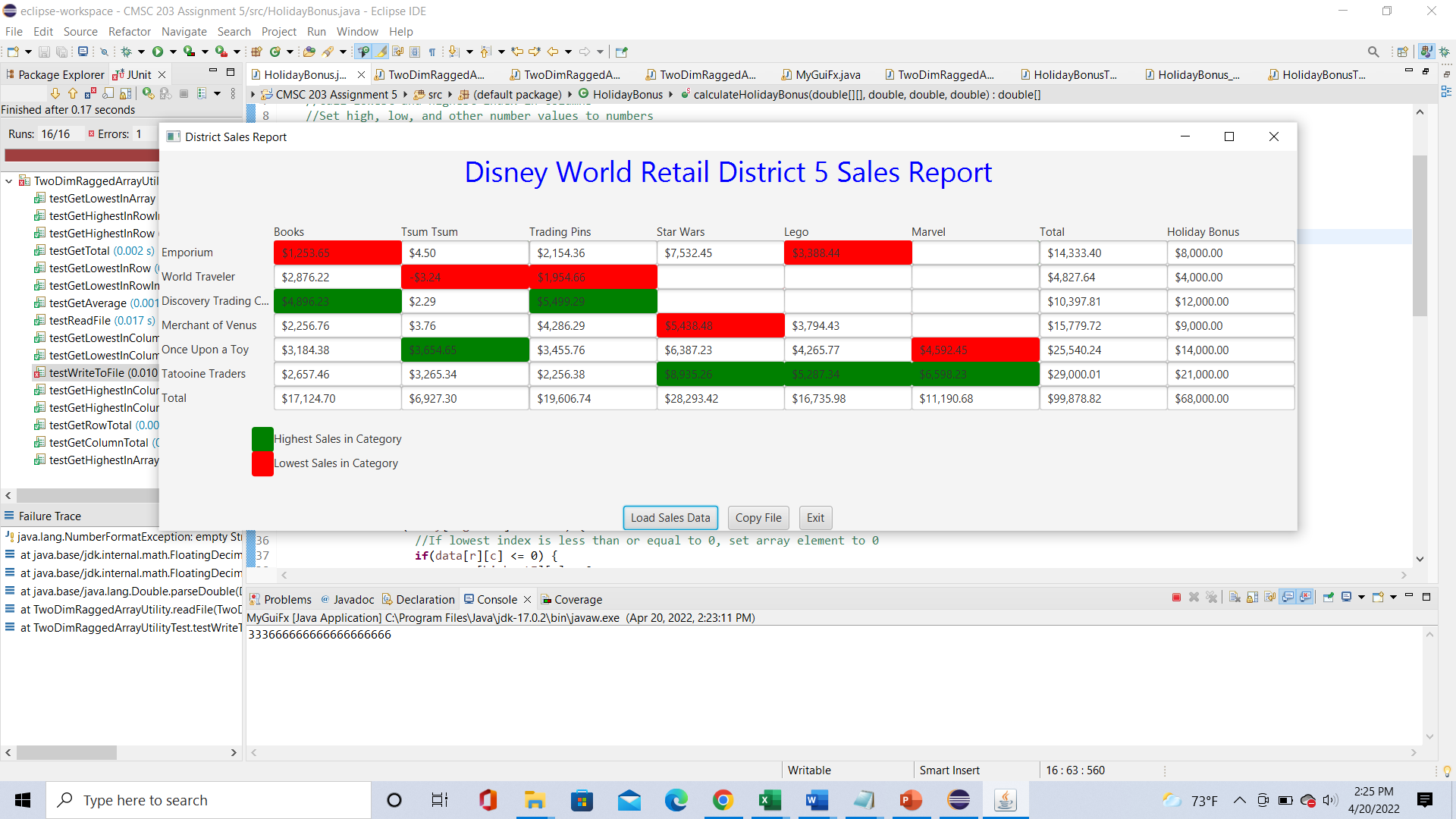
****

****

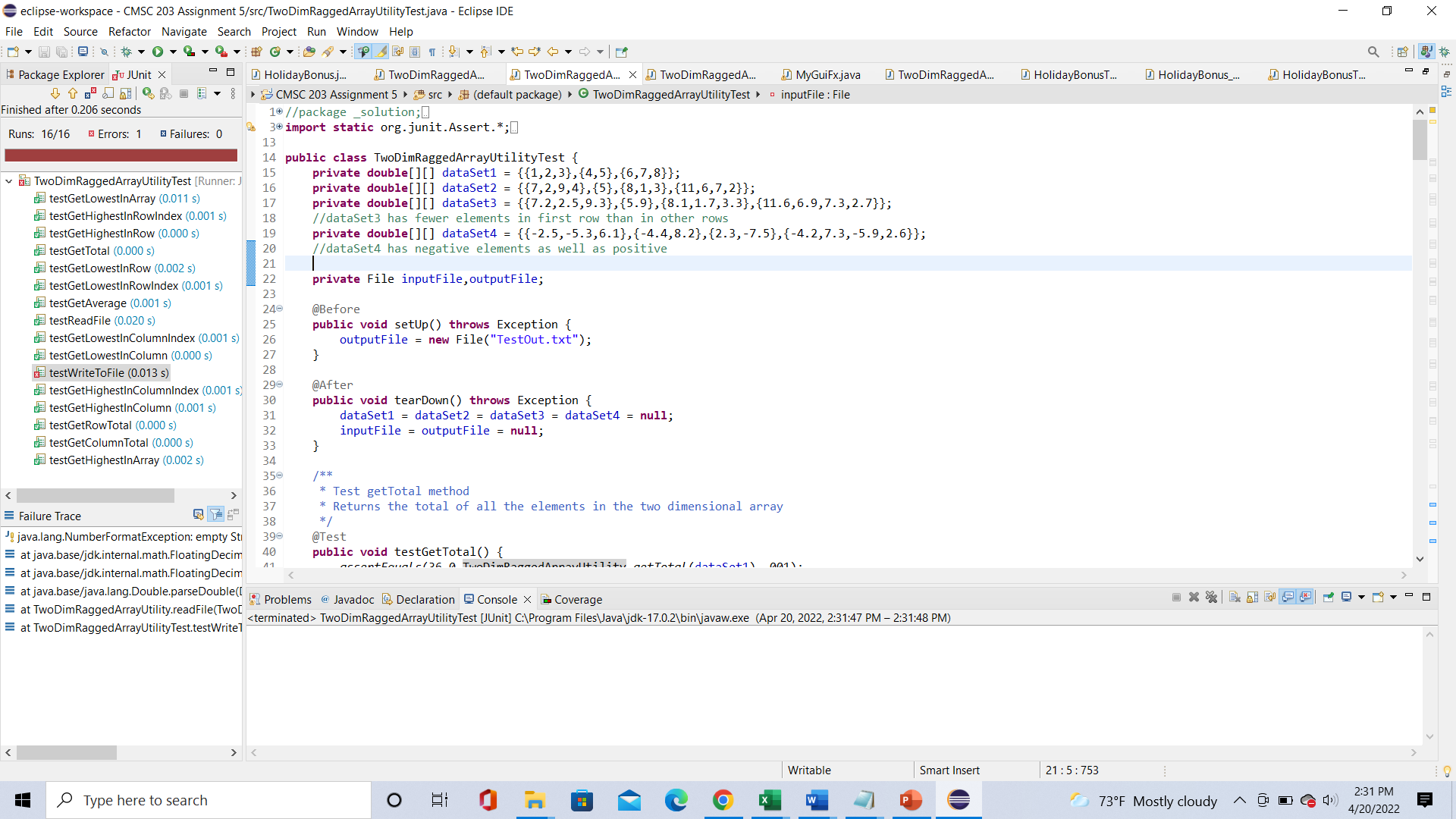
**Screenshots of GUI properties:**

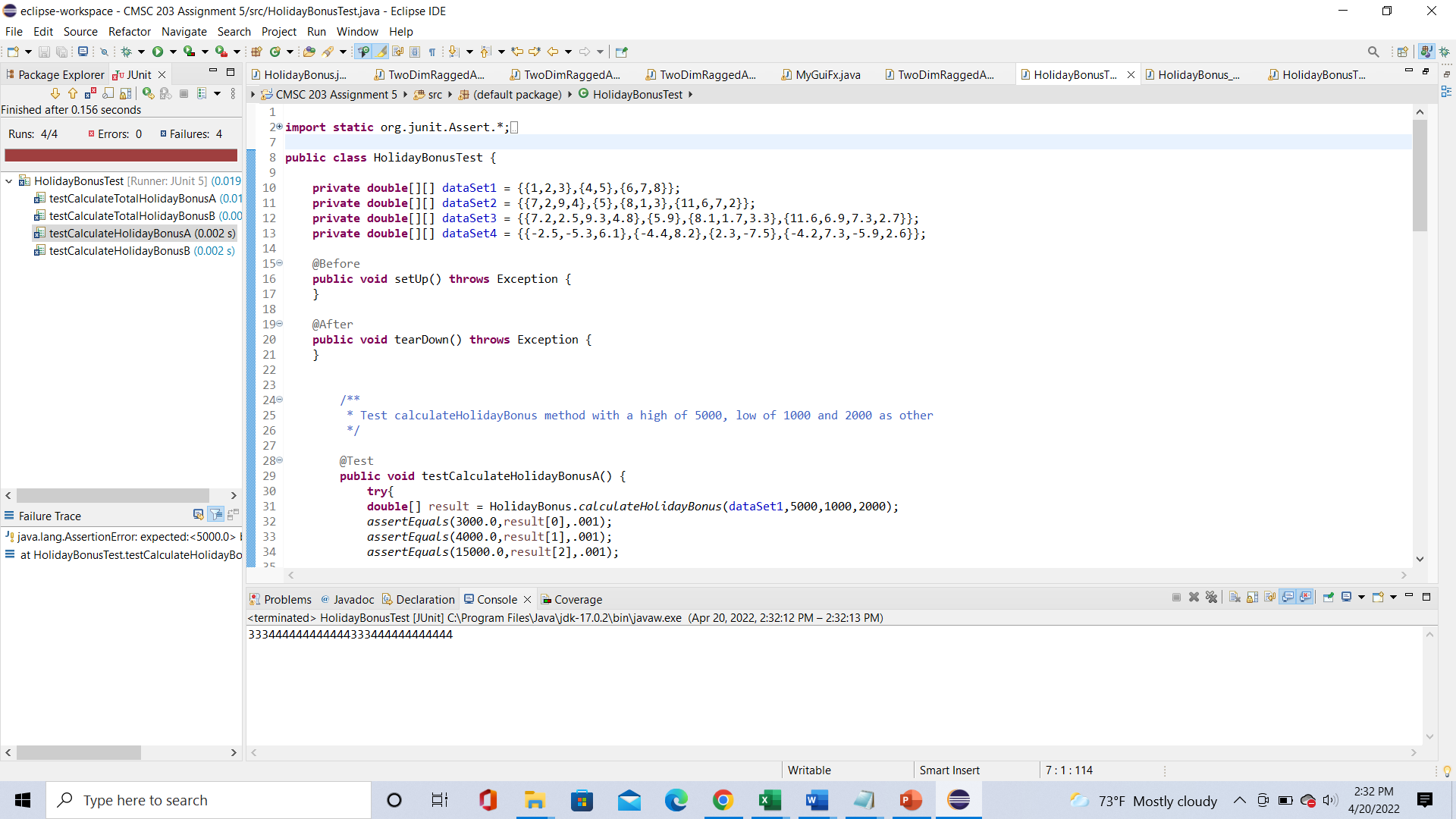
****

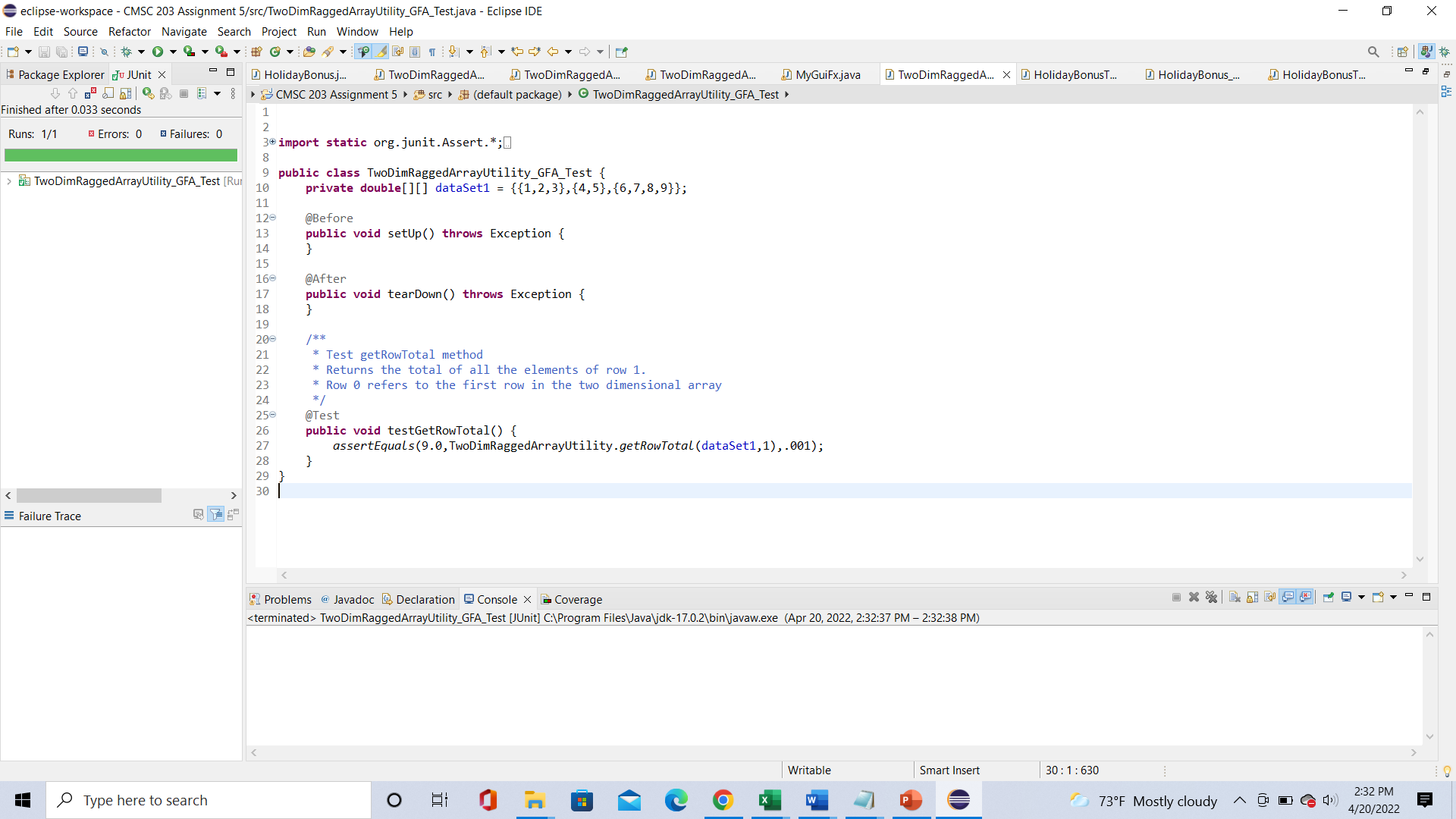
****

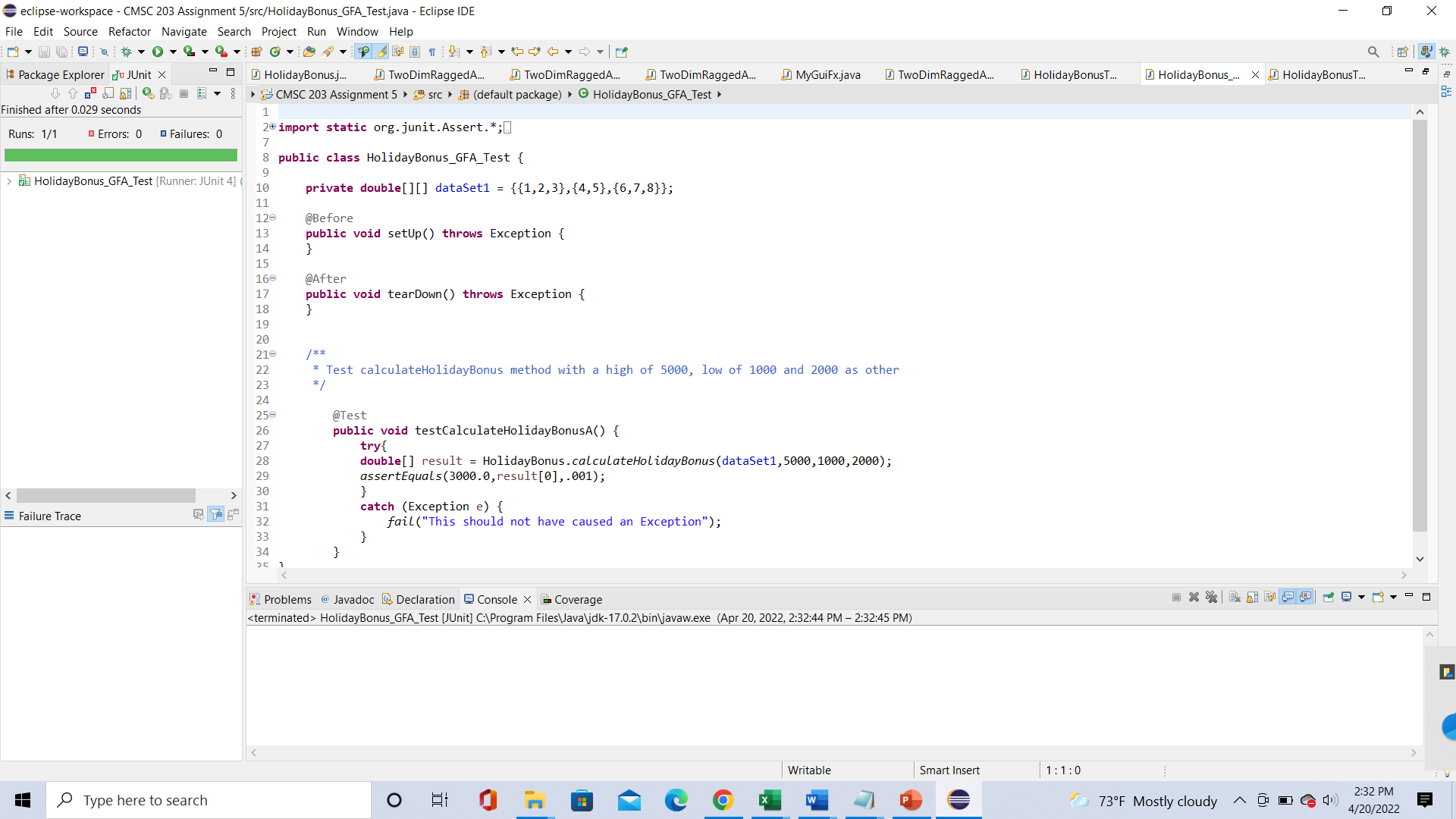
****

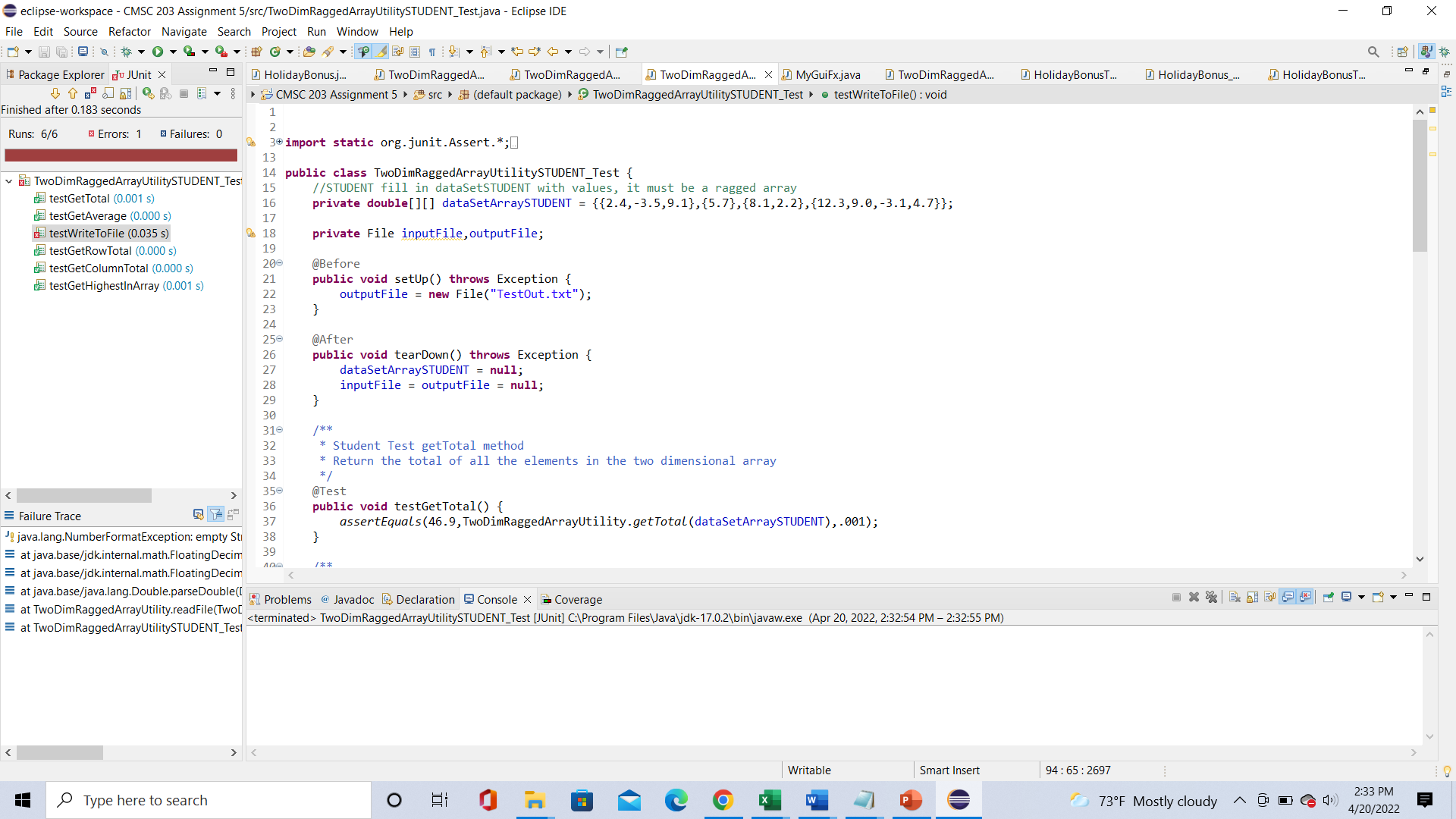
**J\_Unit Tests:**

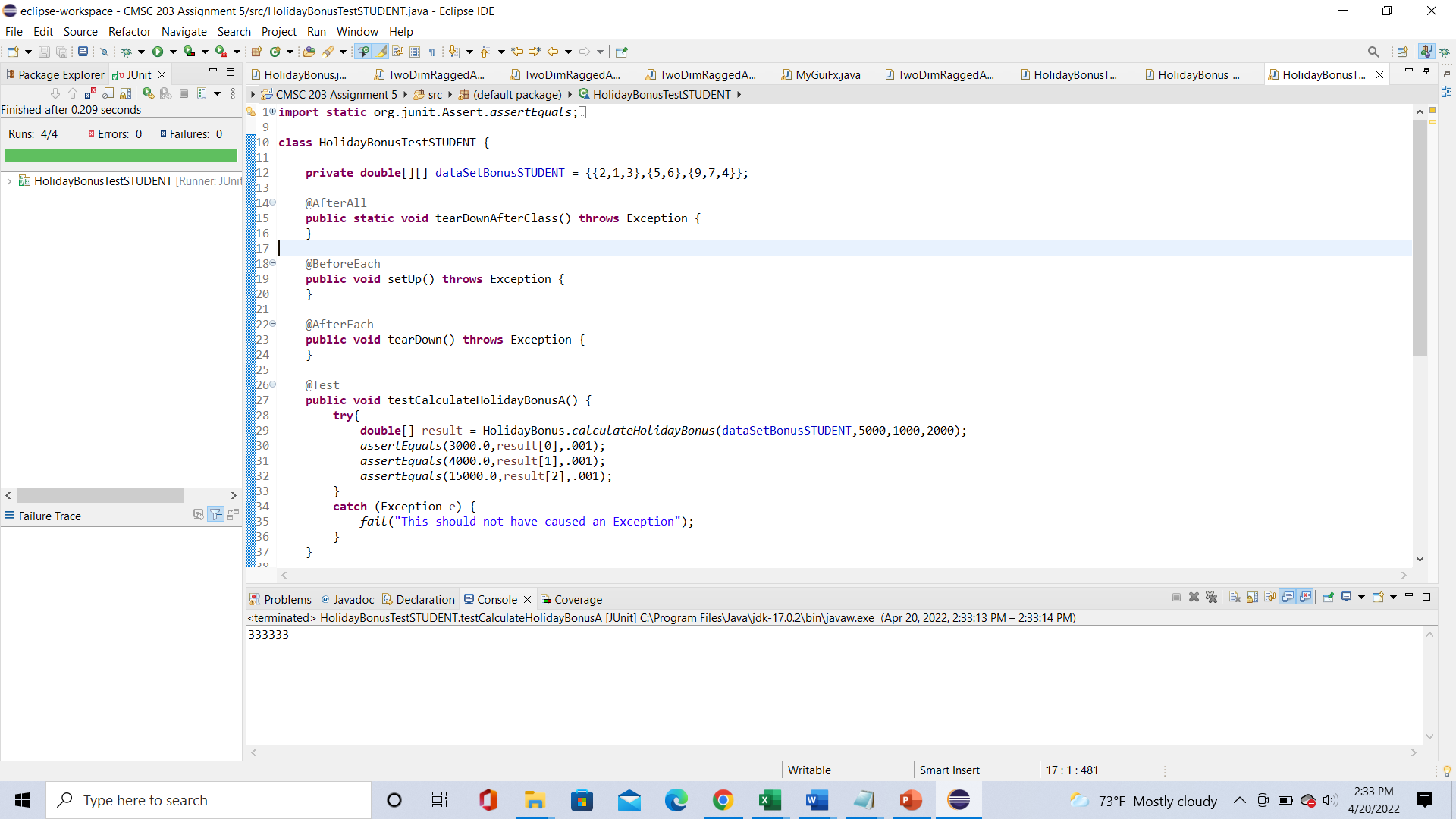
****

****

****

****

****

****

**Git Hub Submission:**

**Lessons Learned** <Provide answers to the questions listed above>**:**

Write about your Learning Experience, highlighting your lessons learned and learning experience from working on this project.

This program was a little but less complicated than the previous one. I started with setting up the methods. The assignment didn’t have any variables. The get methods were quite simple to set up, especially the getHighest methods. I realized that the getLowest methods were not working well. To fix this I changed the variables that are first initialized from 0 to 1000000, and this worked. I also did the STUDENT Tests based on the provided Tests.

The read and write file was bit more tricky. I got a little bit of help on the readFile method. I created a new scanner variable to scan the file and created a while loop that would take count of how many rows there are while there is still a next line. Then I closed the object, made a new array and opened a new scanner object. Then, I did another similar while loop, but the text line in a string variable and separated the values by using the split method. Then, after creating a new array object with the column length, I set up a for loop that went through the columns, set up double.parsedouble to convert all the column values from string to double, and incremented the current row value. Then I closed the second scanner array and returned the data array. The writeFile method was a bit easier since all I had to do was call the outputFile several times.

However, the toughest method was the calculateHolidayBonus method. At first, I tried to make two for loops to go through the rows and columns and print each element. This didn’t work and after a close look at how the method was used in the GUI and J\_Unit tests, I realized that the method should assign values to the array elements. I tried to add the values to the result[row] array if a condition was met. This resulted in the elements showing up in the graph, but they were all wrong. I kept these if statements in the class but had to find other code to add to the class. I tried a few ways to assign the high, low, and other values to elements, but they didn’t work. Eventually, after a bit of assistance, I called the getHighestInColumnIndex and getLowestInColumnIndex methods and set them as highestI and lowestI. I set up if else statements that would make each array value 0 if the data array element was less than or equal to 0 and if not, then it would assign the array element to low if it was the lowest element in the column, to high if it was the highest element in the column, and other if it was neither the highest or lowest element. I also put these if else statements in another if statement that would make sure the array element was not null.

There were a few things with the J\_Unit tests that weren’t perfect, but I didn’t know how to fix them.

What have you learned?

I learned how important the read file method is and how important it is to assign certain values to variables.

What did you struggle with?

I struggled with the read and write files methods and the holiday bonus method.

What would you do differently on your next project?

I would start my next project sooner and get more help when I could. I think I should also update the java files more often instead of updating them on the same day the project is due.

What parts of this assignment were you successful with, and what parts (if any) were you not successful with?

I was mostly successful with the getHighest and getTotal methods. I mainly didn’t succeed with converting the negative numbers and a few things with the J\_Unit tests.

Provide any additional resources/links/videos you used to while working on this assignment/project.

Assignment 5 Check List (include Yes/No or N/A for each item)

|  |  |  |  |
| --- | --- | --- | --- |
| **#** |  | **Y/N or N/A** | **Comments** |
|  | **Assignment files:** |  |  |
|  | * FirstInitialLastName\_ Assignment5\_Moss.zip | **Y** | **The original java files are in the Moss folder** |
|  | * FirstInitialLastName\_Assignment5\_Complete.zip | **Y** | **Contains, this document, a UML diagram, the two arrays from the STUDENT test, 4 java files, and 4 html files** |
|  | **Program compiles** | **Y** | **The GUI works. I took separate screen shots similar to the ones from the assignment description.** |
|  | **Program runs with desired outputs related to a Test Plan** | **Y** | **As the document desired, I took the arrays from the student tests and made them their own text files.** |
|  | **Documentation file:** |  |  |
|  | * Comprehensive Test Plan | **Y** | **Two tests of the arrays from the STUDENT tests** |
|  | * Screenshots for each Junit Test | **Y** | **All 6 J\_Unit test have a screen shot** |
|  | * Screenshots for each Test case listed in the Test Plan | **Y** | **Two screen shots** |
|  | * Screenshots of your GitHub account with submitted Assignment# (if required) | **Y** | **Required** |
|  | * UML Diagram | **Y** | **Done** |
|  | * Algorithms/Pseudocode | **Y** | **Not as many comments in this project** |
|  | * Flowchart (if required) | **N/A** | **Not required** |
|  | * Lessons Learned | **Y** | **Done** |
|  | * Checklist is completed and included in the Documentation | Y | Complete |

Additional resources:

# Pearson Revel online platform/textbook: Revel: Java Control Structures through Objects plus (1st Edition)