Nikit Gokhe

Class – Comp D1

Roll No.224024

Gr No. 21810522

Assignment 1

Aim -: Create a file containing hashmap/ hashtable which includes correctly spelled words .Implement "ispell" Linux utility using above file.

Theory -:

HASHMAP:

- HashMap is a part of java.util package.
- HashMap extends an abstract class AbstractMap which also provides an incomplete implementation of Map interface.
- It also implements Cloneable and Serializable interface. K and V in the above definition represent Key and Value respectively.
- HashMap doesn't allow duplicate keys but allows duplicate values. That means A single key can't contain more than 1 value but more than 1 key can contain a single value.
- HashMap allows null key also but only once and multiple null values.
- This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time. It is roughly similar to HashTable but is unsynchronized.

HashMap provides 4 constructors and access modifier of each is public:

1. **HashMap():** It is the default constructor which creates an instance of HashMap with initial capacity 16 and load factor 0.75.
2. **HashMap(int initial capacity):** It creates a HashMap instance with specified initial capacity and load factor 0.75.
3. **HashMap(int initial capacity, float loadFactor):** It creates a HashMap instance with specified initial capacity and specified load factor.
4. **HashMap(Map map):** It creates instance of HashMap with same mappings as specified map.

HASHTABLE:

This class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value.
To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method.

- It is similar to HashMap, but is synchronised.
- Hashtable stores key/value pair in hash table.
- In Hashtable we specify an object that is used as a key, and the value we want to associate to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.

Constructors:

- **Hashtable():** This is the default constructor.
- **Hashtable(int size):** This creates a hash table that has initial size specified by size.
- **Hashtable(int size, float fillRatio):** This version creates a hash table that has initial size specified by size and fill ratio specified by fillRatio. **fill ratio:** Basically it determines how full hash table can be before it is resized upward.and its Value lie between 0.0 to 1.0
- **Hashtable(Map m):** This creates a hash table that is initialised with the elements in m.

Code :

SpellCheck.java

```
import java.util.ArrayList;

import java.util.Scanner;
```

```
public class SpellCheck {


   private Dictionary dict;

   final static String filePath =
"C:\\Users\\dharm\\OneDrive\\Desktop\\words.txt";

   final static char[] alphabet = "abcdefghijklmnopqrstuvwxyz".toCharArray();


   SpellCheck() {

      dict = new Dictionary();

      dict.build(filePath);


   }


   void run() {

      Scanner scan = new Scanner(System.in);

      boolean done = false;

      String input;


      while (true) {

         System.out.print("\n-------Enter a word: ");
```

```
        input = scan.nextLine();
        if (input.equals("")) {
            break;
        }




if (dict.contains(input)) {
            System.out.println("\n" + input + " is spelled correctly");
        } else {
            System.out.print("is not spelled correctly, ");
            System.out.println(printSuggestions(input));
        }
    }
 }

 String printSuggestions(String input) {
    StringBuilder sb = new StringBuilder();
    ArrayList<String> print = makeSuggestions(input);
    if (print.size() == 0) {
        return "and I have no idea what word you could mean.\n";
    }
    sb.append("perhaps you meant:\n");
    for (String s : print) {
        sb.append("\n  -" + s);
    }
```

```
        return sb.toString();
    }





private ArrayList<String> makeSuggestions(String input) {
        ArrayList<String> toReturn = new ArrayList<>();
        toReturn.addAll(charAppended(input));
        toReturn.addAll(charMissing(input));
        toReturn.addAll(charsSwapped(input));
        return toReturn;
    }

    private ArrayList<String> charAppended(String input) {
        ArrayList<String> toReturn = new ArrayList<>();
        for (char c : alphabet) {
            String atFront = c + input;
            String atBack = input + c;
            if (dict.contains(atFront)) {
                toReturn.add(atFront);
            }
            if (dict.contains(atBack)) {
                toReturn.add(atBack);
            }
        }
        return toReturn;
    }




private ArrayList<String> charMissing(String input) {
        ArrayList<String> toReturn = new ArrayList<>();

        int len = input.length() - 1;
        //try removing char from the front
        if (dict.contains(input.substring(1))) {
            toReturn.add(input.substring(1));
        }
        for (int i = 1; i < len; i++) {
            //try removing each char between (not including) the first and last
            String working = input.substring(0, i);
            working = working.concat(input.substring((i + 1), input.length()));
            if (dict.contains(working)) {
                toReturn.add(working);
            }
        }
        if (dict.contains(input.substring(0, len))) {
            toReturn.add(input.substring(0, len));
        }
        return toReturn;
    }


    private ArrayList<String> charsSwapped(String input) {
        ArrayList<String> toReturn = new ArrayList<>();


        for (int i = 0; i < input.length() - 1; i++) {
            String working = input.substring(0, i);// System.out.println("    0:" +
working);
            working = working + input.charAt(i + 1);  //System.out.println("    1:" +
working);
            working = working + input.charAt(i); //System.out.println("    2:" +
working);
            working = working.concat(input.substring((i + 2)));//System.out.println("
FIN:" + working);
            if (dict.contains(working)) {
                toReturn.add(working);
            }
        }
        return toReturn;
    }
          public static void main(String[] args) {
        SpellCheck sc = new SpellCheck();
        sc.run();
    }
}
```

Dictionary.java

```
import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;


public class Dictionary {

   private int M = 1319; //prime number

   final private Bucket[] array;

   public Dictionary() {

      this.M = M;


      array = new Bucket[M];

      for (int i = 0; i < M; i++) {

         array[i] = new Bucket();

      }

   }


   private int hash(String key) {

      return (key.hashCode() & 0x7fffffff) % M;

   }


   //call hash() to decide which bucket to put it in, do it.

   public void add(String key) {

      array[hash(key)].put(key);
```

```
    }

    //call hash() to find what bucket it's in, get it from that bucket.
    public boolean contains(String input) {
        input = input.toLowerCase();
        return array[hash(input)].get(input);
    }

    public void build(String filePath) {
        try {
            BufferedReader reader = new BufferedReader(new
FileReader(filePath));
            String line;
            while ((line = reader.readLine()) != null) {
                add(line);
            }
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }

    }
    //this method is used in my unit tests
    public String[] getRandomEntries(int num){
        String[] toRet = new String[num];
        for (int i = 0; i < num; i++){
            //pick a random bucket, go out a random number
            Node n = array[(int)Math.random()*M].first;
            int rand = (int)Math.random()*(int)Math.sqrt(num);

            for(int j = 0; j<rand && n.next!= null; j++) n = n.next;
            toRet[i]=n.word;


        }
        return toRet;
    }

     class Node {

            String word;
            Node next;

            public Node(String key, Node next) {
                this.word = key;
                this.next = next;
            }

        }
    class Bucket {

        private Node first;

        public boolean get(String in) {        //return key true if key exists
```

```
          Node next = first;
          while (next != null) {
              if (next.word.equals(in)) {
                  return true;
              }
              next = next.next;
          }
          return false;
      }

      public void put(String key) {
          for (Node curr = first; curr != null; curr = curr.next) {
              if (key.equals(curr.word)) {
                  return;                  //search hit: return
              }
          }
          first = new Node(key, first); //search miss: add new node
      }




  }
}
```

words.text

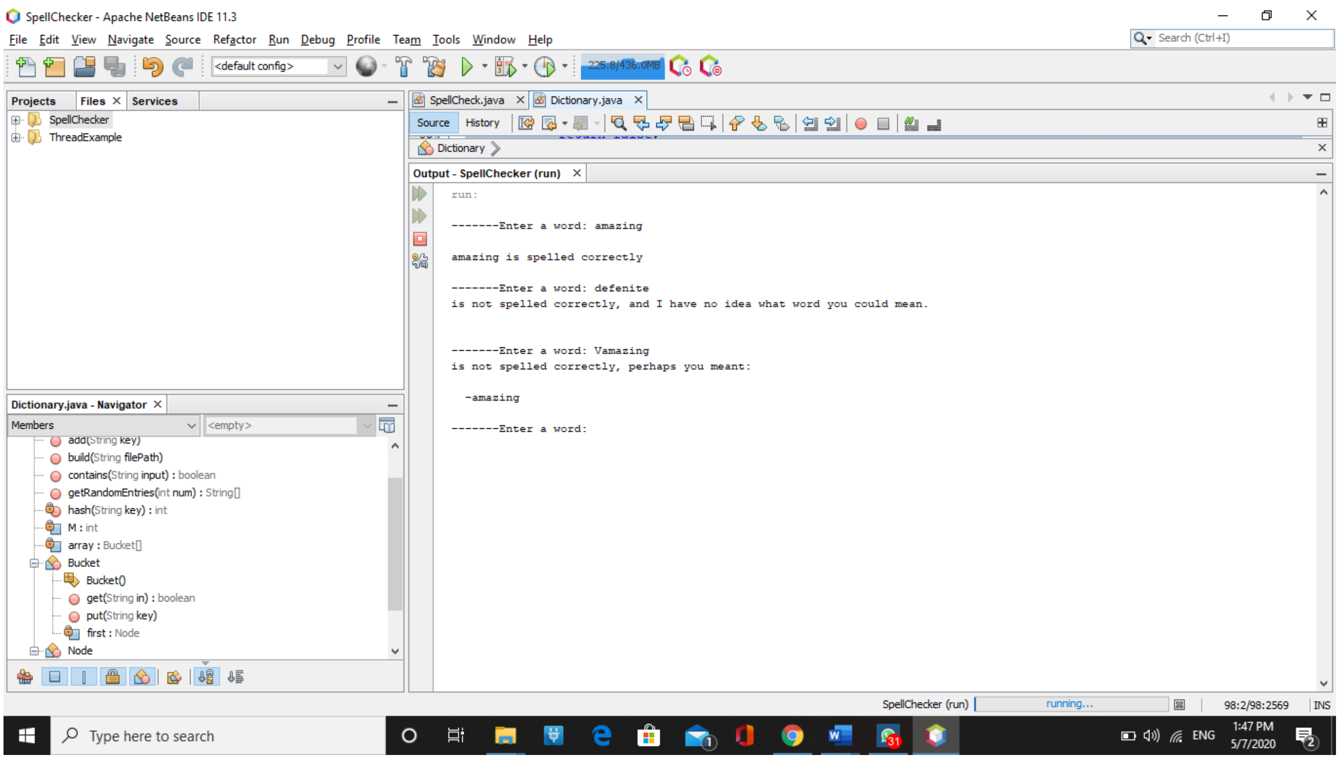
word

alphabet

zebra

amazing

Output :



SpellChecker - Apache NetBeans IDE 11.3
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services
SpellChecker
ThreadExample
SpellCheck.java Dictionary.java
Source History
Dictionary
Output - SpellChecker (run)
Dictionary.java - Navigator
Members
add(String key)
build(String filePath)
contains(String input) : boolean
getRandomEntries(int num) : String[]
hash(String key) : int
M : int
array : Bucket[]
Bucket
Bucket()
get(String in) : boolean
put(String key)
first : Node
Node
SpellChecker (run) running...
98:2/98:2569 INS
Type here to search
1:47 PM
5/7/2020

```
run:

-------Enter a word: amazing

amazing is spelled correctly

-------Enter a word: defenite
is not spelled correctly, and I have no idea what word you could mean.


-------Enter a word: Vamazing
is not spelled correctly, perhaps you meant:

  -amazing

-------Enter a word:
```