Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

FATİH DURAL
151044041

Course Assistant: Ayşe Şerbetçi Turan

1 INTRODUCTION

1.1 Problem Definition

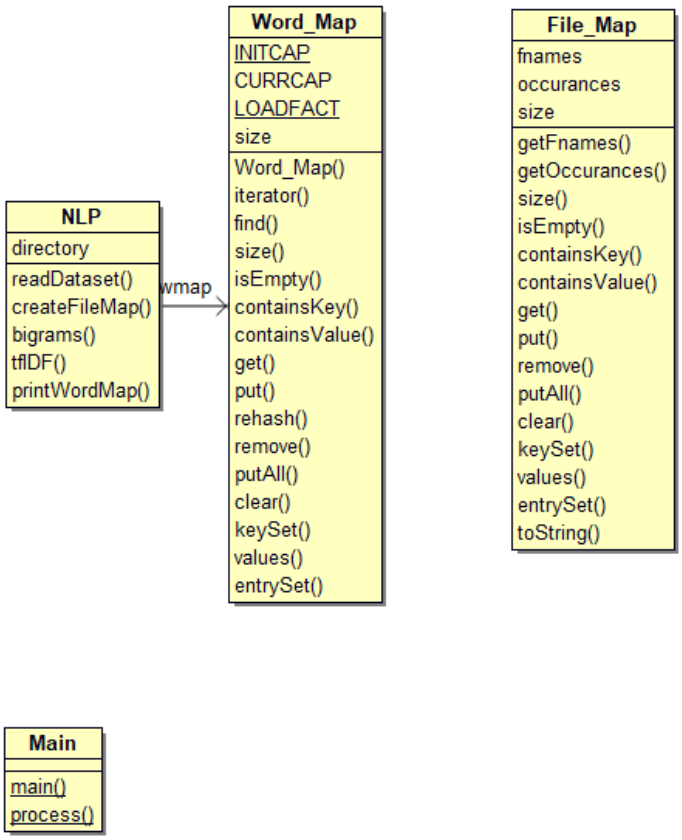
In this homework, I will implement two different HashMap classes to perform basic Natural Language Processing operations. I will read a text dataset consisting of multiple input files and keep the words in the Word HashMap. The key for the word hashmap will be the words and the value will refer to another hashmap (file hashmap) which keeps the occurrences of the word in different files. The key for the file hashmap is the filename and the value is an arraylist containing the word positions in that file. In order to be able to traverse the word map in an efficient way, each entry should keep a pointer to the next inserted entry, allowing the structure to be Iterable (implements Iterable interface). By this way, the methods such as containsValue(), containsKey() and keySet() can be implemented in a more efficient way, without the need for visiting empty cells in the table. After obtaining this structure, I will implement two basic operations used in NLP : retrieving bi-grams and calculating TFIDF values.

1.2 System Requirements

There will be 3 different classes in this assignment. The first one is the Word Map class, the second is the File Map class, and finally the NLP class, in addition to the driver class - main. Various methods in the NLP class will use other classes. A robust implementation is needed. I need a node class in Word Map. Node key, value and next to keep the linked structure will be established. I also need iterator to navigate. I will write all methods their own class. I will test using main class and i am going to write the program using intellij.

2 METHOD

2.1 Class Diagrams



2.2 Use Case Diagrams

Add use case diagrams if required.

2.3 Other Diagrams (optional)

Add other diagrams if required.

2.4 Problem Solution Approach

At first, I started writing a word map class. Node array exists, each element in this array holds a Node object. In the Node class I have the key, value, next. When the word is put in the map, the find function has an index and the element is placed in that index. It works at (1) time. Linked structure is established. When the memory is full, the rehash method is called up and continues. Written in other methods and the Word Map class is

now complete. Word Map class's key a string and value is a File Map. The file map also uses the integer array as the key, the file name and the value as the value. I used the ArrayList, which is a ready-made Java class that I don't take part in when I put it in File Map. In the NLP class, we can use other classes. The readDataset read directory and the Word Map object is created. Biagrams and TFIDF values are available. There is also a print method in Word Map. The time complexity varies between O (1) and O (n).

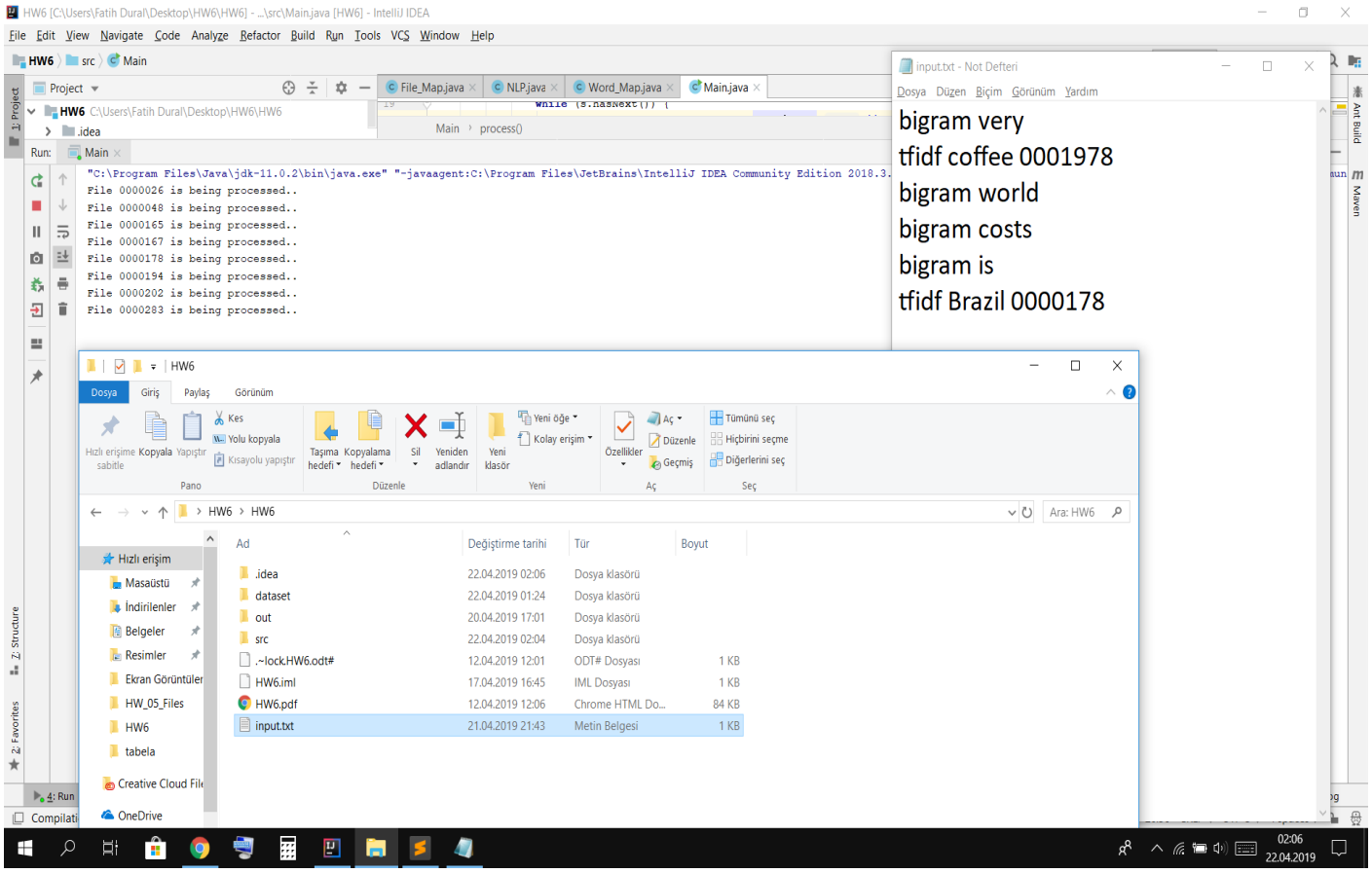
3 RESULT

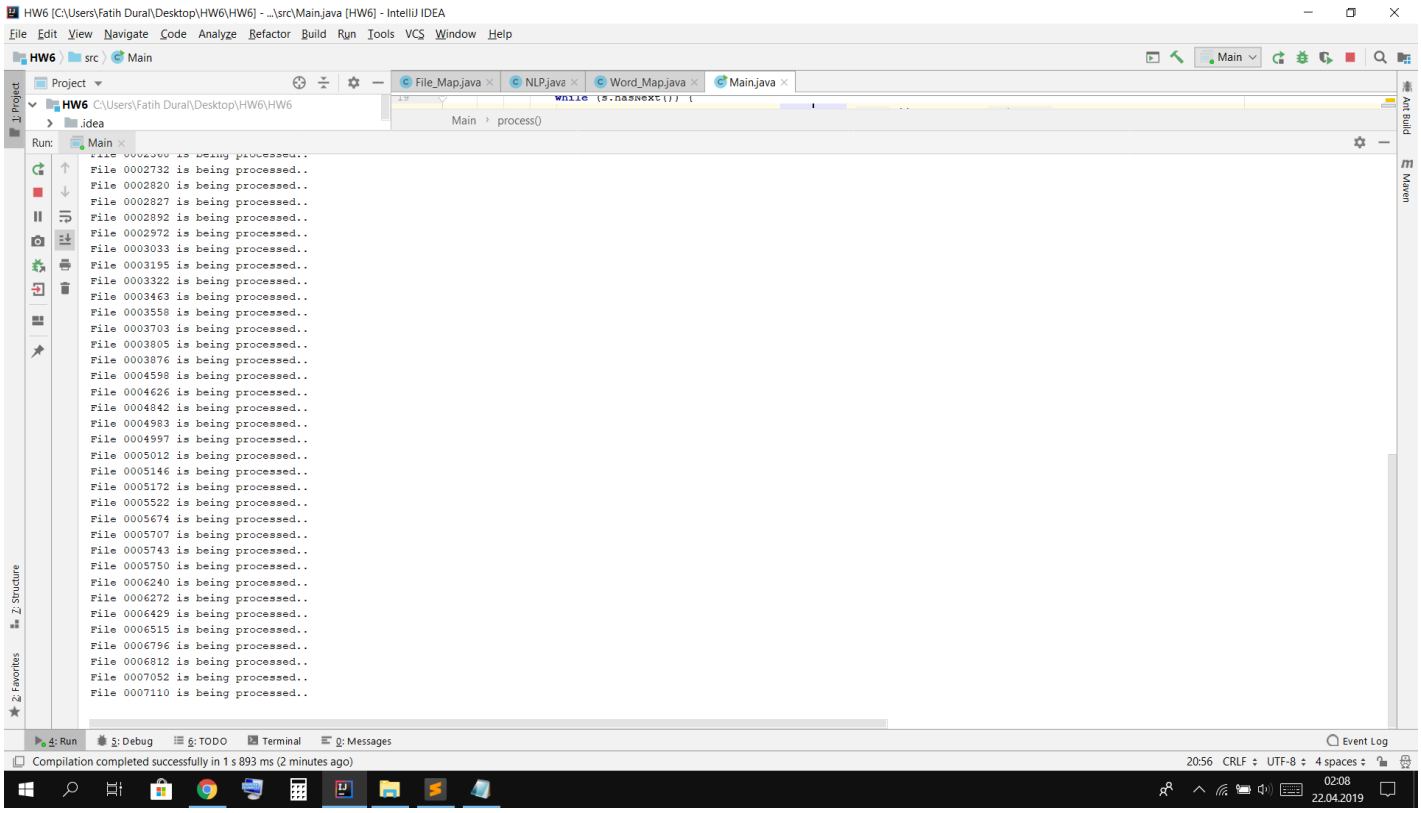
3.1 Test Cases

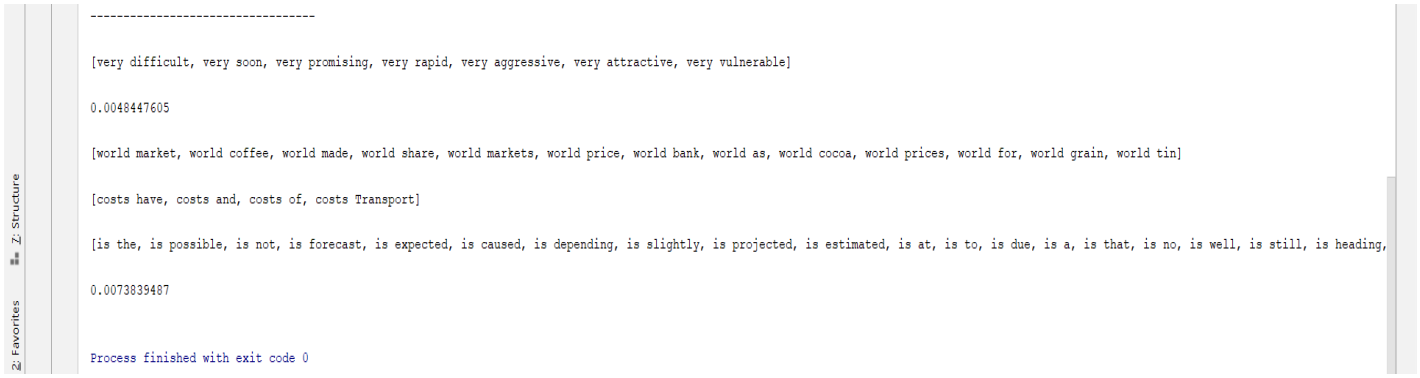
```
public class Main {
    public static void main(String[] args) {
        String dir = "dataset/";        // dataset - src - input.txt are same priority directory. If you can, change it.
        String instructionsFile = "input.txt";  // if you change input.txt location, please give correct path.
        NLP natLanProc = new NLP();     // create NLP object.
        natLanProc.readDataset(dir);   // read data set and create Word Map.
        //nlp.printWordMap();           // if you want, you can print Word Map.
        process(natLanProc, dir, instructionsFile);        // homework job done.
    }

    private static void process( NLP nlp, String directory, String instructionsFile ){
        try {
            Scanner sc2 = new Scanner(new File( instructionsFile ));
            while( sc2.hasNextLine() ){
                Scanner s = new Scanner(sc2.nextLine());
                while (s.hasNext()) {
                    String word = s.next().trim().replaceAll( regex: "\\p{Punct}",  replacement: "");
                    if( word.equals("tfidf") ){ // if word is tfidf
                        word = s.next().trim().replaceAll( regex: "\\p{Punct}",  replacement: "");     // corrected
                        String fileName = s.next().trim().replaceAll( regex: "\\p{Punct}",  replacement: ""); // corrected
                        float TFIDF = nlp.tfIDF(word, fileName);        // calculate TFIDF = TF * IDF.
                        System.out.println(TFIDF);
                    }
                    else if( word.equals("bigram") ){   // if word is biagram
                        word = s.next().trim().replaceAll( regex: "\\p{Punct}",  replacement: "");     // corrected
                        ArrayList<String> bigram = (ArrayList<String>) nlp.bigrams(word);   // calculate biagram
                        System.out.print("[");      // print standart output.
                        for( int i = 0; i < bigram.size()-1; i++){
                            System.out.print(bigram.get(i) + ", ");
                        }
                        System.out.print(bigram.get(bigram.size() - 1));
                        System.out.print("]\n");
                    }
                    System.out.println();
                }
            }
        }
        catch (Exception e){
```

3.2 Running Results







- Main titles -> 16pt , 2 line break
- Subtitles -> 14pt, 1.5 line break
- Paragraph -> 12pt, 1.5 line break