**Montgomery College**

**CMSC 203**

**Assignment 5 Design**

1. Turn in a UML diagram for TwoDimRaggedArrayUtility and HolidayBonus classes.
2. Write the pseudo code for the methods of *TwoDimRaggedArrayUtility* and *HolidayBonus* class based on the Assignment 5 Description given to you.

Your pseudo-code should be part-way between English and java. There is no need to spell out all the details of variable declaration, etc., but by the same token, the pseudo-code needs to have enough detail that a competent Java programmer could implement it.

Refer to the [**Pseudocode Guideline**](#bookmark=id.gjdgxs)on how to write Pseudocode.

**Pseudocode Guideline**

Pseudocode is code written for human understanding not a compiler. You can think of pseudocode as “English code,” code that can be understood by anyone (not just a computer scientist). Pseudocode is not language specific, which means that given a block of pseudocode, you could convert it to Java, Python, C++, or whatever language you so desire.

Pseudocode will be important to your future in Computer Science. Typically pseudocode is used to write a high-level outline of an algorithm.

As you may already know, an algorithm is a series of steps that a program takes to complete a specific task. The algorithms can get very complicated without a detailed plan, so writing pseudocode before actually coding will be very beneficial.

**How to Write Pseudocode**

There are no concrete rules that dictate how to write pseudocode, however, there are commonly accepted standards. A reader should be able to follow the pseudocode and hand-simulate (run through the code using paper and pencil) what is going to happen at each step. After writing pseudocode, you should be able to easily convert your pseudocode into any programming language you like.

We use indentation to delineate blocks of code, so it is clear which lines are inside of which method (function), loop, etc. Indentation is crucial to writing pseudocode. Java may not care if you don't indent inside your **if** statements, but a human reader would be completely lost without indentation cues.

**Remember:** Human comprehension is the whole point of pseudocode. So, what does pseudocode look like?

| **Pseudocode** | **Real Code in Java** |
| --- | --- |
| Declare an integer variable called n  Declare an integer variable sum.  Declare an integer variable f1  Declare an integer variable f2  If n is less than 2  sum =n  else  set sum to 0  set f1 and f2 to 1  repeat n times  sum = f1 + f2  f2 = f1  f1 = sum  end loop  print sum | **int** n,k, f1, f2, sum;  **if** ( n < 2 )  sum =n;  **else**  {  sum=0;  f1 = f2 = 1;    **for**(k=2; k<n; k++)  {  sum = f1 + f2;  f2 = f1;  f1 = sum;  }  }  System.***out***.println("Fibonacci of number " + n + " is "+ sum); |

**Finding the Fibonacci numbers till n:**

**Remember that pseudocode is not language specific so we are not looking for “almost Java” code, but instead, we are looking for a strong understanding of the algorithm at hand.**

getAverage:

Declare double variable total

Declare int variable count

For each value in the data array:

Add the value to total

Increment count

Return total divided by count

getColumnTotal:

Declare double variable total

For each value in data[][col]:

Add value to total

Return total

getHighestInArray:

Declare double variable max = data[0][0]

For each value in data:

if value is greater than max, max = value

Return max

getHighestInColumn:

Declare double variable max

For each value in data[][col]:

if value is greater than max, max = value

Return max

getHighestInColumnIndex:

Declare double variable max

Declare int variable index

For each value in data[][col]:

if value is greater than max, max = value

index = index of value

Return index

getHighestInRow:

Declare double variable max

For each value in data[row]:

if value is greater than max, max = value

Return max

getHighestInRowIndex:

Declare double variable max

Declare int variable index

For each value in data[row]:

if value is greater than max, max = value

index = index of value

Return index

getLowestInArray:

Declare double variable min = data[0][0]

For each value in data:

if value is less than min, min = value

Return min

getLowestInColumn:

Declare double variable min

For each value in data[][col]:

if value is less than min, min = value

Return min

getLowestInColumnIndex:

Declare int variable index

Declare double variable min

For each value in data[][col]:

if value is less than min, min = value  
 index = index of value

Return index

getLowestInRow:

Declare double variable min

For each value in data[row]:

if value is less than min, min = value

Return min

getLowestInRowIndex:

Declare double variable min

Declare int variable index

For each value in data[row]:

if value is less than min, min = value

index = index of value

Return index

getRowTotal:

Declare double variable total

for each value in data[row]:

total += value

Return total

getTotal:

Declare double variable total

for each value in data[][]:

total += value

Return total

readFile:

Open file

Create jagged array double[10][]

while file has next line:

Declare String variable line = next line

Split string variable with “ “

set value in array to Double.valueOf(value)

close file input

return jagged array

writeToFile:

Declare printWriter variable FileOutput

for each element in data array:

print value + “ “ to file

close file output

calculateHolidayBonus:

create double array with size data.length

for each value in data:

if value = getHighestInColumn

set value in doubleArray to high

if value = getLowestInColumn

set value in doubleArray to low

if value > 0

set value in doubleArray to other

else

set value in doubleArray to 0

return doubleArray

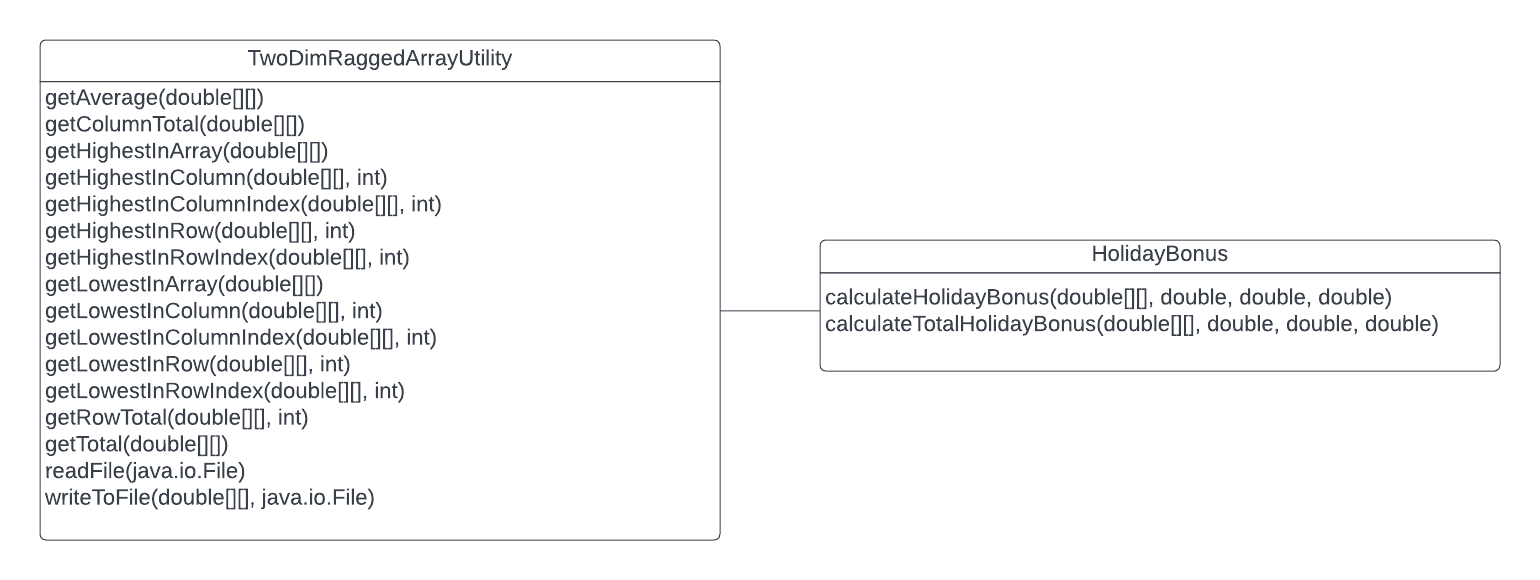
calculateTotalHolidayBonus:

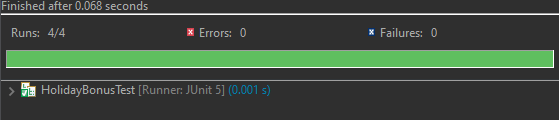
Declare double variable total

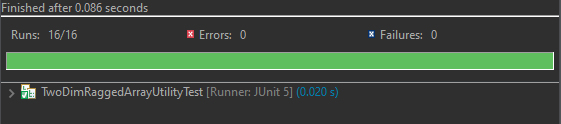
for each value in data[][]:

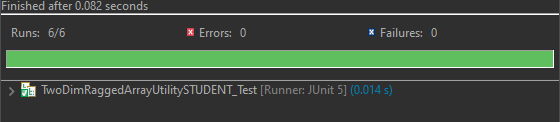
total += value

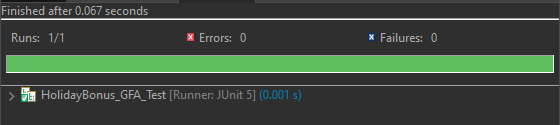
return total

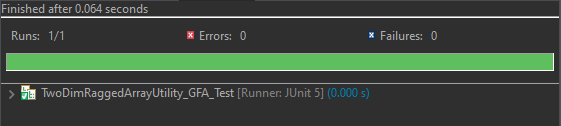


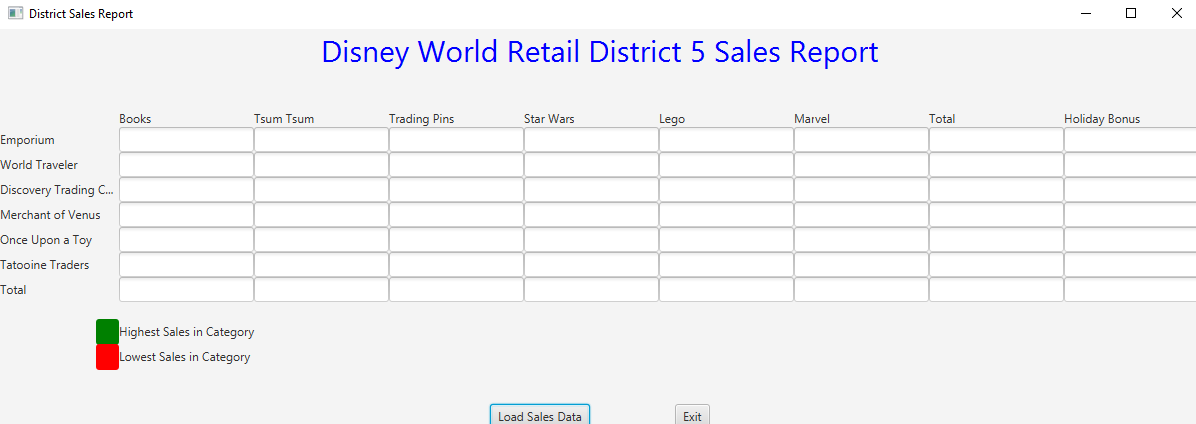


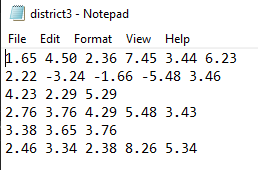




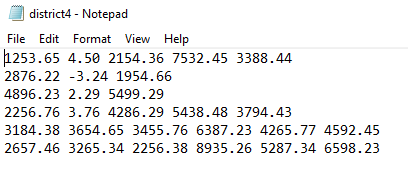


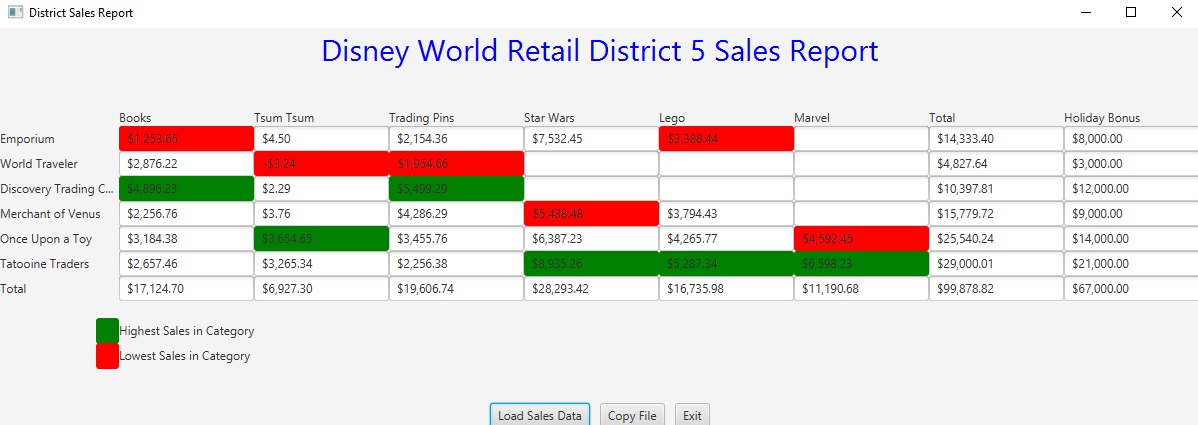


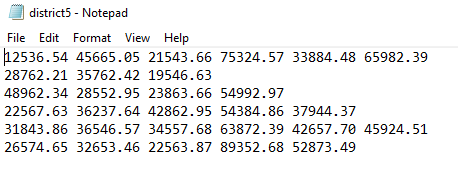


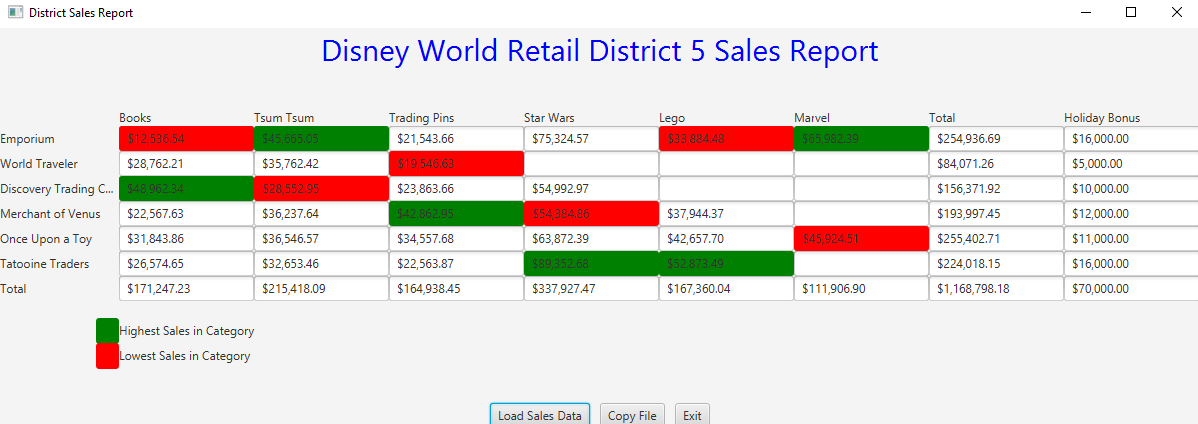












**Lessons Learned:**

**By working on this project, I have learned how to create multiple different classes with various methods that do different functions, and implement the methods in other classes for use in applications. Writing every method and making sure they all work properly was challenging, but it was a good experience to develop and expand my programming knowledge. I struggled with making sure all of my methods passed the JUnit tests, but eventually I figured out all my issues and created a successful program. I succeeded with putting all of the methods together successfully and using them all for their intended purpose within the program, to make the gui work properly.**

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** |  |
|  | * FirstInitialLastName\_Assignment5\_Complete.zip | **Y** |  |
|  | **Program compiles** | **Y** |  |
|  | **Program runs with desired outputs related to a Test Plan** | **Y** |  |
|  | **Documentation file:** |  |  |
|  | * Comprehensive Test Plan | **Y** |  |
|  | * Screenshots for each Junit Test | **Y** |  |
|  | * Screenshots for each Test case listed in the Test Plan | **Y** |  |
|  | * Screenshots of your GitHub account with submitted Assignment# (if required) | **Y** |  |
|  | * UML Diagram | **Y** |  |
|  | * Algorithms/Pseudocode | **Y** |  |
|  | * Flowchart (if required) | **N/A** |  |
|  | * Lessons Learned | **Y** |  |
|  | * Checklist is completed and included in the Documentation | Y |  |