**ADVANCED COMPUTING CONCEPTS**

**COMP8547 - SECTION 1**

SUBMITTED TO – DR. OLENA SYROTKINA

**TEAM DETAILS – LAB GROUP 3 | BIT BY BIT**

|  |  |
| --- | --- |
| **Student Name** | **Student ID** |
| Yugapriya Shankar | 110090132 |
| Shalini Devendrakumar Shah | 110093386 |
| Kartik Attri | 110091738 |
| Khushi Paul | 110091686 |
| Riteesh kumar Kanamarlapudi | 110090433 |

|  |
| --- |
| **VARIANT**  **The idea of the project is to analyze and understand different credit card offerings and find the best one according to the user's lifestyle.**  **1. Choose a website that offers credit cards (e.g. bmo.com).**  **2. Specify the details of your search (cashback, no annual fee, etc.)**  **3. Find the best offerings in the selected criteria.** |

Source code:[*https://github.com/Yuga16S/credit\_card\_analysis*](https://github.com/Yuga16S/credit_card_analysis)

**The following java source files are checked in by Yugapriya Shankar** 110090132

Default package

MainApplication.java

Helpers

Constants.java

Util.java

Model

CreditCard.java

Services

CreditCardFetcher.java

KeyWordService.java

WebCrawler.java

WebDriverService.java

External

SplayBST.java (Modified and added few methods for my use case to the implementation at <https://algs4.cs.princeton.edu/33balanced/SplayBST.java>)

Running the application in Eclipse IDE:

* Clone the public repository from <https://github.com/Yuga16S/credit_card_analysis>
* Install chrome if your machine doesn’t have it installed already. Check its version and download compatible chromedriver from <https://chromedriver.chromium.org/downloads>
* Copy the executable chromedriver to <path\_to\_project>/resources/bin/ directory
* Update CHROME\_DRIVER\_PATH in <path\_to\_project>/src/helpers/Constants.java file to point to the chromedriver executable from the previous point
* Make sure it has appropriate permissions (chmod 777 chromedriver would give it all permissions)





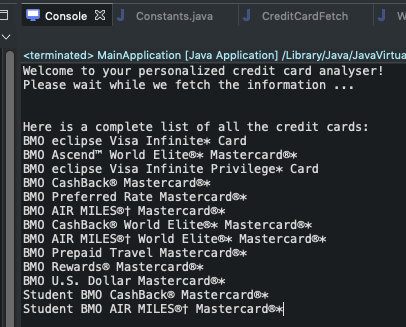
* Run/Debug MainApplication.java in default package

**Features:**

Feature 1: Listing all the credit card names from BMO (HTML Parsing): (Yugapriya Shankar)

* Loads BMO’s all credit card page in a headless browser (Chromedriver – through selenium)
* Scrape all the credit card names with link to its page
* Stores it using the model: *CreditCard.java* which has name and link as it’s member variables
* Uses ArrayList Data structure to store all the credit card objects.

This feature scrapes content from <https://www.bmo.com/main/personal/credit-cards/all-cards/> and displays the credit card names to the console as shown below.



The core functionality of this feature is implemented in CreditCardFetcher.java which in turn uses WebDriverService methods to load the URL and parse its content.

Since most of the content in <https://www.bmo.com/main/personal/credit-cards/all-cards/> is loaded dynamically through javascript on page load, Selenium web driver has been utilized to load the page first and then the HTML page source is extracted from it.

This is illustrated below:

Selenium Web Driver

Chrome Driver

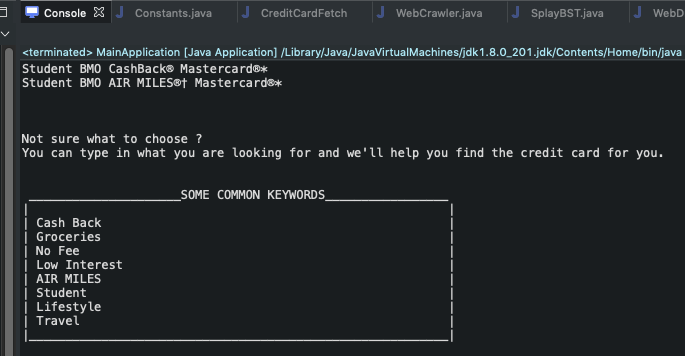
Credit Card Analysis

Chrome Driver Executable

Loads page, executes js in page and downloads page source

*FIg:1 Usage of Selenium web driver in the application*

Additionally, some commonly used keywords are pretty printed on to the console to help the users who aren’t familiar with the credit card features.



Data structures and Algorithms used:

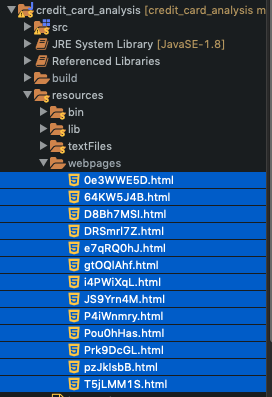
This feature uses ArrayList to store list of credit card objects. Each of these credit card objects has a name and link to its corresponding webpage.

HTML files are parsed using jsoup library which constructs a DOM Tree from the html content. This tree is traversed through Depth first search algorithm to select nodes based on the CSS query.

Feature 2: Web Crawler (Yugapriya Shankar)

* Crawls and downloads all individual credit card pages, and stores them as HTML files
* It downloads pages recursively based on preset *DEPTH* that is configurable.
* Maintains a Hashmap of CreditCard object (key) against all its related filenames (Map<CreditCard, List<String>>)

This feature crawls the credit card pages in bmo.com and downloads all the pages related to them. These pages are stored as html files in webpages directory shown below.



The crawlAndDownload() public method in KeyWordService.java uses the webdriver service as discussed in the previous feature and downloads the page content in a similar fashion. This method returns a map with credit card object as a key and list of filenames as value. It returns a list of filenames because there could be multiple webpages for a given credit card based on the depth configured.

The web crawler’s depth can be configured with values > 1 to enable Deep crawling, if desired

The flow of our recursive algorithm is shown below

Crawled depth= 0

Load URL

Save page content to a file

Crawled depth ++

crawled depth = configured depth ?

Parse the downloaded file

Extract Links

No

Yes

*Fig: 2 Flow chart for web crawling algorithm*

Data structures and Algorithms used:

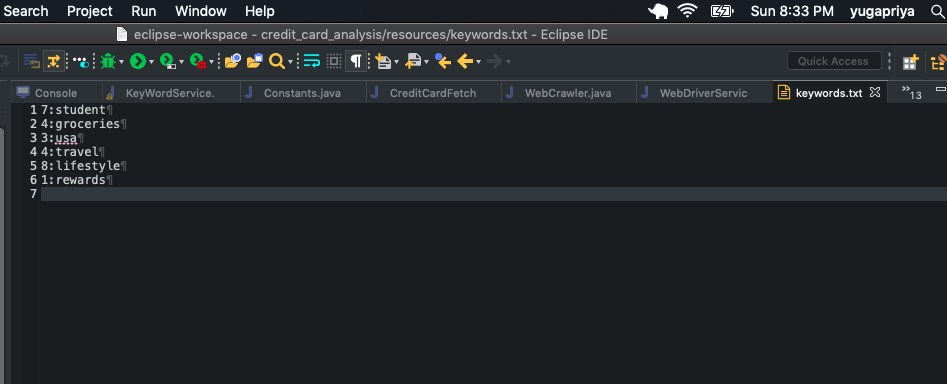
This feature uses a recursive algorithm as shown in the flowchart above. The downloaded HTML pages are parsed for anchor tags when deep crawling is enabled. This uses DFS tree traversal as discussed in the HTML parser feature.

HashMap is used to store an array list of filenames with corresponding Credit card object as a key.

Feature 3: Recently searched word with frequency

* Displays the most recently searched keyword along with its frequency (number of times it has been searched till now)
* Uses a text file (keywords.txt) to persist keywords with its frequency
* Stack Data Structure is used as an intermediate DS to maintain ordering in keywords.txt. Since this file is read from top to bottom, we add the lines read into a stack and then pop them into the Splay BST. This way, the most recently accessed keyword, which is always at the top of the file is also at the root of the Splay tree.
* Splay BST is used to get the most recently accessed keyword.

In this feature, the last searched keyword is shown to the user along with the number of times it has been searched. To persist the search occurrences, a file has been used. Both the keyword and the frequency are delimited through the “:” character and stored in the file in the format below.



These lines in the keywords.txt are read from top to bottom and stored in a stack first. The most recently searched keyword is maintained at the top of the keywords.txt file and is pushed into the stack first. This makes the most recently searched keywords end up in the bottom of the stack. Hence, this is popped last and inserted into the splay tree last making it the last accessed node in splay tree. This operation is implemented in the getKeywords() method in KeyWordService.java and is illustrated below

11:groceries

---------------

----------------

Keywords.txt

Stack

11:groceries

Most recently searched

--

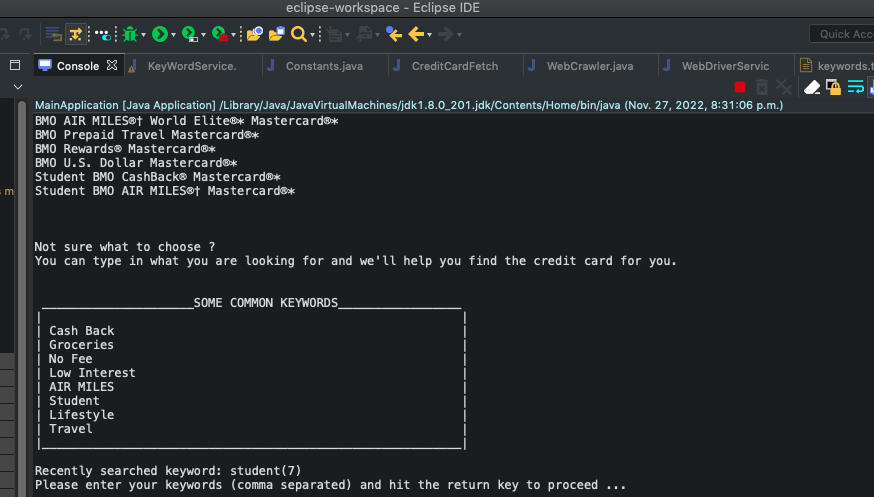
--

Most recently searched

SplayBST

*Fig: 3 SplayBST utilization in getLastSearchedKeywordWithFrequency method*

The root node from this tree is read by the getLastSearchedKeywordWithFrequency () method in KeyWordService.java and displayed as shown below



The keywords searched by the user are looked up in the SplayTree first and its count is incremented by 1. Since we are inserting/updating a node in the splay tree, this is made as the new root node through splaying operation. The root node is then removed from the splay tree and written back to the file in overwrite mode. This way the contents of the keywords.txt are updated for each user search.

Note: The root node is removed from the splay tree before writing to the text file because the getRootKey() function of the SplayTree always returns the root node which would remain the same if it is not removed.

Time complexity for Splay Tree operations:

Worst Case: O(n)

Average: O(log n)

Time complexity of HashMap insertion and retrieval: O(1)

- End of features developed by Yugapriya Shankar -

The following java source files are checked in by Riteesh Kumar Kanamarlapudi 110090433

**Model**

Posting.java

**Services**

CreateInvertedIndex.java

CreateTextFiles.java

HtmlToText.java

Features:

Feature 1 : Finds the file name(s) of each keyword in the scraped files from banking site (Inverted Indexing)

* Iterates through and checks all the files that are scraped from the banking site for the keyword searched.
* When match is found, returns the file name(s) that contains the given keyword.
* Uses Hash map to return output which shows the file name(s) and keyword searched.

Feature 2: Finds the frequency of each keyword in the scraped files from banking site (Frequency count)

* The frequency of each word or key word of the files is counted and it is stored in the Hash map i.e. Inverted index map as in a ArrayList as value of the hash map along with the keyword.

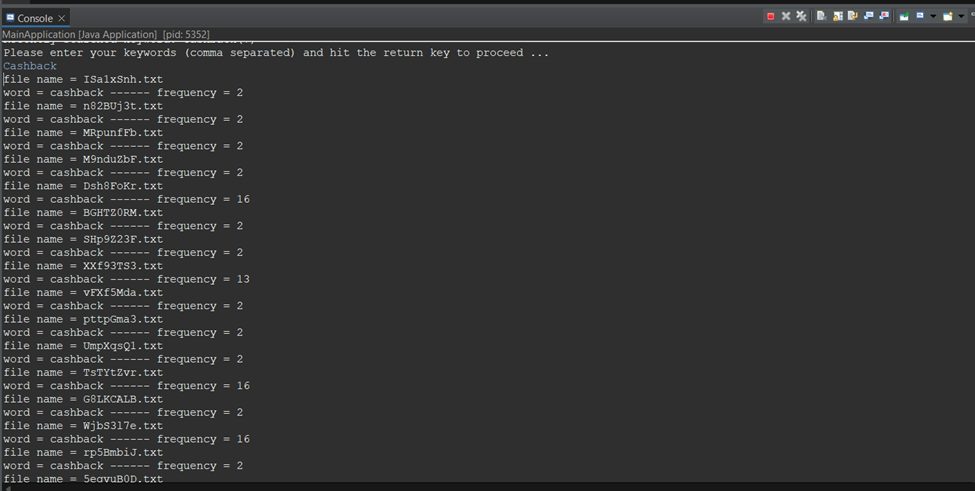
**Data Structures/Algorithms used for implementing these features**

To implement these features i.e. Inverted indexing and frequency count the data structures mainly used Hash map, ArrayList and Linked list. The main advantage of hash map is that the time complexity for inserting and finding a word is O(1), which is important for inverted index to achieve quick results. The advantage of linked list is it takes O(1) time complexity for appending data (i.e. adding data to the end) and also used ArrayList to store the frequency of words which is used to create Inverted index map, the benefit of using ArrayList over array is it is dynamic that it can change its size based on the input and as well we can save memory as it doesn’t need to initialize the size of it during declaration which happens in case of using array. In the project, the hash map.

HashMap<String, ArrayList<WordFrequency>> finalFrequencyMap = new HashMap<String, ArrayList<WordFrequency>>();

which uses file name as key i.e. of string data type, and array list of WordFrequency object which contains keyword and its frequency count in it.

**Output**



The above screenshot depicts the inverted indexing and frequency count features, it is seen in the screenshot when a word is given an input to the invert index it will display the corresponding files which contains that word and respective frequency of that word in each file that it is present.

The following java source files are checked in by Shalini DevendraKumar Shah- 110093386

**Default Package:**

MainApplication.java

**Services:**

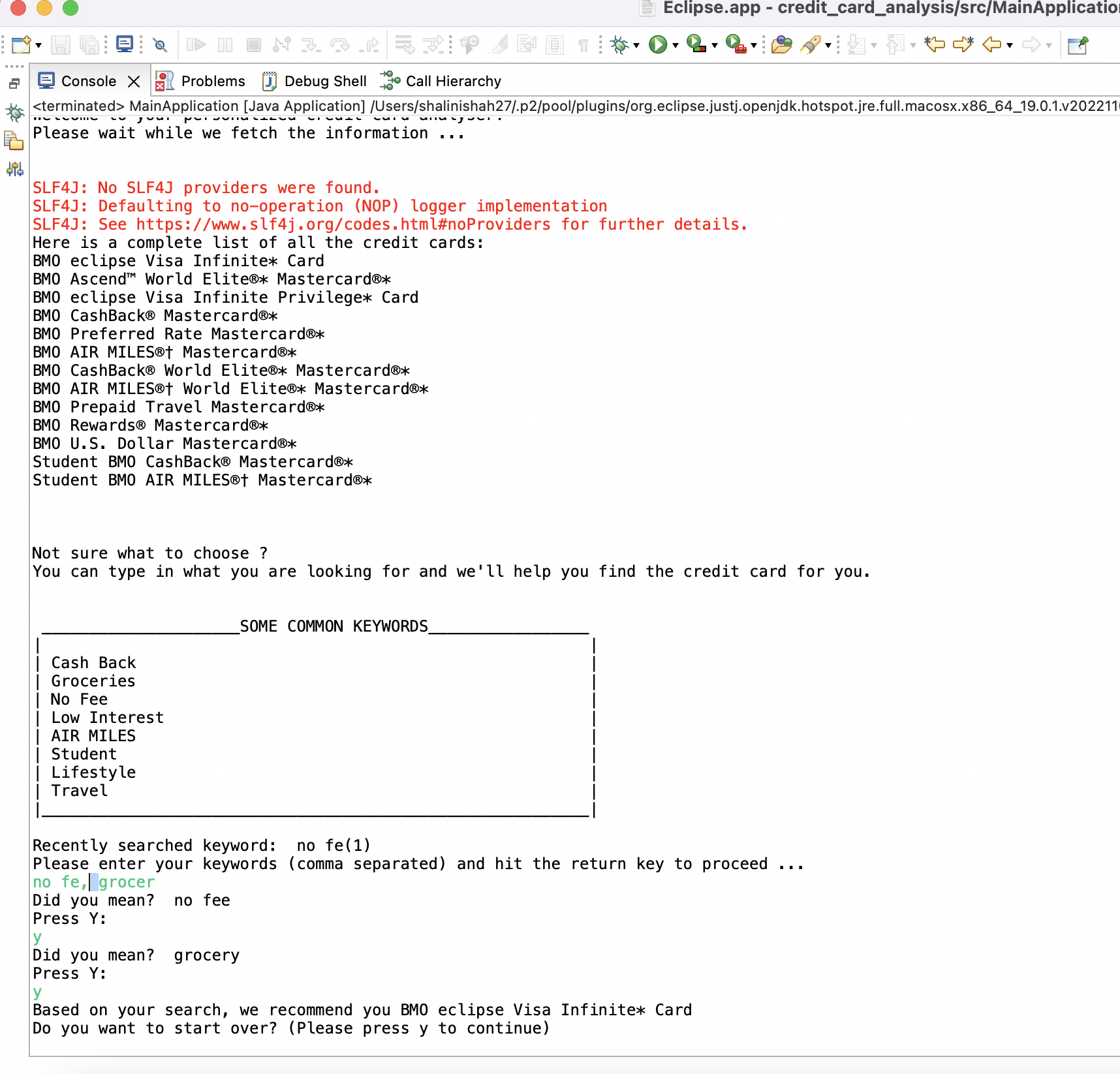
Searchword.java

# **Feature1: Input User Validation:**

* The system will take the words from the user separated by a comma. And the system will scan the input word and validate the word using regex and it will check whether the input is valid or not.
* If the input is valid, it will check for an alternate word and suggest the card.
* If the user enters any number between the words, it will count as invalid input and it will show the output as “Search keywords cannot have numbers in it”.

**Search Word:**

* In this, HashMap is used as a data structure. It will map the keys to the value. An array of buckets is used in a hash table to provide an index, also known as a hash code, from which the requested data can be retrieved. The key is hashed during lookup, and the resulting hash shows where the relevant value is kept.
* HashMap<String, Integer> numbers = **new** HashMap<String, Integer> ();
* It will search for similar word in file and if the word is incomplete, it will suggest the alternate similar word.
* By using edit distance, the system will differentiate two strings from one another are determined by counting the least number of operations necessary to change one string into the other.
* After comparing the input word (no fe) with the files, the system will suggest the similar words as: “Do you mean: no fee” to the user.



**References:**

1. <https://jsoup.org/cookbook/>

2. <https://github.com/jhy/jsoup>

3. <https://algs4.cs.princeton.edu/33balanced/SplayBST.java.html>

4. <https://www.selenium.dev/documentation/webdriver/>

5. <https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html>

6. <https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html>

7. <https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>

8. <https://docs.oracle.com/javase/8/docs/api/java/io/FileWriter.html>

9. <http://en.wikipedia.org/wiki/Inverted_index>

10. <https://dev.to/im_bhatman/components-of-inverted-index-the-dictionary-1gf5>

11. <https://www.geeksforgeeks.org/inverted-index/>

**The following java source files are checked in by KARTIK ATTRI (110091738)**

Default package

MainApplication.java

Services

CreditCardRanking.java

CreateInvertedIndex.java

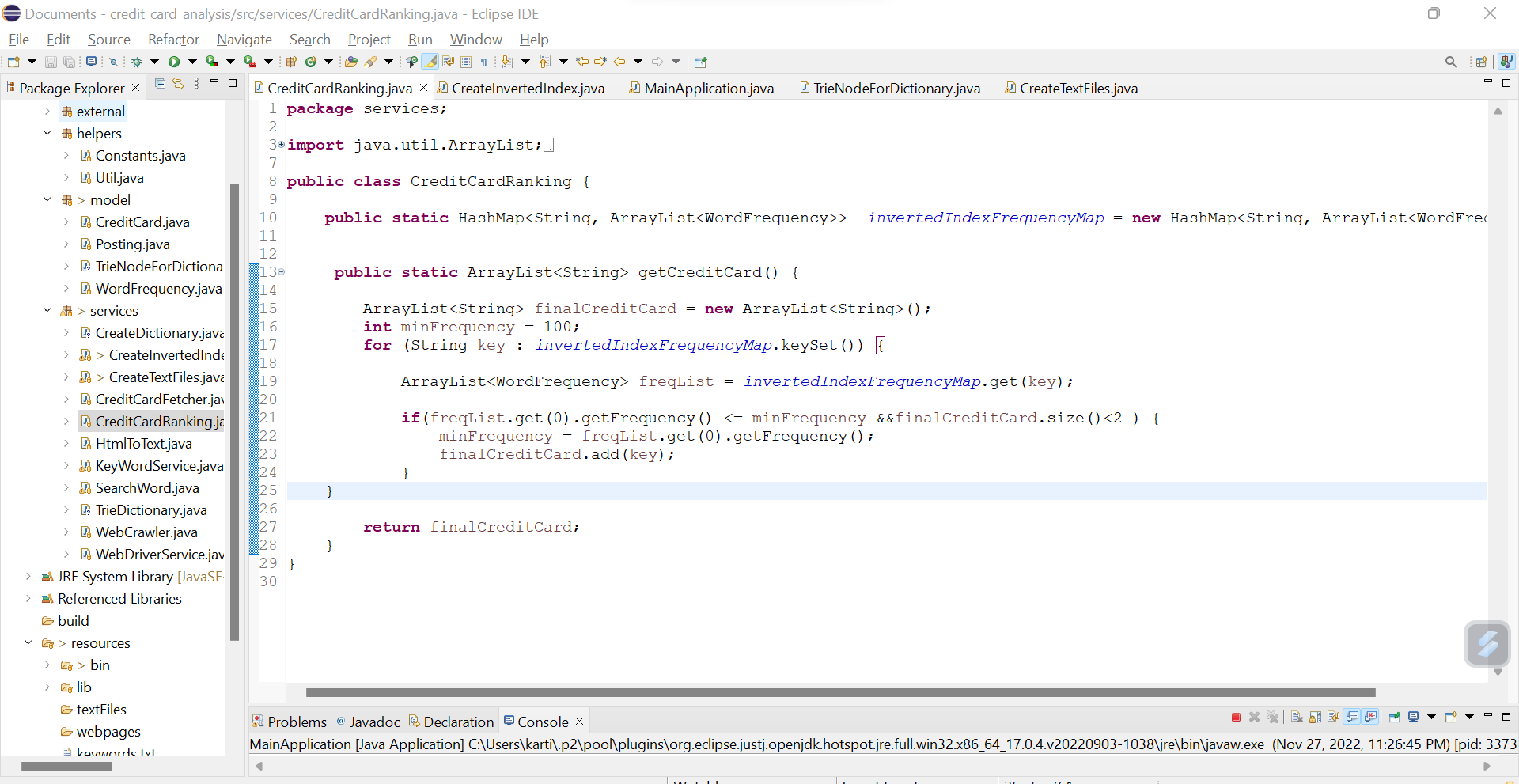
CreateDictionary.java

Integration of all features

**Feature 1: Page Ranking - CreditCardRanking.java (Kartik Attri)**

* This algorithm will work on the inverted-index map created.
* Frequency of keyword entered by user, is counted in each file and is stored in inverted index map.
* Then frequency in each file is compared to each other and then finally the best credit card is displayed as output to user.
* Hashmap is defined as :-

public static HashMap<String, ArrayList<WordFrequency>> invertedIndexFrequencyMap = new HashMap<String, ArrayList<WordFrequency>>();



**Data structures used**

* **HashMap** - to store frequency of keyword with filename in hashMap
* **ArrayList**- to store best credit card details and display them to user

**Complexity**

* **HashMap** - O(N) for traversing array-list

O(1) for inserting into hash-map

O(1) for loop up in hash-map

* **ArrayList** - O(N) for traversing array-list

O(1) for inserting into array-list

**Why Hash-Map and ArrayList Data structure?**

* Hash-map is used at it gives best finding and insertion complexities.
* Array-list because it can increase its size dynamically as compared to arrays where we must specify the size of array before.

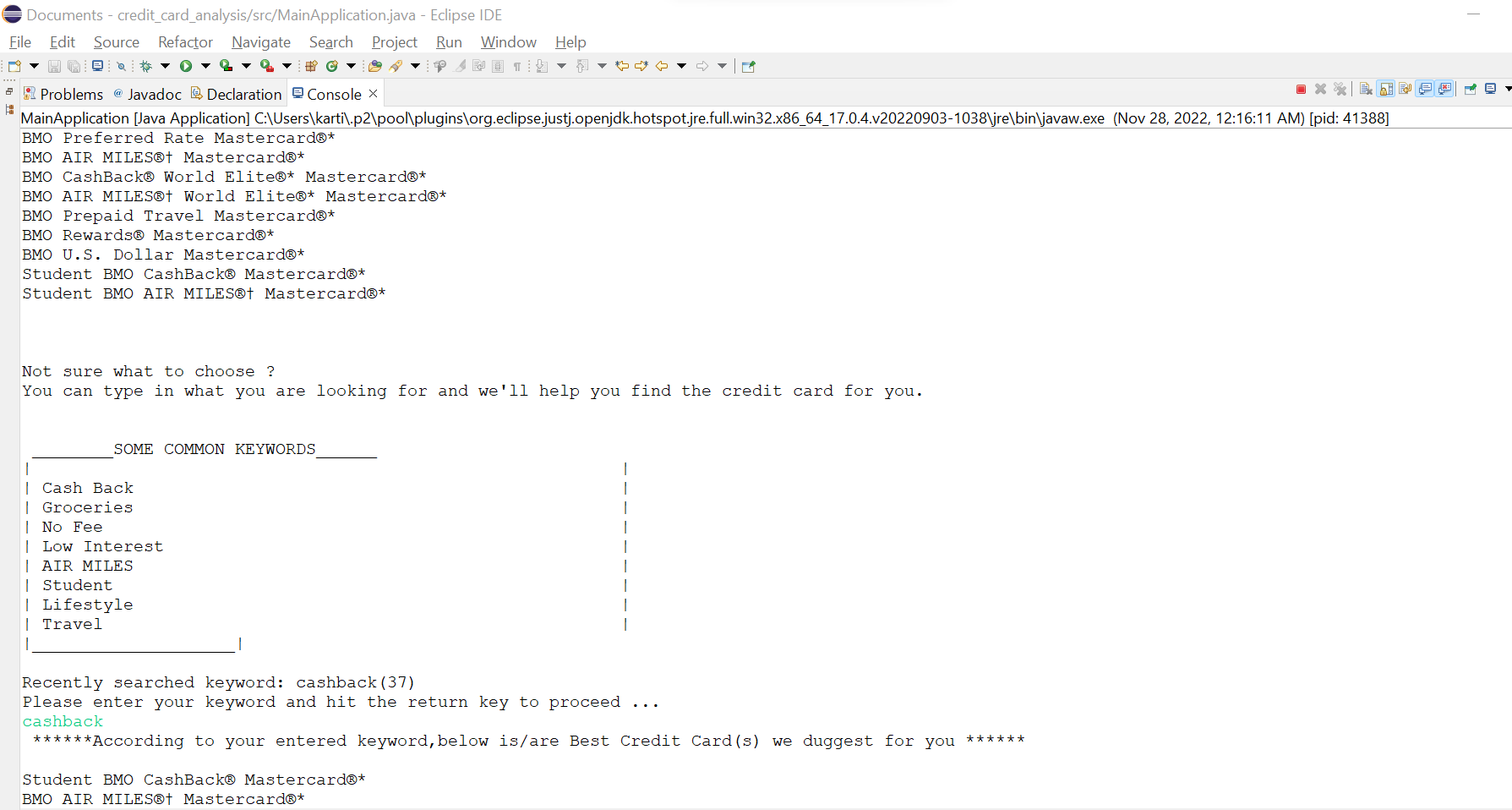
**Feature 2: InvertedIndex freqeuncy map - CreateInvertedIndex.java (Kartik Attri)**

* Contributed my team member in creating invert index map function
* **Hash map** is used to create invertedIndex map.

**Feature 3: SpellChecker - CreateInvertedIndex.java (Kartik Attri)**

* Contributed my team member in creating dictionary by reading text files.
* Using **StringTokenizer** to generate tokens and store them in dictionary.
* For creating dictionary, **trie data structure** is used

**Final output:-**



**References**

* All lectures given by Professor. Olena Syrotika during Fall 2022 term for Advanced Computing Concepts subject.

The following java source files are checked in by **Khushi Paul 110091686**

**Default Package:**

MainApplication.java

**Services:**

CreateDictionary.java

TrieDictionary.java

**Model:**

TrieNodeForDictionary.java

**Data Structures Used:**

1. **Trie:** With the help of trie added and retrieived words in the dictionary. As its time complexity is optimal than other data structures.

**Complexities:**

**Function. Complexity**

Insertion. O(n)

Searching O(n)

Deletion. O(n)

\*\* Where ‘n’ is the length of the single word.

**Features:**

Faster than BST(Binary search tree)

Faster than hashing as we dont need to make hash functions as well as no collision is there.

2. **ArrayList**: Used array list to store the words, strings and list of files

**Features:**

1. We can access the list at random because using Java ArrayList.
2. Primitive types like int, char, and other kinds cannot be utilised with array lists. For such circumstances, we require a wrapper class.

**Features Implementation**

1. **Creating Dictionary/SpellCheck:**
2. Word dictionary was created using converted.txt files in order to verify that the user-entered keywords are included in the files and display the credit cards in accordance with their keyword requirements.
3. Trie was employed to store and retrieve words in the project.
4. **Testing:**

When all of the code is combined into one file, testing is done to ensure that every feature functions as intended.

**ScreenShot:**

