## INSTRUCTIONS

Complete the project detailed in this document using valid, executable Java code. Submit your answer source code file (`DataCombiner.java`) ONLY to your instructor via eCampus no later than Friday 12/4/2020 @11:59PM. All other submitted formats will be ignored (zipped files or projects or folders). At the top of the file, include a block comment displaying **your last name, first name, project name and number, section number, lab section number, and date**. Start a new project and write your own code, your work will be compared with online sources as well as other students' work to detect plagiarism. **No late submissions area allowed**. You are highly encouraged to start working on this project ASAP.

Make sure you name your class and methods EXACTLY as provided in the instructions. Pay attention to the spelling and the case of letters in each. You **will** lose points for not following the naming criteria in the instructions. You may add extra methods as needed. Make sure to include comments describing what each block of code does, and what each method does, accepts (as parameters), and returns. Adding those comments will **earn 10 POINTS.**

## Background

Your project should read loan data from two separate text files (loanamounts.txt and loanrecipient.txt), process each, and generate two text files, one of ".txt" extension and one of ".arff" extension. ARFF files (Attribute-Relation File Format) files are text files that organize data differently than a regular txt or csv file. You may read more about arff files by accessing the Waikato web page. You may also refer to the attached samples files (Ant1.txt and Ant1.arff) to see the difference. As you may see, the data in the .txt file is organized in columns and rows. Column names are on top, and the data is provided in rows. Whereas Arff files define the data types (columns names) first, then the data is provided separately in each row. Creating an arff file in java is like the way of creating any text file, the only difference is the extension.

## INSTRUCTIONS

1. Download the two text files (loanamount.txt and loanrecepients.txt) into the same directory.
   a. **loanamounts.txt** – Total amount of student loans disbursed by quarter from 2014 to 2020.

| Column Name | Type | Description |
|---|---|---|
| Year | Number | Year of the data |
| Quarter | Number | Quarter of the data |
| Stafford Subsidized | Currency | Amount of Stafford subsidized loans disbursed, in billions of dollars. |
| Stafford Unsubsidized | Currency | Amount of Stafford unsubsidized loans disbursed, in billions of dollars. |
| Grad PLUS | Currency | Amount of Grad PLUS loans disbursed, in billions of dollars. |
| Parent PLUS | Currency | Amount of Parent PLUS loans disbursed, in billions of dollars. |
| Perkins | Currency | Amount of Perkins loans disbursed, in billions of dollars. |
| Consolidation | Currency | Amount of consolidation loans disbursed, in billions of dollars. |

   b. **loanrecipient.txt** – Total number of student loan recipients by quarter from 2014 to 2020.

| Column Name | Type | Description |
|---|---|---|
| Year | Number | Year of the data. |
| Quarter | Number | Quarter of the data |
| Stafford Subsidized | Number | Number of Stafford subsidized loan recipients, in millions. |
| Stafford Unsubsidized | Number | Number of Stafford unsubsidized loan recipients, in millions. |
| Grad PLUS | Number | Number of Grad PLUS loan recipients, in millions. |
| Parent PLUS | Number | Number of Parent PLUS loan recipients, in millions. |
| Perkins | Number | Number of Perkins loan recipients, in millions. |
| Consolidation | Number | Number of consolidation loan recipients, in millions. |

2. Create a Java project named **DataCombiner.java** The project will read data from two text files, process the two files, and generate two text files one with ".txt" extension and one with ".arff" extension.

In **`DataCombiner.java`** the implement the following:

1. Create a method called `public static float [][] read_data(String strFileName)`, which accepts one parameter (`strFileName`) and returns a 2-D array. The accepted parameter could be any of the data files (**loanamounts.txt** or **loanreceipients.txt**), and the returned 2-D (24 rows X 7 columns) array is the array containing the numerical data in that processed file. Before this method is called, display a message to the user asking whether the files are in the workspace "local" or not "external". If the user types "local" then read each file directly from the workspace, otherwise, ask the user to provide the directory to the files, then read each file from that directory. **[20 points]**
    a. Include catch/ try, or exception handler every time your functions try to read file. Provide an error message (file not found) if the file name or location is incorrect or invalid.

2. Create a method called `public static String [][] get_years_quarters(String strFile)`, which accepts one parameter (loanamounts.txt), and returns a 2-D array (24 rows X 2 columns). The first columns should contain the years, and the second column should contain the quarters of the processed text file. **[10 points]**

3. Create a method called `public static float[][] cost_per_student (float[][]loan_data,float[][]recipients_data)` which accepts TWO 2-D arrays and returns a 2-D array (24 rows X 21 columns). The two parameters are the data acquired from each file in Step 1. This method calculates the total loan amount per student by implementing the following formula: **[30 points]**
    a. Calculates the `loan amount per student` by which equals:
    (Loan amount * 1,000,000,000)/( Recipients * 1,000,000)

$$\frac{loan\ amount * 1{,}000{,}000{,}000}{recipients * 1{,}000{,}000}$$

    The $loan\ amount$ is acquired from `loan_data` and the $recipients$ are acquired from `recipients_data.`
    b. Returns a 2-D array consisting of data from the two passed parameters 2-D arrays in addition to the `loan amount per students` calculated in Step 3.a. This 2-D array should have its columns sorted (from left to right) first column from `loan_data`, second column

from `recipients_data,` followed by the `loan amount per students`. Note that this 2-D array should only include numerical data. The returned 2-D array should look like the data in Figure 1.

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 255.30 | 28.20 | 9053.19 | 350.70 | 25.50 | 13752.94 | 606.00 | 30.70 | 19739.41 | 37.80 | 0.90 | 42000.00 | 62.20 | 3.10 | 20064.52 | 8.30 | 3.00 | 2766.67 | 373.60 | 11.80 | 31661.02 |
| 254.10 | 28.00 | 9075.00 | 355.10 | 25.50 | 13925.49 | 609.20 | 30.50 | 19973.77 | 38.60 | 0.90 | 42888.89 | 61.90 | 3.10 | 19967.74 | 8.20 | 2.90 | 2827.59 | 378.80 | 11.80 | 32101.69 |
| 258.80 | 28.70 | 9017.42 | 370.80 | 26.30 | 14098.86 | 629.70 | 31.40 | 20054.14 | 41.20 | 1.00 | 41200.00 | 65.10 | 3.20 | 20343.75 | 8.20 | 2.90 | 2827.59 | 385.80 | 11.80 | 32694.92 |
| 258.00 | 28.90 | 8927.34 | 373.00 | 26.60 | 14022.56 | 630.90 | 31.70 | 19902.21 | 40.90 | 1.00 | 40900.00 | 65.10 | 3.20 | 20343.75 | 8.20 | 3.00 | 2733.33 | 394.90 | 11.90 | 33184.87 |
| 262.90 | 28.80 | 9128.47 | 388.40 | 26.50 | 14656.60 | 651.30 | 31.50 | 20676.19 | 43.40 | 1.00 | 43400.00 | 68.70 | 3.20 | 21468.75 | 8.30 | 2.90 | 2862.07 | 402.80 | 11.90 | 33848.74 |
| 260.70 | 28.50 | 9147.37 | 390.80 | 26.40 | 14803.03 | 651.60 | 31.30 | 20817.89 | 44.10 | 1.00 | 44100.00 | 68.00 | 3.10 | 21935.48 | 8.20 | 2.90 | 2827.59 | 410.30 | 11.90 | 34478.99 |
| 264.80 | 29.10 | 9099.66 | 404.90 | 27.10 | 14940.96 | 669.70 | 32.10 | 20862.93 | 46.80 | 1.00 | 46800.00 | 71.10 | 3.30 | 21545.45 | 8.10 | 2.80 | 2892.86 | 416.70 | 11.90 | 35016.81 |

*Figure 1: Structure of the 2-D array returned by cost_per_student method*

4. Create a method called `public static void write_txt_data (String [][] years_quarters, float[][] full_data, String strLocation)` which accepts three parameters; the 2-D array from Step 2 which holds the years and quarters, the 2-D array from Step 3 which holds the whole loans' data; and the location where to output the final text file. This method should create a text file named [**your first name_ your last_name_data.txt**]. The text file should have 24 rows and 9 columns, the first column is named Years, second column is named Quarter, followed by the loan name_cost_per_students for each loan ($) [e.g., Stafford Subsidized cost_per_student ($), Stafford Unsubsidized cost_per_student ($),, Grad PLUS cost_per_student ($), etc.,,,) see Figure 2. **[20 points]**

| Year | Quarter | Stafford Subsidized | Stafford Unsubsidized | Stafford Combined | Grad PLUS | Parent PLUS | Perkins Outstanding | Consolidation Outstanding |
|---|---|---|---|---|---|---|---|---|
| 2014 | Q2 | 9,053 | 13,753 | 19,739 | 42,000 | 20,065 | 2,767 | 31,661 |
| 2014 | Q3 | 9,075 | 13,925 | 19,974 | 42,889 | 19,968 | 2,828 | 32,102 |
| 2014 | Q4 | 9,017 | 14,099 | 20,054 | 41,200 | 20,344 | 2,828 | 32,695 |
| 2015 | Q1 | 8,927 | 14,023 | 19,902 | 40,900 | 20,344 | 2,733 | 33,185 |
| 2015 | Q2 | 9,128 | 14,657 | 20,676 | 43,400 | 21,469 | 2,862 | 33,849 |
| 2015 | Q3 | 9,147 | 14,803 | 20,818 | 44,100 | 21,935 | 2,828 | 34,479 |
| 2015 | Q4 | 9,100 | 14,941 | 20,863 | 46,800 | 21,545 | 2,893 | 35,017 |
| 2016 | Q1 | 9,014 | 14,894 | 20,829 | 46,600 | 20,912 | 2,793 | 35,300 |
| 2016 | Q2 | 9,212 | 15,451 | 21,520 | 49,400 | 22,788 | 2,929 | 35,933 |
| 2016 | Q3 | 9,228 | 15,627 | 21,639 | 50,200 | 22,576 | 2,963 | 36,600 |
| 2016 | Q4 | 9,156 | 15,687 | 21,663 | 48,182 | 22,229 | 2,926 | 37,275 |
| 2017 | Q1 | 9,081 | 15,589 | 21,503 | 48,000 | 22,143 | 2,926 | 38,017 |
| 2017 | Q2 | 9,269 | 16,115 | 22,153 | 50,636 | 23,971 | 3,038 | 38,708 |
| 2017 | Q3 | 9,307 | 16,315 | 22,366 | 51,455 | 23,676 | 3,000 | 39,358 |
| 2017 | Q4 | 9,258 | 16,339 | 22,356 | 49,750 | 23,971 | 3,040 | 40,025 |
| 2018 | Q1 | 9,196 | 16,313 | 22,288 | 49,667 | 23,914 | 3,040 | 40,750 |
| 2018 | Q2 | 9,369 | 16,841 | 22,888 | 52,333 | 25,057 | 3,040 | 41,672 |
| 2018 | Q3 | 9,423 | 17,064 | 23,142 | 53,250 | 25,500 | 3,083 | 42,176 |
| 2018 | Q4 | 9,375 | 17,059 | 23,175 | 51,538 | 24,972 | 3,087 | 42,689 |
| 2019 | Q1 | 9,320 | 17,045 | 23,054 | 47,533 | 24,944 | 3,136 | 43,695 |
| 2019 | Q2 | 9,492 | 17,544 | 23,671 | 54,385 | 26,829 | 3,000 | 44,220 |
| 2019 | Q3 | 9,533 | 17,816 | 23,975 | 55,308 | 26,543 | 3,000 | 44,780 |
| 2019 | Q4 | 9,515 | 17,855 | 23,997 | 53,714 | 26,694 | 3,050 | 45,821 |
| 2020 | Q1 | 9,446 | 17,865 | 23,970 | 53,786 | 26,556 | 3,105 | 46,359 |

*Figure 2: Final txt file structure and data*

5. Create a method called `public static String txt2Arff (String strFile)` which accepts one parameter of type string and returns a string. This method should do the following:
   a. Accepts the text file's name and location (created in step 4.b). **[30 points]**
   b. Creates a file in the same directory as the source file named [**your first name_ your last_name_data.arff**] but a different format (e.g., accepts .txt and writes .arff). The arff file should reflect the same data in the txt file (see Figure 3) and should include the following:
      i. Start with (@relation [file_name]).
      ii. Should have each column formatted as (@attribute [column_name] [data type])
      iii. Use numeric for all columns except for Quarter, use text.

c. Returns a confirmation message that stating that the .arff file has been created successfully.

```
@relation File_name

@attribute Year numeric
@attribute Quarter text
@attribute Stafford Subsidized numeric
@attribute Stafford unsubsidized numeric
@attribute Stafford combined numeric
@attribute Grad plus numeric
@attribute Parent plus numeric
@attribute Perkins plus numeric
@attribute Consolidation outstanding numeric


@data
2014,Q2,9053,13753,19739,42000,20065,2767,31661
2014,Q3,9075,13925, 19974,42889,19968,2828, 32102

...
...
...
...
```

*Figure 3: Arff file format generated by the txt2Arff method*

6. Create a main method which does the following: **[20 points]**
   a. Asks whether files are local or external
   b. Calls the `read_data(strFile)` twice, once for each of the text files (loanamount.txt and loanreceipience.txt).
   c. Stores each of the read files in a 2-D array.
   d. Calls the `get_years_quarters(strFile)`, and stores the returned data into a 2-D array.
   e. Calls the method `cost_per_student (loan_data, recipients_data)` and stores the returned data in a 2-D array.
   f. Calls the method `write_txt_data (years_quarters, full_data, strLocation)` using the data from Steps 5.e, and 5.d) which writes the text file (final_file.txt) and returns nothing.
   g. Calls the method `txt2Arff (String strFile)` using the file created in step 5 which writes the arff file to the same directory as the text file.

## Pseudo code:

```
Display Welcome message:

                ==========================================
            Your last name, Your first name, CS110 Project #2
              ==========================================

Display message [are your files local or external?]

Accept user input: choice

If choice == "external":
     Display message [please provide directory to files]
     set strFileName = user input +"/loanamounts.txt"

Else:
     set strFileName = "./loanamounts.txt"

Read_data from file loanamounts.txt using strFileName
Read_data from file loanreceipients.txt using strFileName
get_years_quarters from file any of the text files

Write txt file

     [your first name_ your last_name_data.txt].

Write arff file

     [your first name_ your last_name_data.arff].
```

# Grading Rubric

This assignment is worth 140 points. It will be graded by your instructor using this rubric, with partial credit awarded as appropriate:

| | | | |
|---|---|---|---|
| Step 1 | 20 points total | Step 6 | 20 points total |
| Step 2 | 10 points total | Documentation | 10 points total |
| Step 3 | 30 points total | | |
| Step 4 | 20 points total | | |
| Step 5 | 30 points total | | |