Project 10: Expense Tracker

Due on Monday, November 8 by 11:59 p.m.

Description: Keeping track of your expenses is very useful for financial management. For this project we will write a simple program that lets you store, retrieve, and analyze your finances over a period of months.

The data will initially be in four categories: Utilities, Entertainment, Rent, and Food (you can add other categories later). This data will be stored in a file, that can be updated. A sample file is shown below.

```
4
Utilities 57.0 54.23 22.49 35.4
Entertainment 15.4 35.22 98.2 52.21
Rent 800.0 800.0 800.0 800.0
Food 280.87 301.2 295.14 329.18
```

The 4 at the start shows that there are four categories.
Each category is given on a single line, with the category name first, then four months of data, ending in a newline character.

Your program should allow the user to find out how much was spent in any given month, in any given category, the total amount spent in the four months, the total of any transactions in a given range of values (e.g. $50 to $100), and be able to add items to new or existing categories.

This data cannot be stored in a single array—instead three arrays are needed because some of the data is a String and other is double, and arrays can't do that. One array stores the names of the rows. Another array stores the names of the columns (months). A third array stores the amount you spent in that month in each category. The data above is shown in these arrays below. The cols array is perfect size and contains the names of the four months. The rows array is oversize and contains the names of the categories of expenses. By making this array oversize we can add in more categories. The data array is something new to us—it is a two-dimensional array that is used to store the amount spent each month. So $800.00 was the amount spent in rent each month. While $35.22 is the amount spent on Entertainment in February.

Cols:

Jan	Feb	Mar	Apr

Rows:

Utilities
Entertainment
Rent
Food

Data:

57.0	54.23	22.49	35.4
15.40	35.22	98.2	52.21
800.0	800.0	800.0	800.0
280.87	301.20	295.14	329.18

RowSize: 4

Two dimensional arrays are used to store data this is accessed by two indices, instead of one. The first index shows the number of rows (rows are horizontal) and the second shows the number of columns (columns are vertical—just like they are on the front of a building). The indices for a small two-dimensional array are shown below.

[0,0]	[0,1]	[0,2]	[0,3]
[1,0]	[1,1]	[1,2]	[1,3]
[2,0]	[2,1]	[2,2]	[2,3]

Constructing a two-dimensional array is very much like constructing a one-dimensional array—you just tell Java how many rows and columns you need. The row index is on the left and the column index is on the right. If you reverse the order, you will have trouble.

```
double[] data = new double[NUM_ROWS][NUM_COLS];
```

Printing out the contents of a two-dimensional array is done with the nested loop below. Notice that the row index is in the outer loop and the column index is in the inner loop. This is a convention. This means that it could work the other way too, but we all agree to do it the same way so we don't drive each other nuts.

```
for (int row = 0; row < NUM_ROWS; ++row)
        for (int col = 0; col < NUM_COLS; ++col)
                System.out.println(data[row][col]);
```

Parameter passing for two-dimensional array is done using pass by sharing—so two-dimensional arrays work exactly the same way that one-dimensional arrays do. This means that when you change the contents of a two-dimensional array in a method, the contents are changed in the calling method. If you reallocate a two-dimensional array in a method (which you should not do in this project), the array reference must be returned and assigned.

The two-dimensional array that we will use in this project has four columns (one each for four months) and is oversized in the number of rows. Construct this array to have 100 rows and keep track of the number of rows that are stored.

Objectives: Your program will be graded according to the rubric below.

1. (15 points) Declare and construct the rows, cols and data arrays in the main program.
2. (10 points) Call the readFile, writeFile and performOperations methods.
3. (10 points) Write and call the findGrandTotal method.
4. (15 points) Write and call the findRangeTotal method.
5. (10 points) Write the findLabelIndex method.
6. (10 points) Write and call the findMonthlyTotal method, calling the findLabelIndex method inside this method.
7. (10 points) Write and call the findItemTotal method, calling the findLabelIndex method inside this method.
8. (10 points) Write and call the addToData method.

9. (10 points) Use meaningful variable names, consistent indentation, and whitespace (blank lines and spaces) to make your code readable. Add comments where appropriate to explain the overall steps of your program.

Sample Output: Below is an example run of the program. The user input is in bold. Format your print statements so your output matches it given the same input.

```
Enter the file name:
Data.txt
Choose one of the following options:
1. Find monthly total
2. Find item total
3. Find grand total
4. Find range total
5. Add to table
6. Exit program
1
Enter the month
Jan
The total is 1153.27
Choose one of the following options:
1. Find monthly total
2. Find item total
3. Find grand total
4. Find range total
5. Add to table
6. Exit program
2
Enter the item
Utilities
The total is 169.12
Choose one of the following options:
1. Find monthly total
2. Find item total
3. Find grand total
4. Find range total
5. Add to table
6. Exit program
3
The total is 4776.540000000001
Choose one of the following options:
1. Find monthly total
2. Find item total
3. Find grand total
4. Find range total
5. Add to table
6. Exit program
4
Enter the lower amount
50
Enter the upper amount
100
The total is 261.64
Choose one of the following options:
1. Find monthly total
```

```
2. Find item total
3. Find grand total
4. Find range total
5. Add to table
6. Exit program
5
What is the item?
Utilities
Which month?
Feb
What is the amount
100.00
Choose one of the following options:
1. Find monthly total
2. Find item total
3. Find grand total
4. Find range total
5. Add to table
6. Exit program
6
```

After the program exits, the file contains:

57.0	154.23	22.49	35.4
15.40	35.22	98.2	52.21
800.0	800.0	800.0	800.0
280.87	301.20	295.14	329.18

Method Descriptions

```
/** Find the total of all values in the data array between 0 and rowSize for all
columns.
*
* @param data An array of data.
* @param rowSize The number of rows that were used in the array.
* @param colSize The number of columns in the array.
* @return The sum of the values in all of the data array between indices 0,0 and
rowsSize, colSize.
*/
public static double findGrandTotal(double[][]data, int rowSize, int colSize)
```

This method finds the sum of all of the values that are stored in the data array using a nested loop and returns this sum to the main program.

Example: If the data array contained the data below and rowSize was 3, then 78.0 would be returned.

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0

9.0	10.0	11.0	12.0

```
/** Find the sum of all of the values in the array that are between lowerAmount and
upperAmount (both inclusive).
*
* @param data The array of data.
* @param rowSize The number of rows in the array that have been used.
* @param colSize The number of columns in the array that have been used.
* @param lowerAmount The lower bound of values to include (inclusive).
* @param upperAmount The upper bound of values to include (inclusive).
* @return
*/
public static double findRangeTotal(double[][]data, int rowSize, int colSize, double
lowerAmount, double upperAmount)
```

This method finds the sum of all of the values that are stored in the data array that are between lowerAmount and upperAmount (both inclusive) using a nested loop and returns this sum to the main program.

Example: If the data array contained the data below, rowSize was 3 and lowerAmount was 10.0 and upperAmount was 12.0, then 33.0 (= 10.0 + 11.0 + 12.0) would be returned. To write this method, you will need to traverse the whole two-dimensional array, but only add in the values that are in the appropriate range.

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	10.0	11.0	12.0

```
/** Find a given label in an array. The array may be either oversize or
* perfect size. In other words, this method may be used with both rows and cols.
* @param array The array to be searched. This array is not sorted and should not be
sorted.
* @param arraySize The number of elements used in the array.
* @param label The label to find in the array.
* @return The index in the array where label was found, or -1 if label was not found.
*/
// This method is written in such a way that it can be used for either an
// oversize or perfect size array
public static int findLabelIndex(String[] array, int arraySize, String label)
```

This method is used to find either the row or column labels. By including an arraySize parameter, I made it possible to use this method in both the perfect size and oversize array case. Remember to use .equals() instead of == with the String contents. This method will be used in the next two methods. This kind of method is called a helper method, BTW, since it will normally only be used to help other methods get their work done.

Example: If cols was below, arraySize was 4 and label was “Feb”, then 1 would be returned.

Jan	Feb	Mar	Apr

```
/** Find the total of the data elements for a given month.
*
* @param cols The names of the columns (the months).
* @param data The data stored in each row and column.
* @param rowSize The number of rows in the data array.
* @param month The month where the data is sought.
* @return The sum of the values in the data array, in the column with label month.
*/
public static double findMonthlyTotal(String[] cols, double[][]data, int rowSize,
String month)
```

This method finds the monthly total in the data array. It does this by first finding the index where the String month is in the array. Then it uses this index to add up the proper column in data.

For example: If the data array is below, rowSize is 3 and cols contains {“Jan”, “Feb”, “Mar”, “Apr”} and month is “Mar”, then 21.0 (3.0 + 7.0 + 11.0) will be returned.

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	10.0	11.0	12.0

```
/** Find the sum of all of the values in the data array with row label item.
*
* @param rows The names of the rows.
* @param rowSize The number of rows in rows and data that were used.
* @param colSize The number of cols in data.
* @param data The data array.
* @param item The name of the row that should be summed.
* @return The sum of the values in the row that is labeled item.
*/
public static double findItemTotal(String[] rows, int rowSize, int colSize,
double[][]data, String item)
```

This method finds the total of the items that are in a given row for all four months.

Example: If rows contains {“Utilities”, “Rent”, “Entertainment”, “Food”}, rowSize is 4, colSize is 4, data is shown below, and item contains “Rent” then 26.0 (= 5.0 + 6.0 + 7.0 + 8.0) will be returned because rent is in the second position of the rows array so we had the second row of data.

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	10.0	11.0	12.0
13.0	14.0	15.0	16.0

```
/** Update an existing row/column or add another line to the rows and data arrays.
*
* @param rows The names of the rows (oversize array)
* @param rowSize The number of rows in the rows and data arrays that have been used.
* @param cols The  names of the columns (perfect size array).
* @param data The data that is stored for each row and column.
* @param item The item to be added to the array. This may go in an existing row or
may form a new rows if item is
* not contained in rows.
* @param month The name of the column to be updated.
* @param amount The amount to be added to the existing element, or stored in the new
element.
* @return The number of rows that have been used in rows and data.  This could be the
same as rowSize, or may be
* one larger if the row name is not already in the rows array.
*/
public static int addToData(String[] rows, int rowSize, String[] cols, double[][]
data, String item, String month, double amount)
```

This method is used to add data to the table. If the item that is given exists in the row table, then the element of the two-dimensional array in the given month is incremented by amount. In this case, rowSize will be returned because the size of the rows and data arrays has not been changed. If however, item is not in the rows array, then it is added to both the rows array and the data array in the last row. rowSize will be incremented by one and returned.

Example: Suppose rows contains {“Utilities”, “Rent”, “Entertainment”, “Food”}, rowSize is 4, cols contains {“Jan”, “Feb”, “Mar”, “Apr”}, and data is given below.

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	10.0	11.0	12.0
13.0	14.0	15.0	16.0

If item contains “Entertainment”, month contains “Feb”, and amount contains 20.0, then 10.0 in the data array will be replaced by 30.0 (as shown below) and 4 will be returned (since data and rows have not changed size).

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	30.0	11.0	12.0
13.0	14.0	15.0	16.0

If, however, item contains “Clothing”, month contains “Feb” and amount contains 20.0, the the data array will be expanded as shown below. Rows will contain {“Utilities”, “Rent”, “Entertainment”, “Food”, “Clothing”} and 5 will be returned

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	10.0	11.0	12.0
13.0	14.0	15.0	16.0
0.0	20.0	0.0	0.0

Suggested Order of Implementation

For this project, I’ve created lots of starter code with some methods already written, some methods for you to write (with stubs, so your code can still run), some method calls for you to do (since this seems to still be a problem for many people), and some gaps for you to fill in.

To make it easier for you to implement this code, I’m using a cool eclipse feature. When you put a comment that starts with the word TODO, eclipse will put a little pencil and paper icon next to it on the left hand side of the line numbers in the window that contains the code. This will make it easier for you to find the places where I’ve removed code so you can fill in the gaps.

Here is a good order to implement this code.

1. Construct the rows, cols and data arrays in the main program. Rows is an oversize array. Cols is a perfect size array (with four month headers: “Jan”, “Feb”, “Mar”, and “Apr”). Data is the two-dimensional array. The oversize arrays should be constructed with 100 rows.
2. Call the readFile method.
3. Call the writeFile method.
4. Call the performOperations method.

When you’ve accomplished these tasks, the program should be able to fully run as you complete the remaining methods. I’ve put these methods in the order of easiest to hardest.

5. Write and call the findGrandTotal method. This method is called in performOperations.
6. Write and call the findRangeTotal method. The method is called in performOperations.
7. Write the findLabelIndex method. This method will be used in the next two methods.
8. Write and call the findMonthlyTotal method, calling the findLabelIndex method inside this method. The findMonthlyTotal method is called in the performOperations method.
9. Write and call the findItemTotal method, calling the findLabelIndex method inside this method. The findItemTotal method is called in the performOperations method.
10. Write and call the addToData method.

Submission Instructions: Submit your source code to the Project 10 Zylab in Zybooks in a .java file. The class name should be Project10.