Gebze Technical University
Computer Engineering

CSE 222 - 2019 Spring

HOMEWORK 6 REPORT

Taylan ÖNDER
151044015

Course Assistant: Ayşe Şerbetçi TURAN

1 INTRODUCTION

1.1 Problem Definition

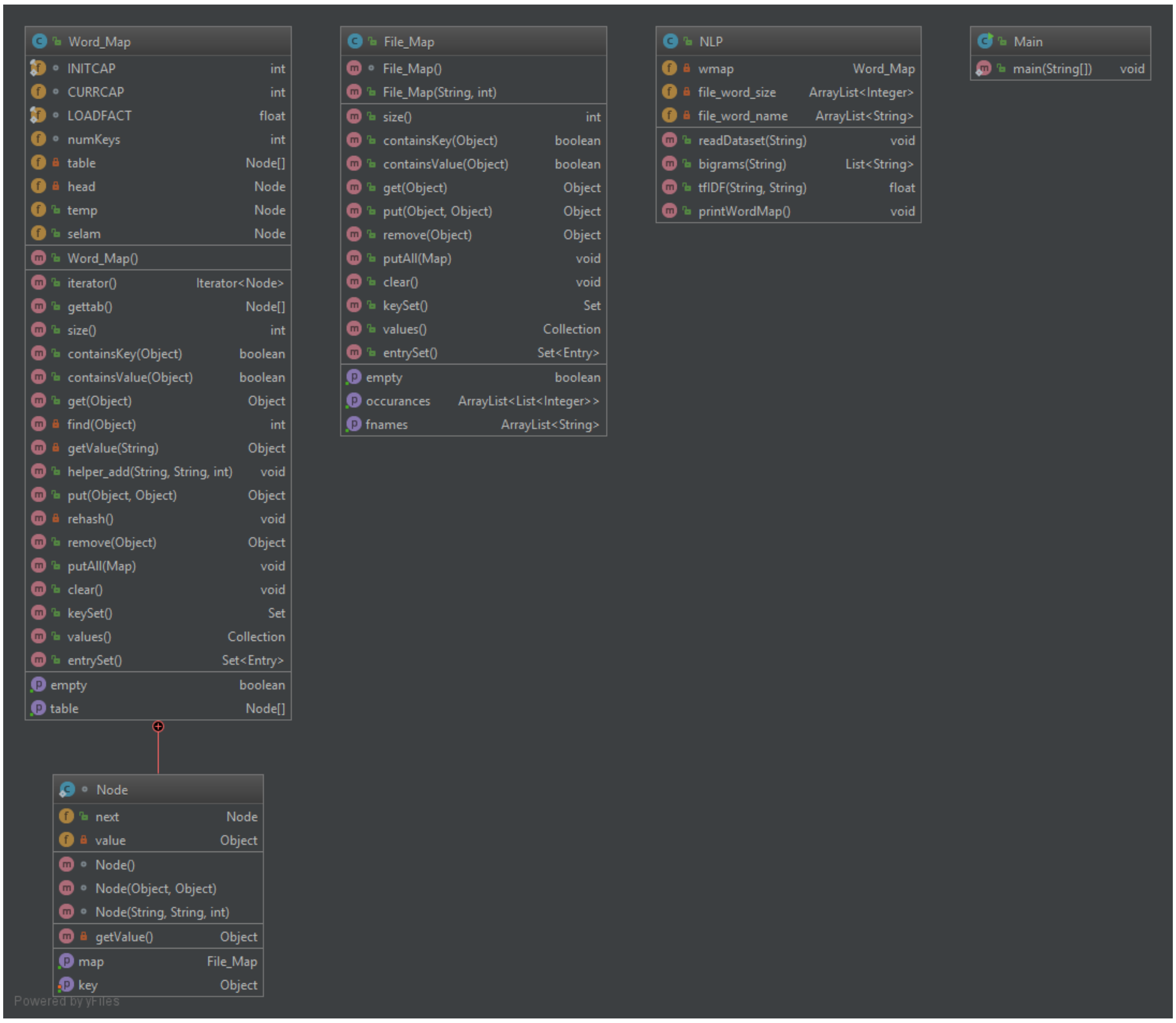
In this homework we will write two hashmap classes to perform basic Natural Language Processing operations. More than one file to read the operation of the file will stored by 2 different hashtable.This hashtables should store words, file names and word locations where in file. Firstly Word Hashmap keep words as a key and FileHashmap objects as a value. Also in Word Hashmap all keys or words linked as a node. For efficiency linked will be used instead of turn index of table by one by. Secondly the key for the file hashmap is the filename and the value is an arraylist containing the word positions in that file.Iterable interface will be implement to accessing hashtables keys and values with iterator. NLP retrieving bi-grams and calculating TFIDF values, which are explained below, respectively. NLP class should read files and implement two important method to use basic Natural Language Processing operations. First of all bigram should put given word and neighbour of this World. We can explain Bi-grams as A bi-gram is simply a piece of text consisting of two sequential words which occurs in a given text at least once. Bigram method return a list of string all bigrams of tablemap. Second important method is tfIDF. tfIDF is term frequency-inverse document frequency. This is a score which reflects the importance of a word for a single document. In NLP, a word is informative for a file to be categorized if it occurs frequently in that file but has very few occurrence in other documents in the dataset. After mathematical calculation according to given formula in given documantain print value on screen and return value. After the program is run and file readings are completed, you can find the relationship with other words within the files that are read for each desired word

1.2 System Requirements

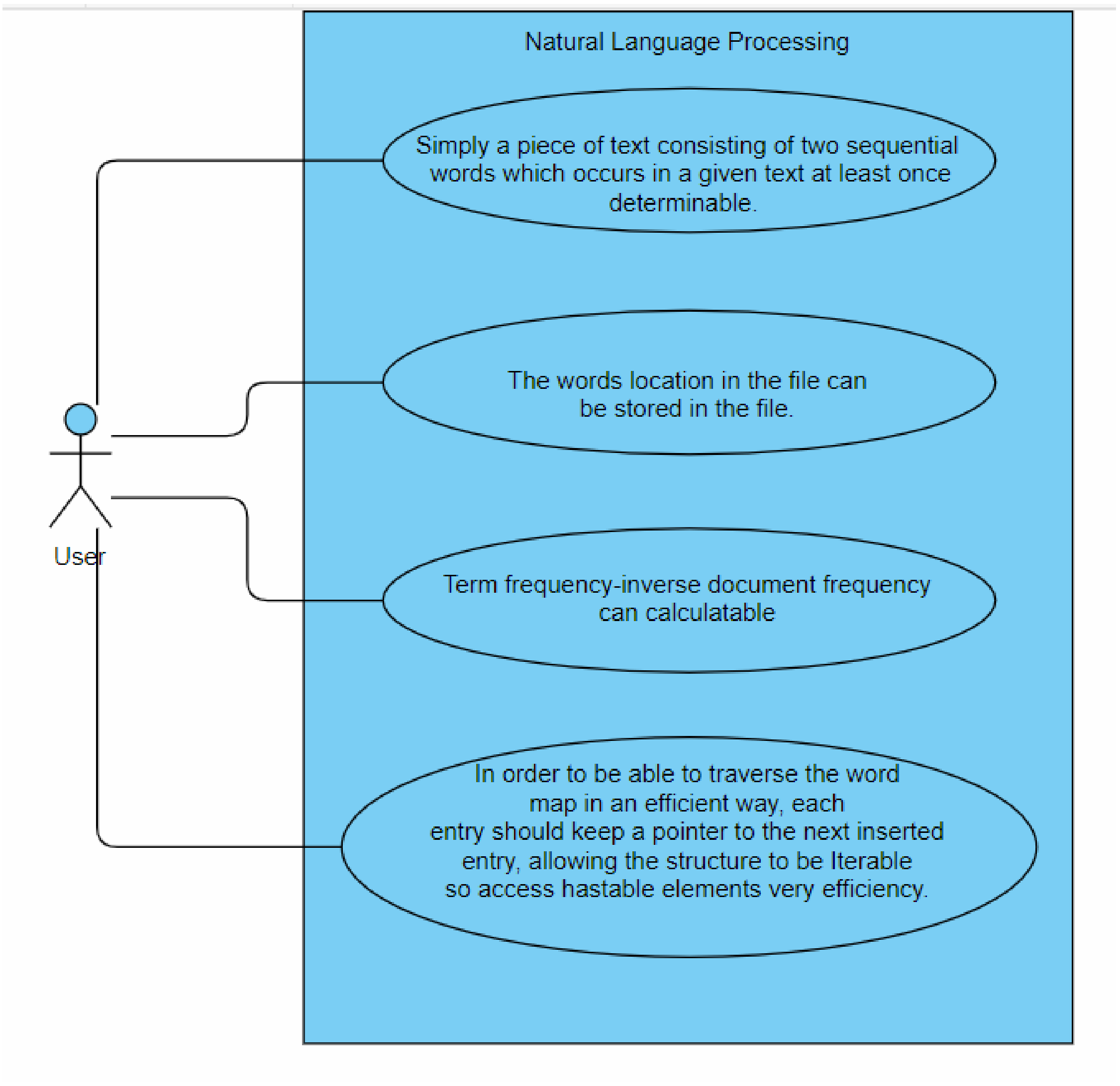
We need WordMap class to store words as a node.So WordMap class also should has inner static node class.Using WordMap class object, value and keys should store in node.Wordmap of key is a FileMap object so FilmMap object also has to create.FileMap object keep filenames as a key and , occurences in that file with a list as a value.So arraylist should be defined as a package also List,String and File packages are should be defined. The nfl object must be created in the main to read the words in all files and to transfer them to the hash. NFL class does not know FileMap objects so, FileMap object should be create in WordMap so In NFL class we send a arraylist as a value in WordMap class constructure and store datas in NFL readdata method.

2 METHOD

2.1 Class Diagrams

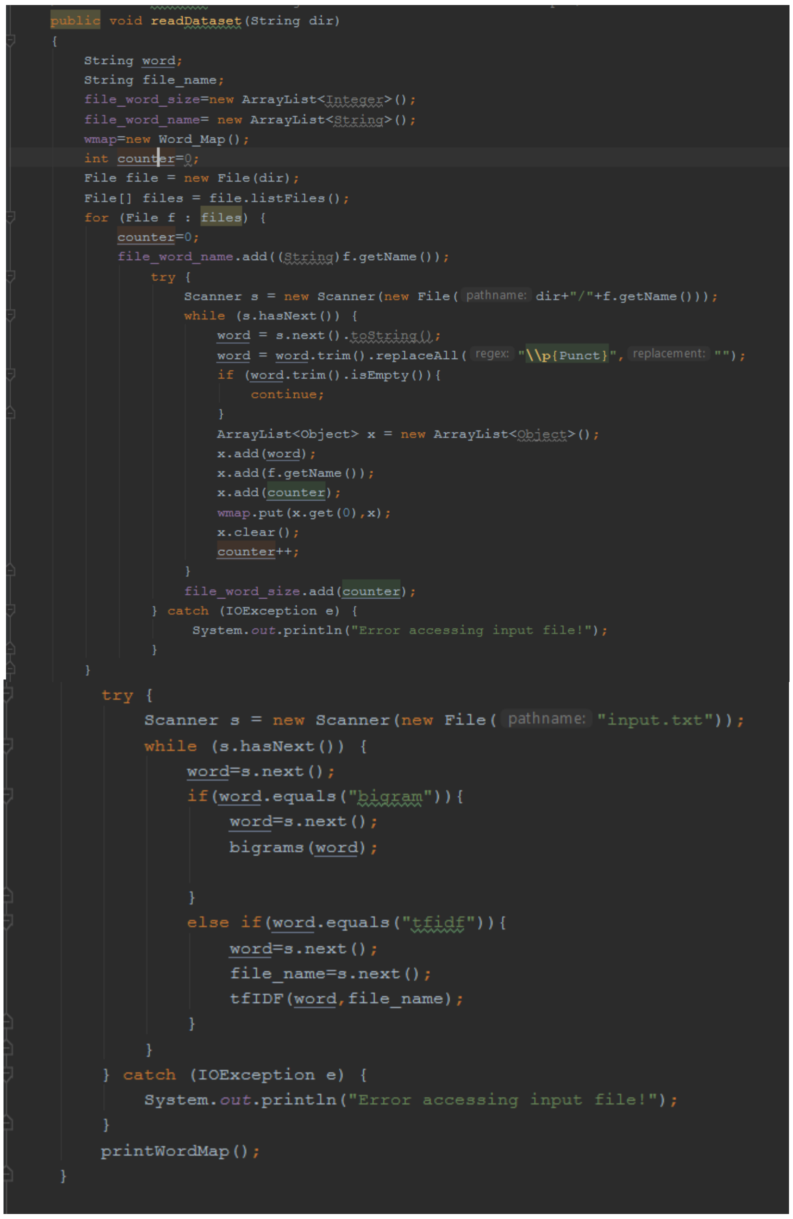


2.2 Use Case Diagram

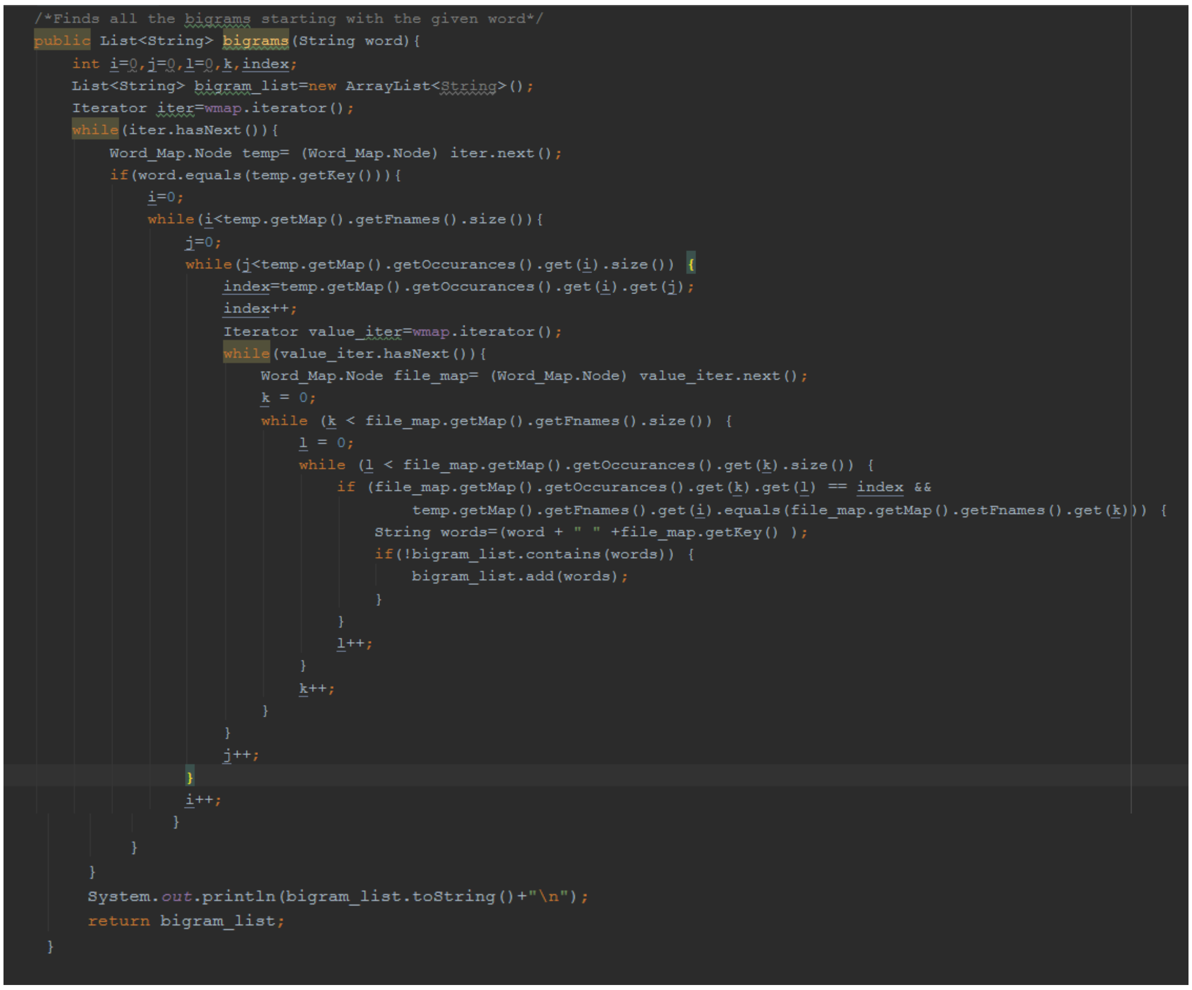

Natural Language Processing
Simply a piece of text consisting of two sequential words which occurs in a given text at least once determinable.
The words location in the file can be stored in the file.
Term frequency-inverse document frequency can calculatable
In order to be able to traverse the word map in an efficient way, each entry should keep a pointer to the next inserted entry, allowing the structure to be Iterable so access hastable elements very efficiency.
User

2.2 Problem Solution Approach

First of all given direcotry in main all files should be read and all words sould be put in wmap World map object of NFL. In NFL class Since we couldn't reach the file map class, I created a local array list and gave it like FileMap object.Also in there I keep total words number of the file which reading in file_word_size arraylist with filenames in file_word_name arraylist to using tfidf method.After given directory reading, I read input.txt and according to words and I run the bigram or tfidf method.

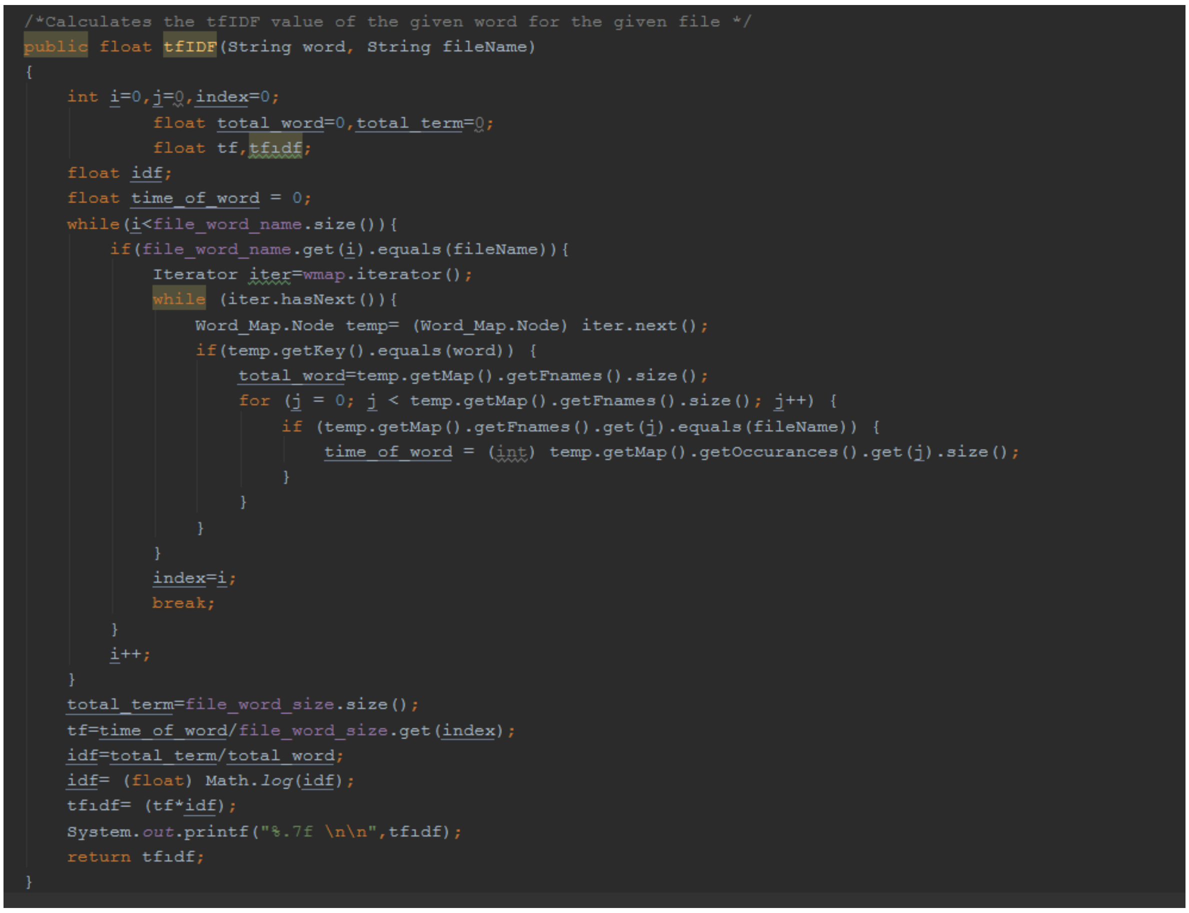


```java
public void readDataset(String dir)
{
    String word;
    String file_name;
    file_word_size=new ArrayList<Integer>();
    file_word_name= new ArrayList<String>();
    wmap=new Word_Map();
    int counter=0;
    File file = new File(dir);
    File[] files = file.listFiles();
    for (File f : files) {
        counter=0;
        file_word_name.add((String)f.getName());
            try {
                Scanner s = new Scanner(new File( dir+"/"+f.getName()));
                while (s.hasNext()) {
                    word = s.next().toString();
                    word = word.trim().replaceAll( "\\p{Punct}", "");
                    if (word.trim().isEmpty()){
                        continue;
                    }
                    ArrayList<Object> x = new ArrayList<Object>();
                    x.add(word);
                    x.add(f.getName());
                    x.add(counter);
                    wmap.put(x.get(0),x);
                    x.clear();
                    counter++;
                }
                file_word_size.add(counter);
            } catch (IOException e) {
                 System.out.println("Error accessing input file!");
            }
    }
        try {
            Scanner s = new Scanner(new File( "input.txt"));
            while (s.hasNext()) {
                word=s.next();
                if(word.equals("bigram")){
                    word=s.next();
                    bigrams(word);

                }
                else if(word.equals("tfidf")){
                    word=s.next();
                    file_name=s.next();
                    tfIDF(word,file_name);
                }
            }
        } catch (IOException e) {
            System.out.println("Error accessing input file!");
        }
        printWordMap();
    }
```



```java
/*Finds all the bigrams starting with the given word*/
public List<String> bigrams(String word){
    int i=0,j=0,l=0,k,index;
    List<String> bigram_list=new ArrayList<String>();
    Iterator iter=wmap.iterator();
    while(iter.hasNext()){
        Word_Map.Node temp= (Word_Map.Node) iter.next();
        if(word.equals(temp.getKey())){
            i=0;
            while(i<temp.getMap().getFnames().size()){
                j=0;
                while(j<temp.getMap().getOccurances().get(i).size()) {
                    index=temp.getMap().getOccurances().get(i).get(j);
                    index++;
                    Iterator value_iter=wmap.iterator();
                    while(value_iter.hasNext()){
                        Word_Map.Node file_map= (Word_Map.Node) value_iter.next();
                        k = 0;
                        while (k < file_map.getMap().getFnames().size()) {
                            l = 0;
                            while (l < file_map.getMap().getOccurances().get(k).size()) {
                                if (file_map.getMap().getOccurances().get(k).get(l) == index &&
                                        temp.getMap().getFnames().get(i).equals(file_map.getMap().getFnames().get(k))) {
                                    String words=(word + " " +file_map.getKey() );
                                    if(!bigram_list.contains(words)) {
                                        bigram_list.add(words);
                                    }
                                }
                                l++;
                            }
                            k++;
                        }
                    }
                    j++;
                }
                i++;
            }
        }
    }
    System.out.println(bigram_list.toString()+"\n");
    return bigram_list;
}
```

Bi-grams: A bi-gram is simply a piece of text consisting of two sequential words which occurs in a given text at leastonce.Check FileMap table with iterator using nodes. According to given word find occurences of this word and check occurences index+1 in Node table. If find index+1 in another filemap occurences in filemap cat word and finding key.After cating words the creating string is pressed and added to the list to be returned.



```java
/*Calculates the tfIDF value of the given word for the given file */
public float tfIDF(String word, String fileName)
{
    int i=0,j=0,index=0;
            float total_word=0,total_term=0;
            float tf,tfıdf;
    float idf;
    float time_of_word = 0;
    while(i<file_word_name.size()){
        if(file_word_name.get(i).equals(fileName)){
            Iterator iter=wmap.iterator();
            while (iter.hasNext()){
                Word_Map.Node temp= (Word_Map.Node) iter.next();
                if(temp.getKey().equals(word)) {
                    total_word=temp.getMap().getFnames().size();
                    for (j = 0; j < temp.getMap().getFnames().size(); j++) {
                        if (temp.getMap().getFnames().get(j).equals(fileName)) {
                            time_of_word = (int) temp.getMap().getOccurances().get(j).size();
                        }
                    }
                }
            }
            index=i;
            break;
        }
        i++;
    }
    total_term=file_word_size.size();
    tf=time_of_word/file_word_size.get(index);
    idf=total_term/total_word;
    idf= (float) Math.log(idf);
    tfıdf= (tf*idf);
    System.out.printf("%.7f \n\n",tfıdf);
    return tfıdf;
}
```

In this code calculate with iteration Number of times term t appears in a document and Number of documents with term t in it.

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

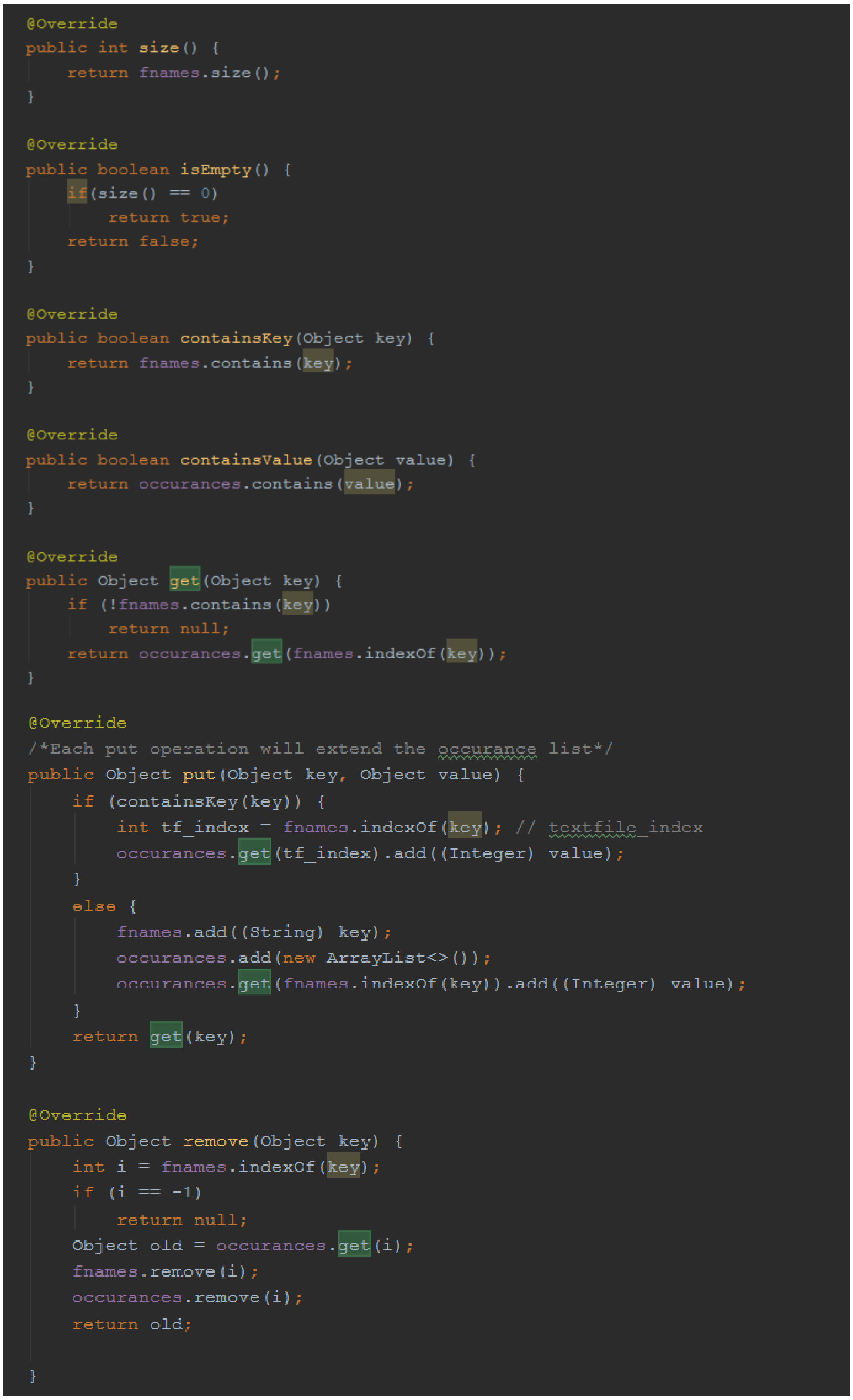
IDF(t) = log(Total number of documents / Number of documents with term t in it)

In code :

TF(t)=time_of_word/file_word_size.get(index);

IDF(t)=total_term/total_word;

FileMap Class



```
@Override
public int size() {
    return fnames.size();
}

@Override
public boolean isEmpty() {
    if(size() == 0)
        return true;
    return false;
}

@Override
public boolean containsKey(Object key) {
    return fnames.contains(key);
}

@Override
public boolean containsValue(Object value) {
    return occurances.contains(value);
}

@Override
public Object get(Object key) {
    if (!fnames.contains(key))
        return null;
    return occurances.get(fnames.indexOf(key));
}

@Override
/*Each put operation will extend the occurance list*/
public Object put(Object key, Object value) {
    if (containsKey(key)) {
        int tf_index = fnames.indexOf(key); // textfile_index
        occurances.get(tf_index).add((Integer) value);
    }
    else {
        fnames.add((String) key);
        occurances.add(new ArrayList<>());
        occurances.get(fnames.indexOf(key)).add((Integer) value);
    }
    return get(key);
}

@Override
public Object remove(Object key) {
    int i = fnames.indexOf(key);
    if (i == -1)
        return null;
    Object old = occurances.get(i);
    fnames.remove(i);
    occurances.remove(i);
    return old;

}
```

Complexity O(1).

Complexity O(1).

Complexity O(1).

Complexity O(1).

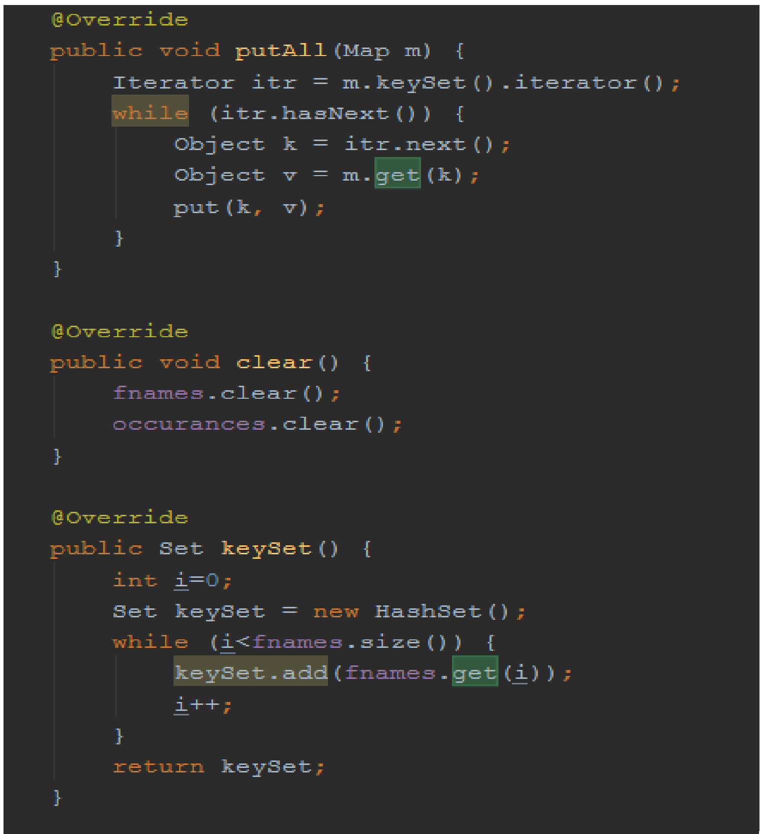
Complexity O(1).

This method add filename and occurence arraylist.

Complexity O(1).

Given key remove filemap arralists according to indexof ley.

Complexity O(1).



```java
@Override
public void putAll(Map m) {
    Iterator itr = m.keySet().iterator();
    while (itr.hasNext()) {
        Object k = itr.next();
        Object v = m.get(k);
        put(k, v);
    }
}
```

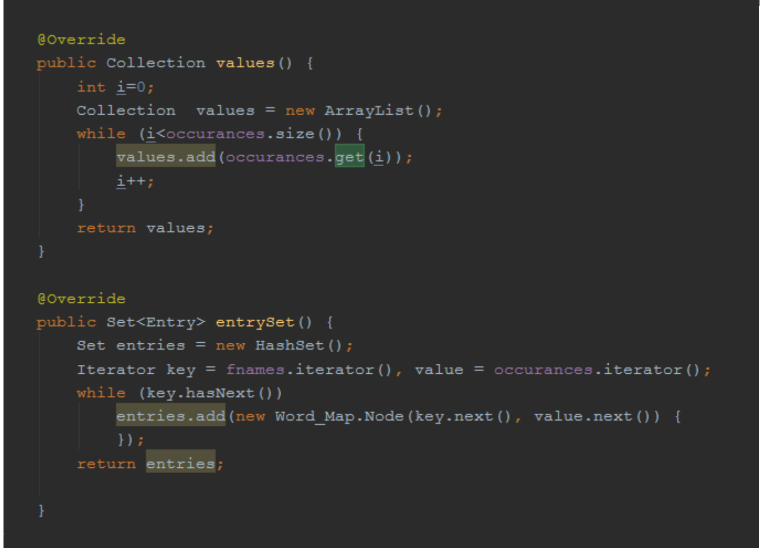
If we say all table (Node[] table) elements (with empty elements) number n , this method not turn empty cells so complexity is O(m). (number of node).

```java
@Override
public void clear() {
    fnames.clear();
    occurances.clear();
}
```

Clear fnames and occurances arraylist all elements. Complexity O(1).

```java
@Override
public Set keySet() {
    int i=0;
    Set keySet = new HashSet();
    while (i<fnames.size()) {
        keySet.add(fnames.get(i));
        i++;
    }
    return keySet;
}
```

If we say all table (Node[] table) elements (with empty elements) number n , this method not turn empty cells so complexity is O(m). (number of node).



```java
@Override
public Collection values() {
    int i=0;
    Collection  values = new ArrayList();
    while (i<occurances.size()) {
        values.add(occurances.get(i));
        i++;
    }
    return values;
}
```

If we say all table (Node[] table) elements (with empty elements) number n , this method not turn empty cells so complexity is O(m). (number of node).

```java
@Override
public Set<Entry> entrySet() {
    Set entries = new HashSet();
    Iterator key = fnames.iterator(), value = occurances.iterator();
    while (key.hasNext())
        entries.add(new Word_Map.Node(key.next(), value.next()) {
        });
    return entries;

}
```

If we say all table (Node[] table) elements (with empty elements) number n , this method not turn empty cells so complexity is O(m). (number of node).

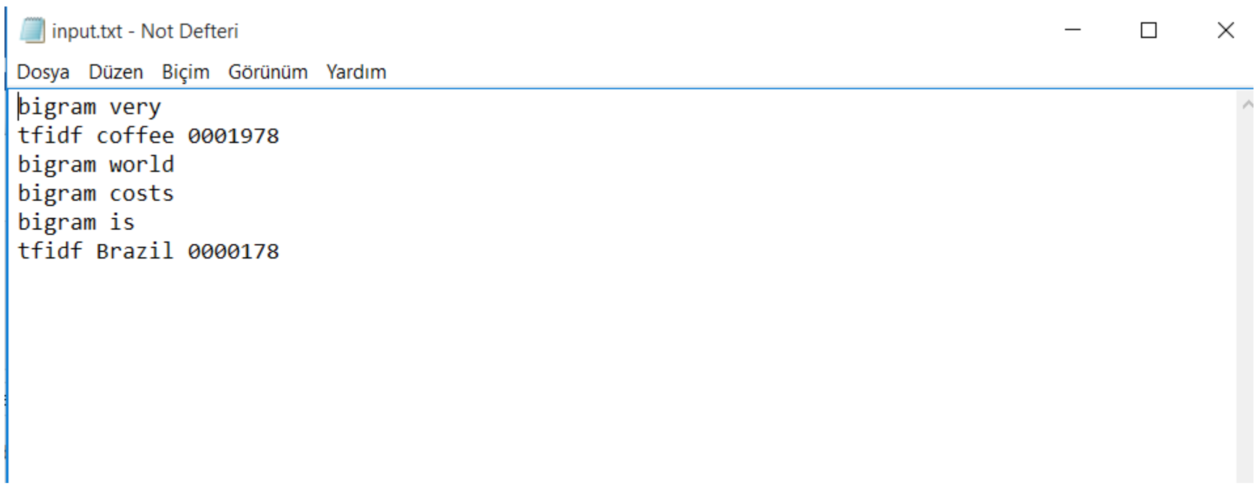
3 RESULT

3.1 Test Cases

Part 1

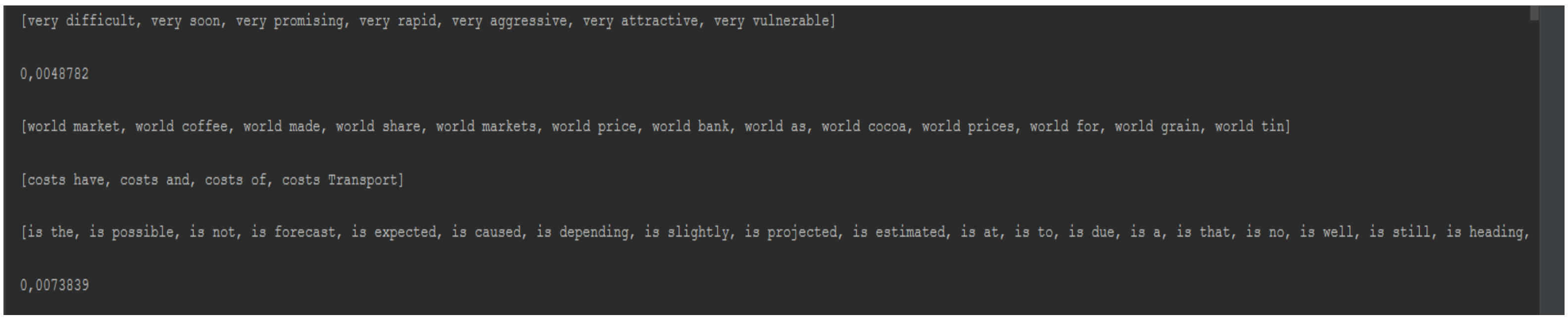
1. Firstly I give a directory path name in map to NLP readdataset method.
2. Secondly reading file to replace words, and creating arraylist which putting word location and file name, in hashmaps.
3. Reading input txt file check bigram and tfidf situation and call bigram and tfidf method according to this.
4. Print bigram or tfidf cases.
5. All test cases show below results.

3.2 Running Results



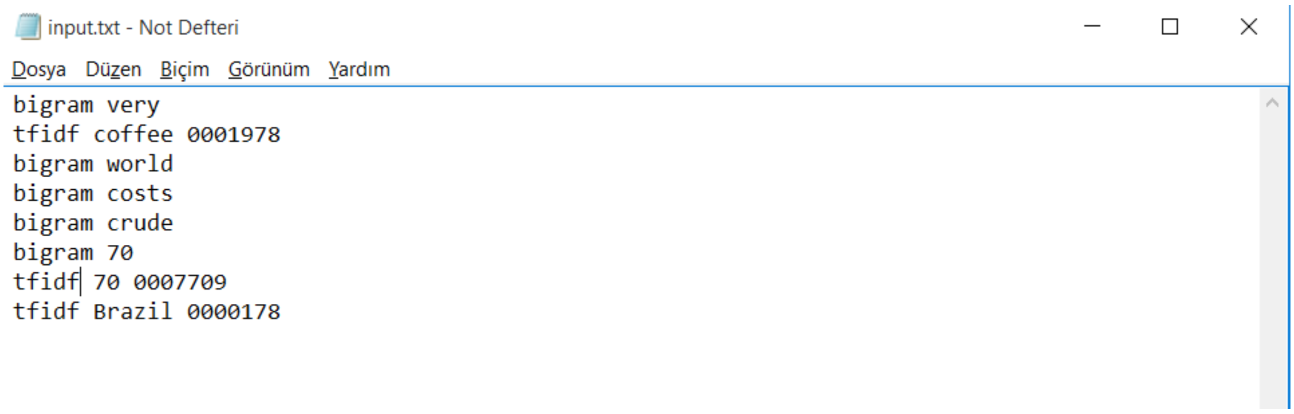
```
bigram very
tfidf coffee 0001978
bigram world
bigram costs
bigram is
tfidf Brazil 0000178
```

Given input file txt.



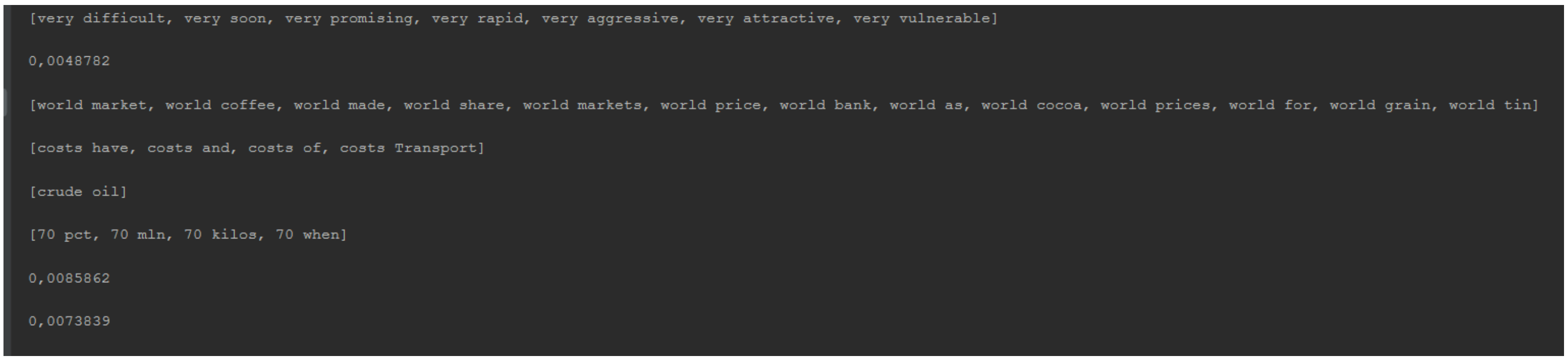
```
[very difficult, very soon, very promising, very rapid, very aggressive, very attractive, very vulnerable]

0,0048782

[world market, world coffee, world made, world share, world markets, world price, world bank, world as, world cocoa, world prices, world for, world grain, world tin]

[costs have, costs and, costs of, costs Transport]

[is the, is possible, is not, is forecast, is expected, is caused, is depending, is slightly, is projected, is estimated, is at, is to, is due, is a, is that, is no, is well, is still, is heading,

0,0073839
```

Output (is continue but does not appear on the screen).I try for given pdf input format.



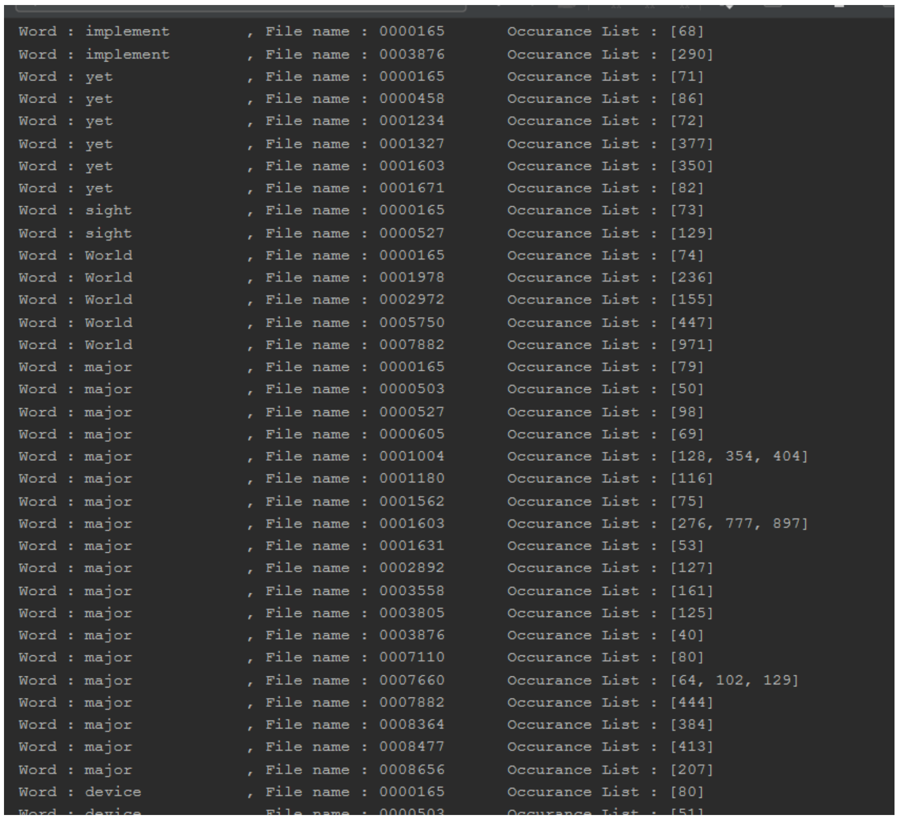
```
bigram very
tfidf coffee 0001978
bigram world
bigram costs
bigram crude
bigram 70
tfidf 70 0007709
tfidf Brazil 0000178
```

Given Input



```
[very difficult, very soon, very promising, very rapid, very aggressive, very attractive, very vulnerable]

0,0048782

[world market, world coffee, world made, world share, world markets, world price, world bank, world as, world cocoa, world prices, world for, world grain, world tin]

[costs have, costs and, costs of, costs Transport]

[crude oil]

[70 pct, 70 mln, 70 kilos, 70 when]

0,0085862

0,0073839
```

Output



```
Word : implement        , File name : 0000165      Occurance List : [68]
Word : implement        , File name : 0003876      Occurance List : [290]
Word : yet              , File name : 0000165      Occurance List : [71]
Word : yet              , File name : 0000458      Occurance List : [86]
Word : yet              , File name : 0001234      Occurance List : [72]
Word : yet              , File name : 0001327      Occurance List : [377]
Word : yet              , File name : 0001603      Occurance List : [350]
Word : yet              , File name : 0001671      Occurance List : [82]
Word : sight            , File name : 0000165      Occurance List : [73]
Word : sight            , File name : 0000527      Occurance List : [129]
Word : World            , File name : 0000165      Occurance List : [74]
Word : World            , File name : 0001978      Occurance List : [236]
Word : World            , File name : 0002972      Occurance List : [155]
Word : World            , File name : 0005750      Occurance List : [447]
Word : World            , File name : 0007882      Occurance List : [971]
Word : major            , File name : 0000165      Occurance List : [79]
Word : major            , File name : 0000503      Occurance List : [50]
Word : major            , File name : 0000527      Occurance List : [98]
Word : major            , File name : 0000605      Occurance List : [69]
Word : major            , File name : 0001004      Occurance List : [128, 354, 404]
Word : major            , File name : 0001180      Occurance List : [116]
Word : major            , File name : 0001562      Occurance List : [75]
Word : major            , File name : 0001603      Occurance List : [276, 777, 897]
Word : major            , File name : 0001631      Occurance List : [53]
Word : major            , File name : 0002892      Occurance List : [127]
Word : major            , File name : 0003558      Occurance List : [161]
Word : major            , File name : 0003805      Occurance List : [125]
Word : major            , File name : 0003876      Occurance List : [40]
Word : major            , File name : 0007110      Occurance List : [80]
Word : major            , File name : 0007660      Occurance List : [64, 102, 129]
Word : major            , File name : 0007882      Occurance List : [444]
Word : major            , File name : 0008364      Occurance List : [384]
Word : major            , File name : 0008477      Occurance List : [413]
Word : major            , File name : 0008656      Occurance List : [207]
Word : device           , File name : 0000165      Occurance List : [80]
Word : device           , File name : 0000503      Occurance List : [51]
```

Sample printWordMap method Output