Report

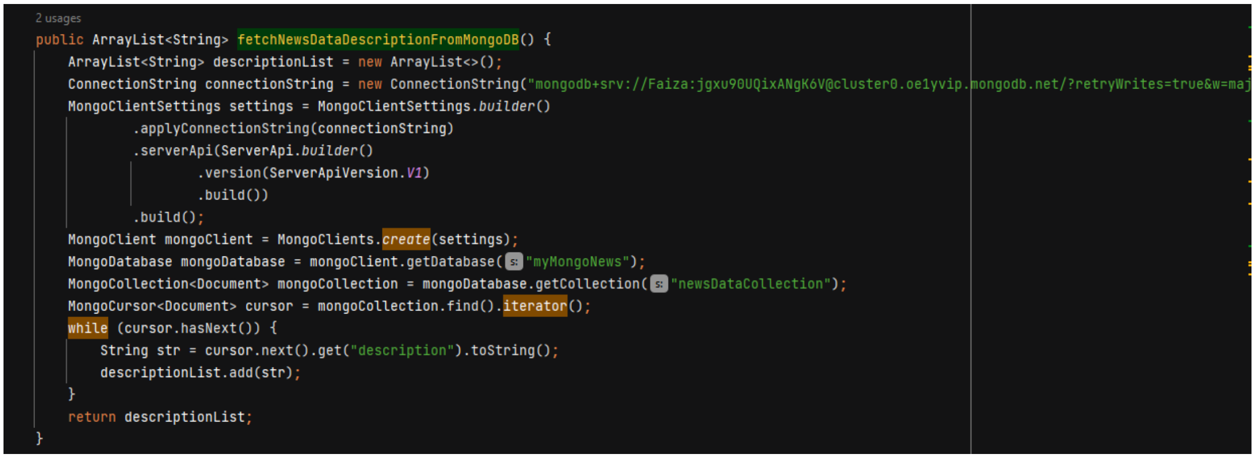
Assignment No. – 4

Problem -3

To perform this task, I considered the processed News description (ignored other fields) that I obtained from NewsAPI and stored in MongoDB in Assignment No. – 3.

Step - 1)

In order to perform this task, I first fetched the News Article containing just the News "Description". Refer Figure 1 to see the method to connect to MongoDB and to fetch just the description from the News Article dataset.

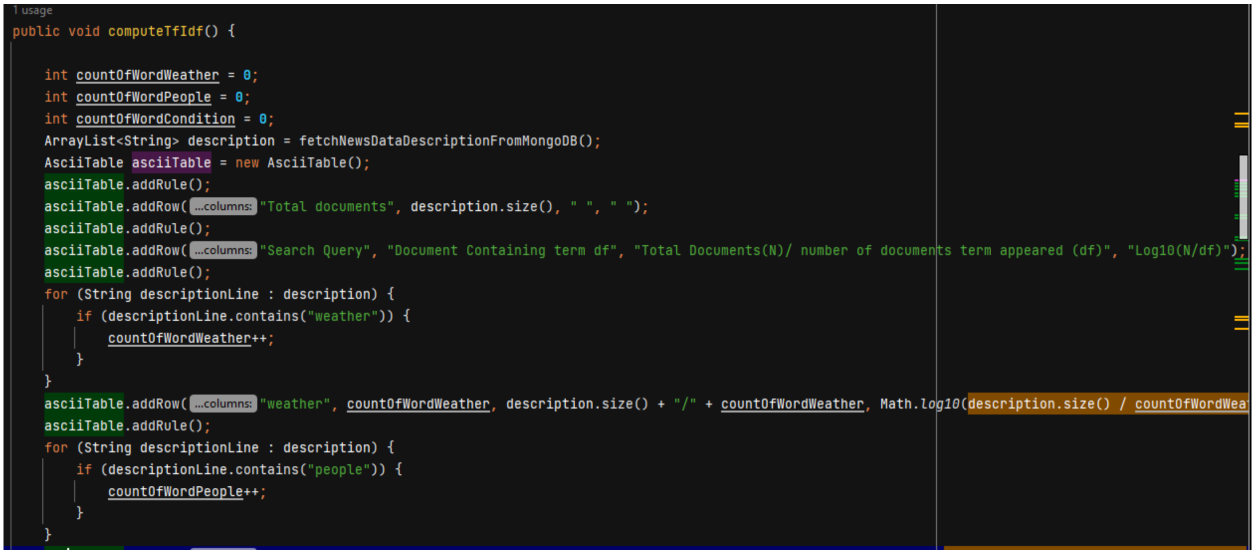


```
2 usages
public ArrayList<String> fetchNewsDataDescriptionFromMongoDB() {
    ArrayList<String> descriptionList = new ArrayList<>();
    ConnectionString connectionString = new ConnectionString("mongodb+srv://Faiza:jgxu90UQixANgK6V@cluster0.oe1yvip.mongodb.net/?retryWrites=true&w=maj
    MongoClientSettings settings = MongoClientSettings.builder()
            .applyConnectionString(connectionString)
            .serverApi(ServerApi.builder()
                    .version(ServerApiVersion.V1)
                    .build())
            .build();
    MongoClient mongoClient = MongoClients.create(settings);
    MongoDatabase mongoDatabase = mongoClient.getDatabase( s: "myMongoNews");
    MongoCollection<Document> mongoCollection = mongoDatabase.getCollection( s: "newsDataCollection");
    MongoCursor<Document> cursor = mongoCollection.find().iterator();
    while (cursor.hasNext()) {
        String str = cursor.next().get("description").toString();
        descriptionList.add(str);
    }
    return descriptionList;
}
```

Figure 1: Represents the method to fetch description from MongoDB

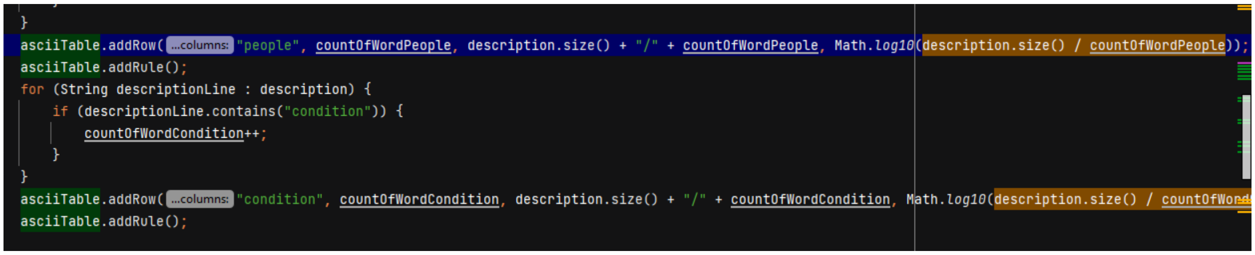
Step - 2)

- In order to compute TF-IDF (term frequency-inverse document frequency), created a method **computeTfIdf()**. In which I need to show the in how many documents the words "weather", "people" and "condition" is present.
- I initialized the word count of each words to 0.
- I made the ArrayList of string to call the description from method **fetchDescriptionFromMongoDB()**.
- I used for each loop to check each line of description containing the word. If it contains the word increment the count.
- I used AsciiTable to print the information in a table format.
- I also used file writer to create a different file named "TF-IDF-table.txt"to print the table. Refer Figure 4
- Refer Figure 2,3 and 4, to see the code to **computeTfIdf()**.



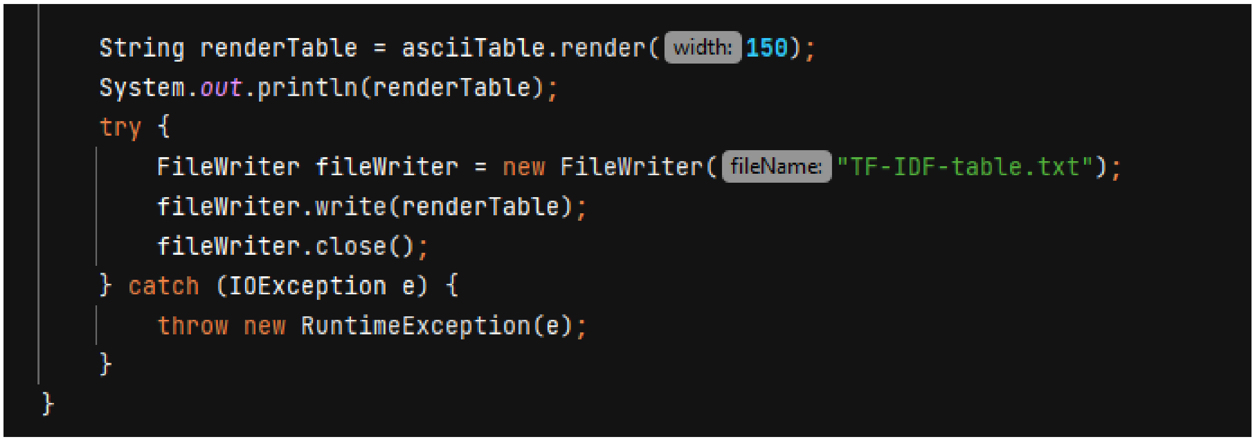
```java
1 usage
public void computeTfIdf() {

    int countOfWordWeather = 0;
    int countOfWordPeople = 0;
    int countOfWordCondition = 0;
    ArrayList<String> description = fetchNewsDataDescriptionFromMongoDB();
    AsciiTable asciiTable = new AsciiTable();
    asciiTable.addRule();
    asciiTable.addRow( ...columns: "Total documents", description.size(), " ", " ");
    asciiTable.addRule();
    asciiTable.addRow( ...columns: "Search Query", "Document Containing term df", "Total Documents(N)/ number of documents term appeared (df)", "Log10(N/df)");
    asciiTable.addRule();
    for (String descriptionLine : description) {
        if (descriptionLine.contains("weather")) {
            countOfWordWeather++;
        }
    }
    asciiTable.addRow( ...columns: "weather", countOfWordWeather, description.size() + "/" + countOfWordWeather, Math.log10(description.size() / countOfWordWea
    asciiTable.addRule();
    for (String descriptionLine : description) {
        if (descriptionLine.contains("people")) {
            countOfWordPeople++;
        }
    }
```

Figure 2: Represents the method to compute TF-IDF



```java
    }
    asciiTable.addRow( ...columns: "people", countOfWordPeople, description.size() + "/" + countOfWordPeople, Math.log10(description.size() / countOfWordPeople));
    asciiTable.addRule();
    for (String descriptionLine : description) {
        if (descriptionLine.contains("condition")) {
            countOfWordCondition++;
        }
    }
    asciiTable.addRow( ...columns: "condition", countOfWordCondition, description.size() + "/" + countOfWordCondition, Math.log10(description.size() / countOfWord
    asciiTable.addRule();
```

Figure 3: Represents the method to compute TF-IDF

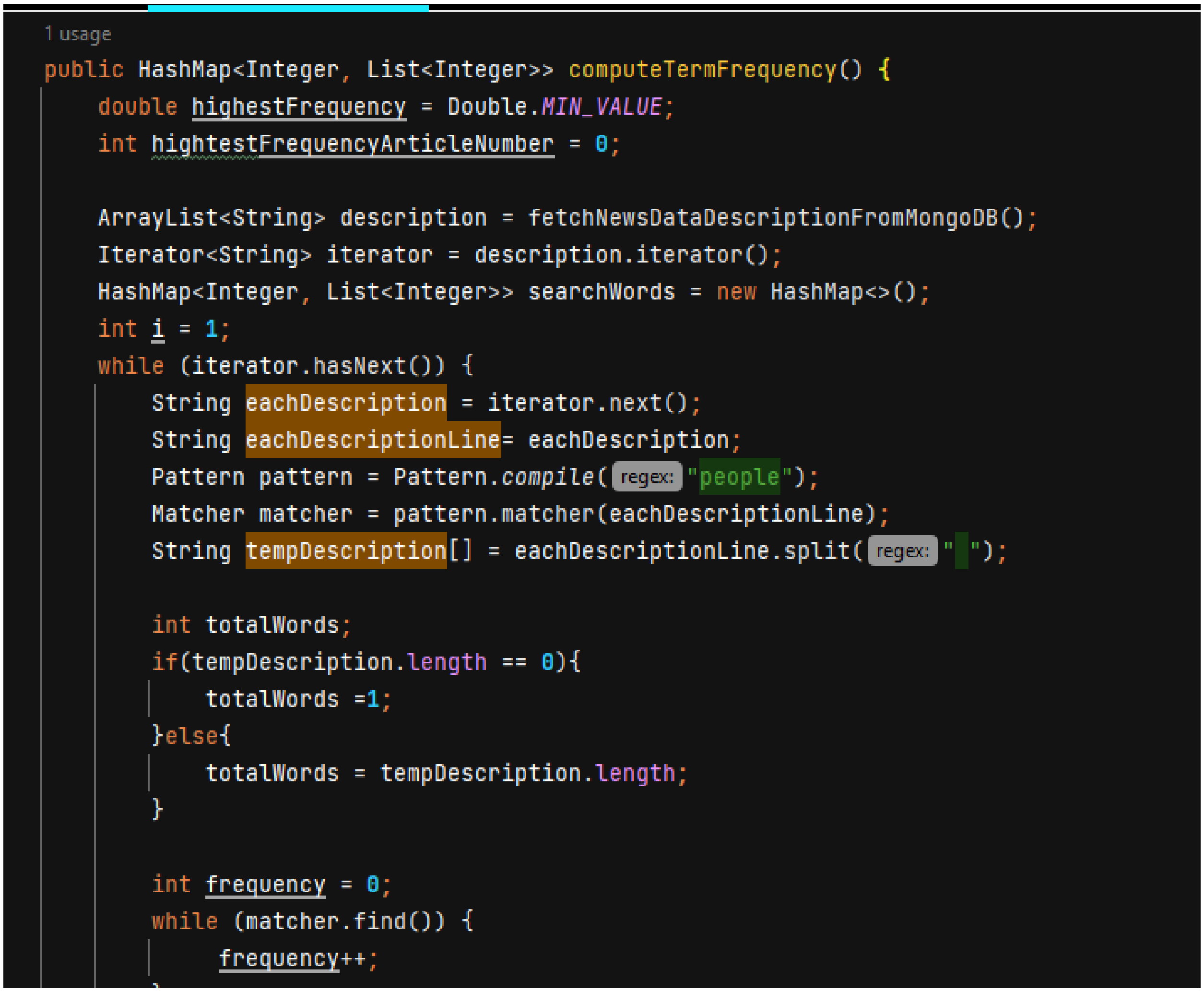


```java
    String renderTable = asciiTable.render( width: 150);
    System.out.println(renderTable);
    try {
        FileWriter fileWriter = new FileWriter( fileName: "TF-IDF-table.txt");
        fileWriter.write(renderTable);
        fileWriter.close();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Figure 4: Represents the method to compute TF-IDF (continution)

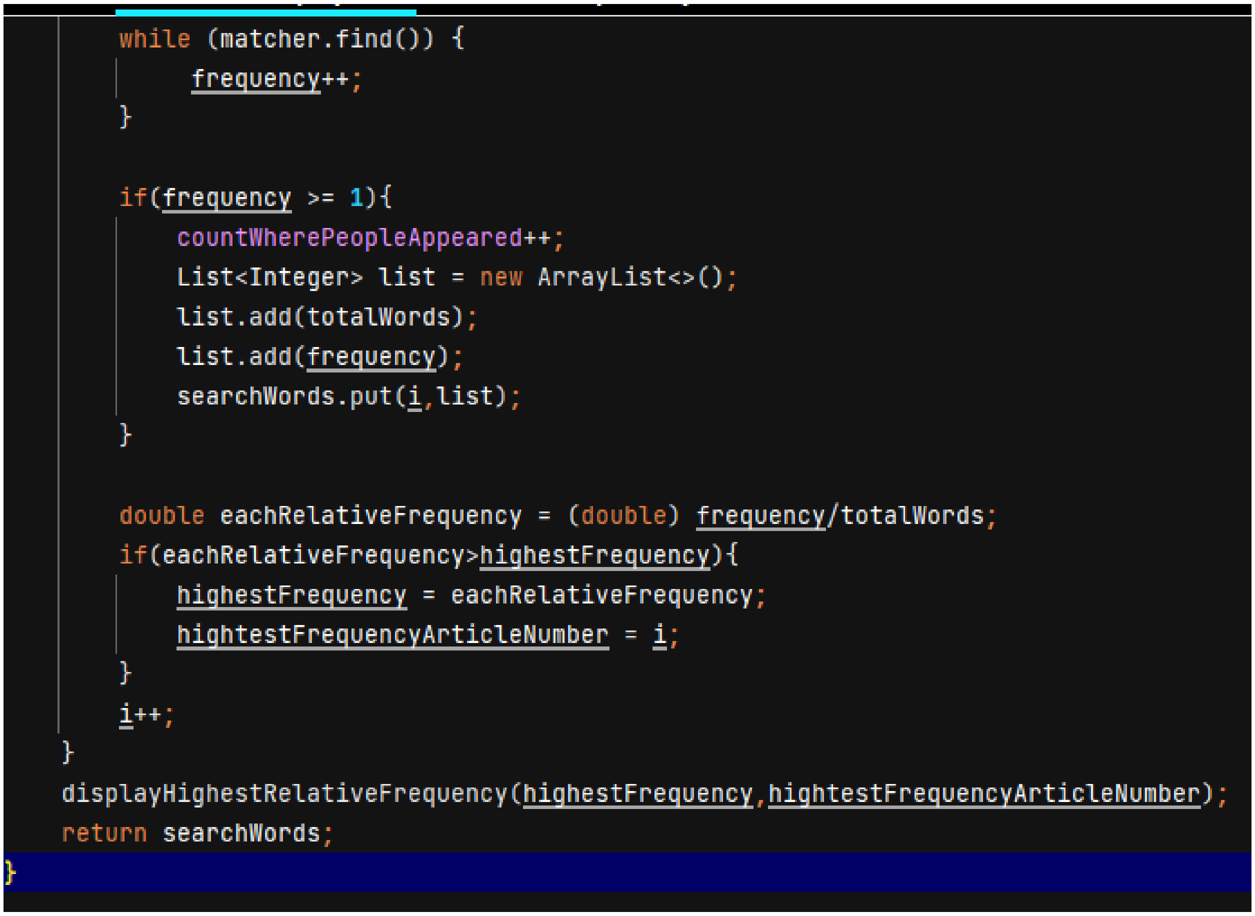
Step-3)

- After performing the above task, I now need to calculate the highest occurrence of the word “people”. For that, I created a method **computeTermFrequency()**.
- I fetched the description by calling the method **fetchNewsDescriptionFromMongoDB()** and stored it in a ArrayList of string.
- Then I used Iterator to get each line of description to search for word “people”.
- I used **Pattern matcher** to check if the word “people” is present in description. If yes, then increment the frequency.
- Now I need to check the highest frequency of word occurred in that document for that, I need to check if **highestFrequency** = **eachRelativefrequency**, initialize **highestFrequencyArticleNumber** to **i** and then increment i. “i” was initialized to 1.
- Refer figure 5 and 6 to see the code to computeTermFrequency.



```
1 usage
public HashMap<Integer, List<Integer>> computeTermFrequency() {
    double highestFrequency = Double.MIN_VALUE;
    int hightestFrequencyArticleNumber = 0;

    ArrayList<String> description = fetchNewsDataDescriptionFromMongoDB();
    Iterator<String> iterator = description.iterator();
    HashMap<Integer, List<Integer>> searchWords = new HashMap<>();
    int i = 1;
    while (iterator.hasNext()) {
        String eachDescription = iterator.next();
        String eachDescriptionLine= eachDescription;
        Pattern pattern = Pattern.compile( regex: "people");
        Matcher matcher = pattern.matcher(eachDescriptionLine);
        String tempDescription[] = eachDescriptionLine.split( regex: " ");

        int totalWords;
        if(tempDescription.length == 0){
            totalWords =1;
        }else{
            totalWords = tempDescription.length;
        }

        int frequency = 0;
        while (matcher.find()) {
            frequency++;
```

Figure 5: Represents the method computeTermFrequency()

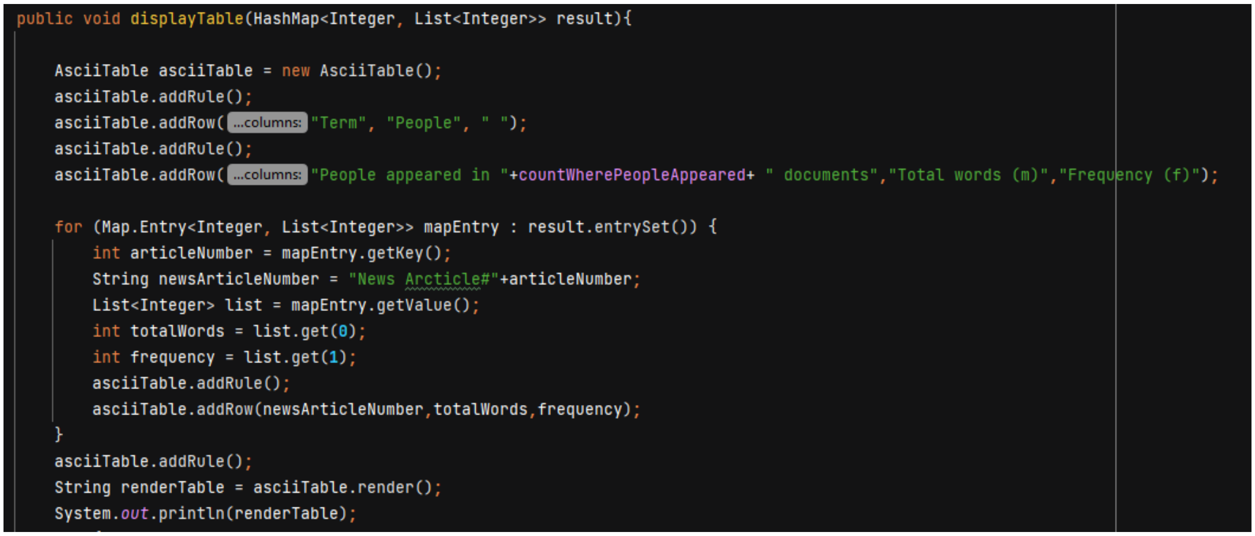


```
        while (matcher.find()) {
            frequency++;
        }

        if(frequency >= 1){
            countWherePeopleAppeared++;
            List<Integer> list = new ArrayList<>();
            list.add(totalWords);
            list.add(frequency);
            searchWords.put(i,list);
        }

        double eachRelativeFrequency = (double) frequency/totalWords;
        if(eachRelativeFrequency>highestFrequency){
            highestFrequency = eachRelativeFrequency;
            hightestFrequencyArticleNumber = i;
        }
        i++;
    }
    displayHighestRelativeFrequency(highestFrequency,hightestFrequencyArticleNumber);
    return searchWords;
}
```

Figure 6: Represents the method computeTermFrequency() (continution)

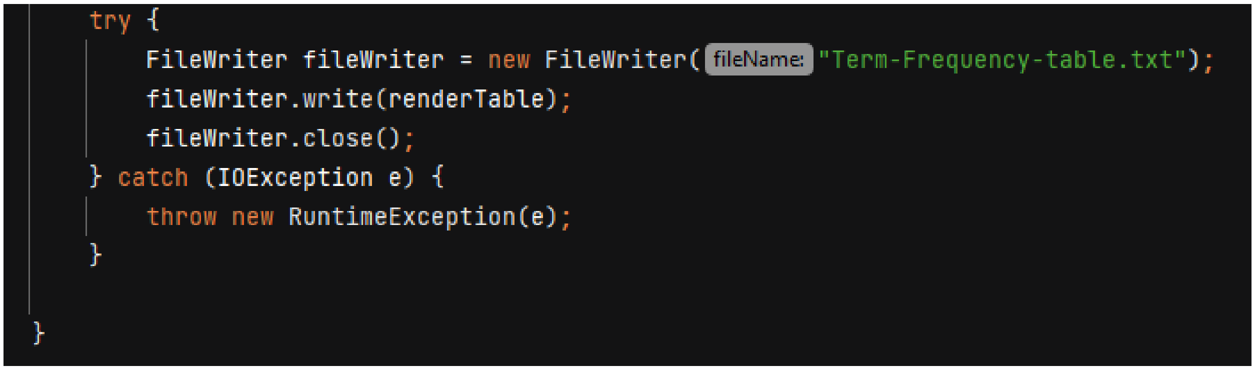
<u>**Step-4)**</u>

- I create a method **displayTable** to display the table containing the columns – “People appeared in documents”, “Total words” and “Frequency”.
- I used **AsciiTable** to render the information of the table and print it.
- I also used file writer to create a different file named “Term-Frequency-table.txt”to print the table. Refer Figure 8.
- Refer Figure 7 and 8, to see the code to displayTable.



```
public void displayTable(HashMap<Integer, List<Integer>> result){

    AsciiTable asciiTable = new AsciiTable();
    asciiTable.addRule();
    asciiTable.addRow(...columns: "Term", "People", " ");
    asciiTable.addRule();
    asciiTable.addRow(...columns: "People appeared in "+countWherePeopleAppeared+ " documents","Total words (m)","Frequency (f)");

    for (Map.Entry<Integer, List<Integer>> mapEntry : result.entrySet()) {
        int articleNumber = mapEntry.getKey();
        String newsArticleNumber = "News Arcticle#"+articleNumber;
        List<Integer> list = mapEntry.getValue();
        int totalWords = list.get(0);
        int frequency = list.get(1);
        asciiTable.addRule();
        asciiTable.addRow(newsArticleNumber,totalWords,frequency);
    }
    asciiTable.addRule();
    String renderTable = asciiTable.render();
    System.out.println(renderTable);
```

Figure 7: Represents the method to display table

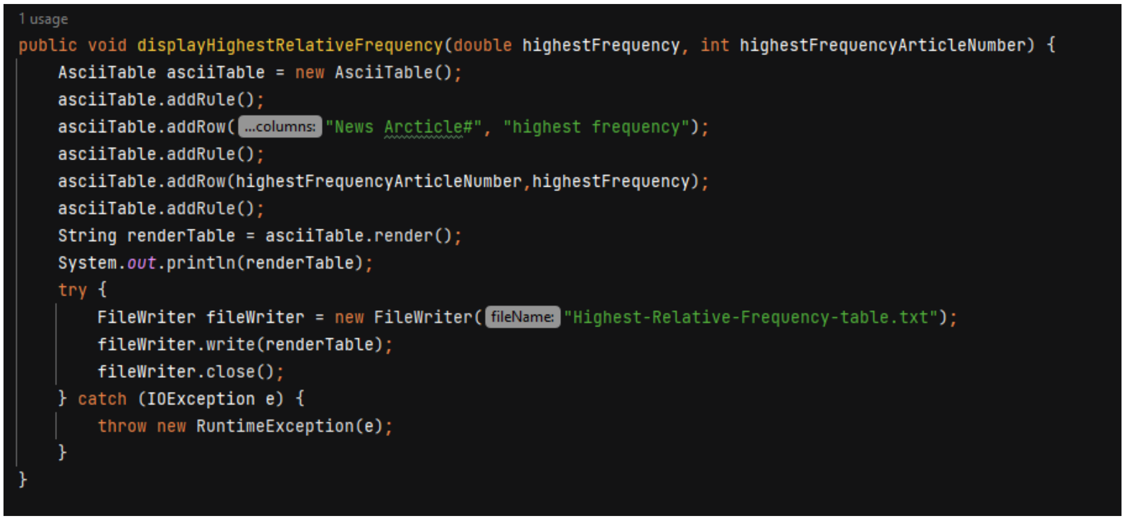


```
        try {
            FileWriter fileWriter = new FileWriter(fileName: "Term-Frequency-table.txt");
            fileWriter.write(renderTable);
            fileWriter.close();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

    }
```

Figure 8: Represents the method to display table (continution)

Step – 5)

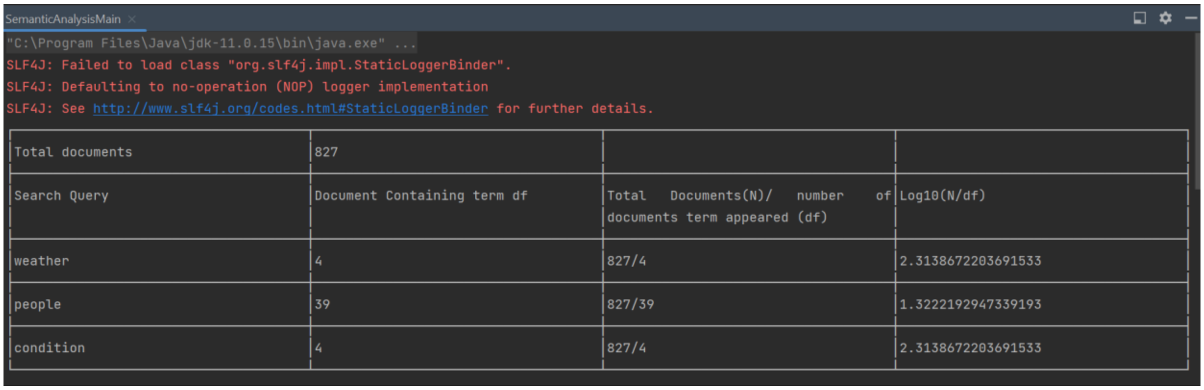
- Highest Relative frequency is nothing but the Frequency (f) / Total Words (m). I calculated the frequency and totalWords in method computeTermFrequency().
- Now I want to display it in table format. So I used AsciiTable to render the highest relative frequency in the table.
- I then called the **displayHighestRelativeFrequency()** method in computeTermFrequency().
- I also used file writer to create a different file named “Highest-Relative-Frequency-table.txt”to print the table. Refer Figure 9.
- Refer Figure 9, to see the code to displayHighestRelativeFrequency().



```
1 usage
public void displayHighestRelativeFrequency(double highestFrequency, int highestFrequencyArticleNumber) {
    AsciiTable asciiTable = new AsciiTable();
    asciiTable.addRule();
    asciiTable.addRow( ...columns: "News Arcticle#", "highest frequency");
    asciiTable.addRule();
    asciiTable.addRow(highestFrequencyArticleNumber,highestFrequency);
    asciiTable.addRule();
    String renderTable = asciiTable.render();
    System.out.println(renderTable);
    try {
        FileWriter fileWriter = new FileWriter( fileName: "Highest-Relative-Frequency-table.txt");
        fileWriter.write(renderTable);
        fileWriter.close();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

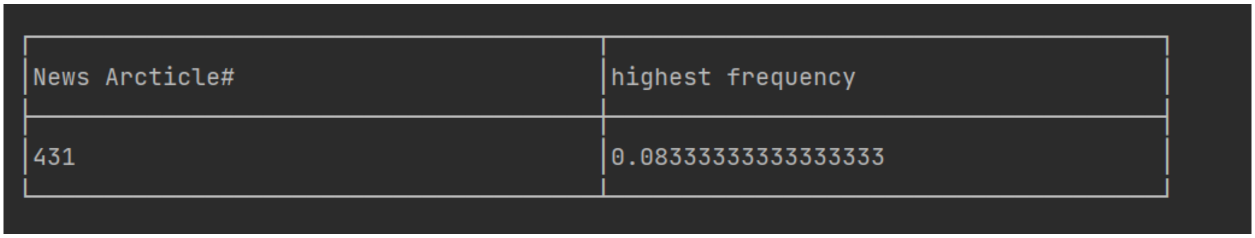
Figure 9: Represents the method to display the highest relative frequency

Output:



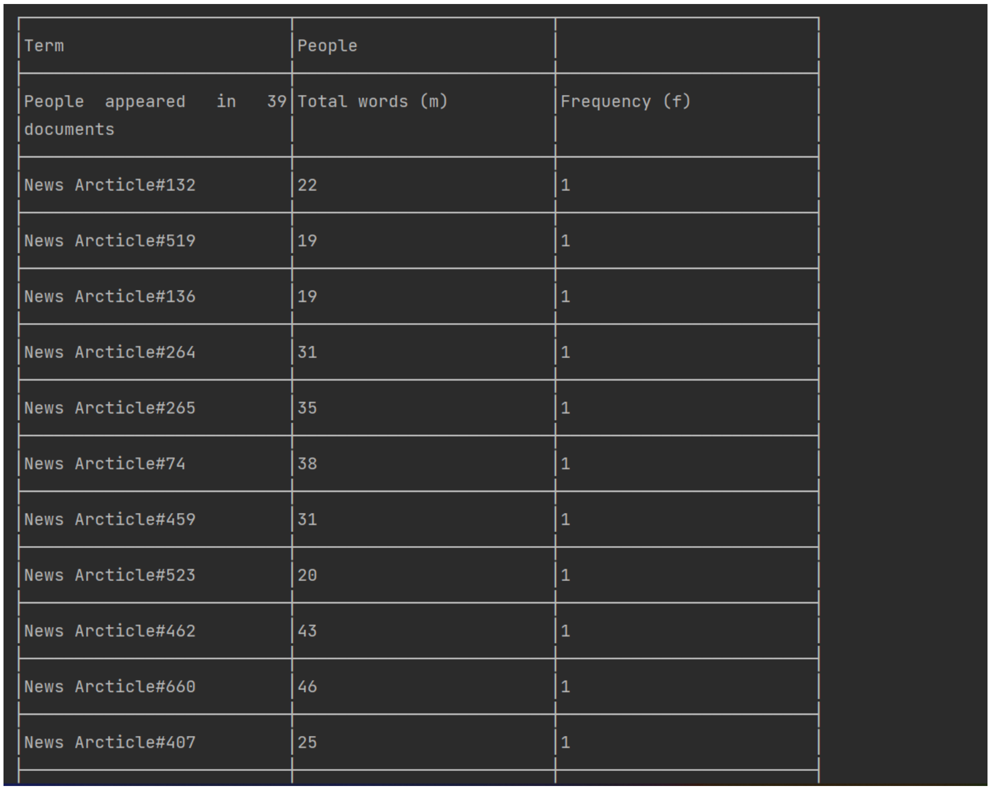
```
"C:\Program Files\Java\jdk-11.0.15\bin\java.exe" ...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
┌──────────────────────────────┬─────────────────────────────┬─────────────────────────────┬─────────────────────────────┐
│Total documents               │827                          │                             │                             │
├──────────────────────────────┼─────────────────────────────┼─────────────────────────────┼─────────────────────────────┤
│Search Query                  │Document Containing term df  │Total   Documents(N)/   number    of│Log10(N/df)                  │
│                              │                             │documents term appeared (df) │                             │
├──────────────────────────────┼─────────────────────────────┼─────────────────────────────┼─────────────────────────────┤
│weather                       │4                            │827/4                        │2.3138672203691533           │
├──────────────────────────────┼─────────────────────────────┼─────────────────────────────┼─────────────────────────────┤
│people                        │39                           │827/39                       │1.3222192947339193           │
├──────────────────────────────┼─────────────────────────────┼─────────────────────────────┼─────────────────────────────┤
│condition                     │4                            │827/4                        │2.3138672203691533           │
└──────────────────────────────┴─────────────────────────────┴─────────────────────────────┴─────────────────────────────┘
```

Figure 10: Represents the TF-IDF table



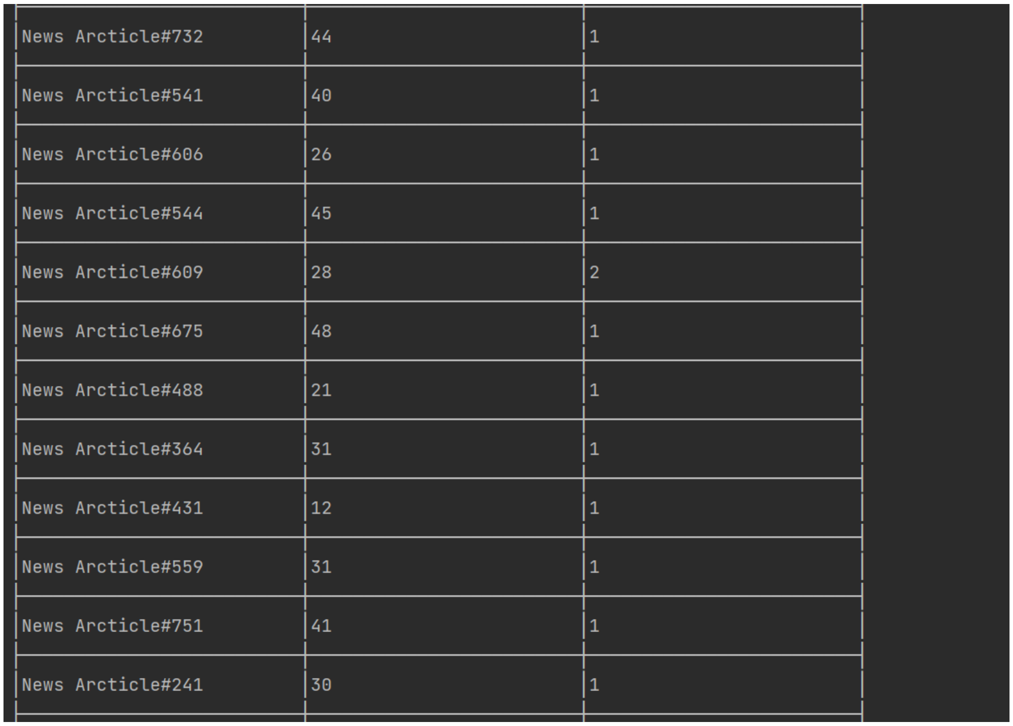
```
┌──────────────────────────────┬─────────────────────────────┐
│News Arcticle#                │highest frequency            │
├──────────────────────────────┼─────────────────────────────┤
│431                           │0.08333333333333333          │
└──────────────────────────────┴─────────────────────────────┘
```

Figure 11: Represents the Highest Relative Frequency



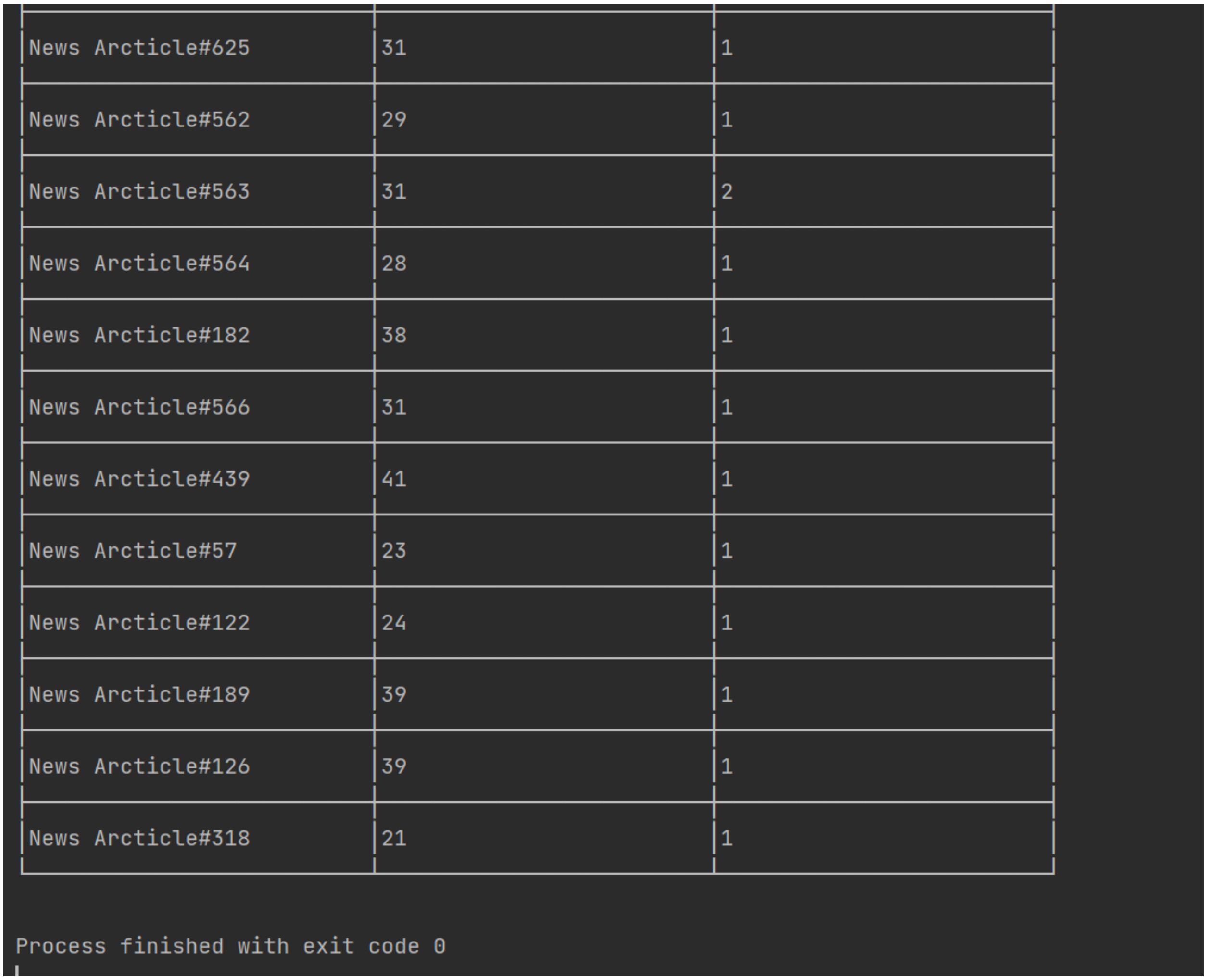
Term	People	
People appeared in 39 documents	Total words (m)	Frequency (f)
News Arcticle#132	22	1
News Arcticle#519	19	1
News Arcticle#136	19	1
News Arcticle#264	31	1
News Arcticle#265	35	1
News Arcticle#74	38	1
News Arcticle#459	31	1
News Arcticle#523	20	1
News Arcticle#462	43	1
News Arcticle#660	46	1
News Arcticle#407	25	1

Figure 12: Represents the Term Frequency Table



News Arcticle#732	44	1
News Arcticle#541	40	1
News Arcticle#606	26	1
News Arcticle#544	45	1
News Arcticle#609	28	2
News Arcticle#675	48	1
News Arcticle#488	21	1
News Arcticle#364	31	1
News Arcticle#431	12	1
News Arcticle#559	31	1
News Arcticle#751	41	1
News Arcticle#241	30	1

Figure 13: Represents the Term Frequency Table (continution)



```
├───────────────────────────┼──────────────────────────┼──────────────────────────┤
│News Arcticle#625          │31                        │1                         │
├───────────────────────────┼──────────────────────────┼──────────────────────────┤
│News Arcticle#562          │29                        │1                         │
├───────────────────────────┼──────────────────────────┼──────────────────────────┤
│News Arcticle#563          │31                        │2                         │
├───────────────────────────┼──────────────────────────┼──────────────────────────┤
│News Arcticle#564          │28                        │1                         │
├───────────────────────────┼──────────────────────────┼──────────────────────────┤
│News Arcticle#182          │38                        │1                         │
├───────────────────────────┼──────────────────────────┼──────────────────────────┤
│News Arcticle#566          │31                        │1                         │
├───────────────────────────┼──────────────────────────┼──────────────────────────┤
│News Arcticle#439          │41                        │1                         │
├───────────────────────────┼──────────────────────────┼──────────────────────────┤
│News Arcticle#57           │23                        │1                         │
├───────────────────────────┼──────────────────────────┼──────────────────────────┤
│News Arcticle#122          │24                        │1                         │
├───────────────────────────┼──────────────────────────┼──────────────────────────┤
│News Arcticle#189          │39                        │1                         │
├───────────────────────────┼──────────────────────────┼──────────────────────────┤
│News Arcticle#126          │39                        │1                         │
├───────────────────────────┼──────────────────────────┼──────────────────────────┤
│News Arcticle#318          │21                        │1                         │
└───────────────────────────┴──────────────────────────┴──────────────────────────┘

Process finished with exit code 0
```

Figure 14: Represents the Term Frequency Table (continution)

References:

[1] S. van der Meer, *README.adoc* at master *vdmeer/asciitable*. [Online]. https://github.com/vdmeer/asciitable/blob/master/README.adoc. [Accessed: 25-Jul-2022].

[2] "Extract multiple fields using MongoDB 3.2.0 java driver," *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/34695546/extract-multiple-fields-using-mongodb-3-2-0-java-driver. [Accessed: 25-Jul-2022].

Git Link:

https://git.cs.dal.ca/umatiya/csci5408_s22_a4_faiza_umatiya_b00899642.git