CS 102: Data Structures

Project One - Super Special Spellchecker

Student Name	Mohammad El-Abid
Student ID	8905652
Professor	Subrina Thompson
Date	Thursday, January 26, 2012

Table of Contents

Abstract

This application read from a static dictionary file and then iterated over the words inside of a text file whose location was specified by the user in the command line. If the program could not find the word in the “dictionary” it would consider it a typo and point it out to the user.

Introduction

I accomplished this application by using a `FileInputStream` to load the dictionary and create an array of “defined” words. Then iterate through the user defined file by use of `Scanner` and “spell checks” by ensuring every word in the file was included in the dictionary array. If there was no index with the same word (case ignored), it alerts the user by print to `STDOUT` (standard output stream).

Screenshots

Application output to `stdout`:

```
Line 01: Four score and seven yeres ago our furthurs brought forth on this
                            ^^^^^
Line 01: Four score and seven yeres ago our furthurs brought forth on this
                                          ^^^^^^^^
Line 04: Now we are engaged in a great civel war, testing whether that nation
                                       ^^^^^
Line 07: portion of that field, as a fonal resting plece for those who here
                                     ^^^^^
Line 07: portion of that field, as a fonal resting plece for those who here
                                                 ^^^^^

5 typos found in 23 lines.
```

Code

File: Application.java

```
// Mohammad El-Abid
// 1/27/12
// Bridgeport.edu

package edu.bridgeport.spellchecker;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;
import java.util.Comparator;
import java.util.Scanner;

class CompareWithoutCase implements Comparator<String> {

    @Override
    public int compare(String arg0, String arg1) {
           return arg1.compareToIgnoreCase(arg0);
    }

}

public class Application {

    private static String filename;

    /**
     * @param arguments
     */
    public static void main(String[] arguments) {
        String[] dict = null;

           // Check arguments
           if(arguments.length < 1) {
                System.err.println("Please pass in the file to spell
check.");
                System.exit(0);
           }

           // Build dictionary if expired
```

```
            File unsorted = new File("words.txt");
            File sorted   = new File("sorted-words.txt");
            if(unsorted.exists() == false){
                  System.out.println("words.txt is missing from running
path.");
                  System.exit(2);
            } else if(sorted.exists() == false ||
unsorted.lastModified() > sorted.lastModified()){
                  System.out.println("Building sorted-words.txt");

                  try {
                        dict = inputStreamToString(new
FileInputStream("words.txt")).split("[^A-Za-z]");
                  } catch (FileNotFoundException e) {
                        e.printStackTrace();
                        System.exit(3);
                  } catch (IOException e) {
                        e.printStackTrace();
                        System.exit(4);
                  }

                  Arrays.sort(dict);

                  try {
                        FileWriter fstream = new FileWriter("out.txt");
                        BufferedWriter out = new
BufferedWriter(fstream);
                        for(String word : dict) out.write(word + "\n");
                        out.close();
                        fstream.close();
                     sorted.setLastModified((new
java.util.Date()).getTime());
                  } catch (IOException e) {
                        e.printStackTrace();
                        System.exit(5);
                  }
                  System.out.println("Built successfully.");
                  System.out.println();
            } else {
                  // Load dictionary
                  String words_file = ""; // Suppresses "might be
uninitalized" when used

                  try {
                        words_file = inputStreamToString(new
FileInputStream("sorted-words.txt"));
                  } catch (IOException e) {
                        System.out.println("Unable to load sorted-
words.txt");
```

```
                    e.printStackTrace();
                    System.exit(1);
               }
               dict = words_file.split("\\s");
          }
          unsorted = sorted = null;

          // Load file
          filename = "";
          for(int i = 0; i < arguments.length; i++) {
               filename += arguments[i];
          }

          Scanner file = null;
          try {
               file = new Scanner( new FileInputStream(filename) );
          } catch (FileNotFoundException e) {
               e.printStackTrace();
               System.exit(0);
          }


          // Spell Check
          int typos = 0;
          int lineNumber = 0;

          while(file.hasNextLine()){
               lineNumber++;

               String    line = file.nextLine();
               String[] words = line.split("[^A-Za-z]"); // remove
all non-letters

               for(String word : words){
                    if( word.isEmpty() ) continue;

                    boolean inDictionary = binarySearch(dict, word,
new CompareWithoutCase()) != -1;
                    if(!inDictionary){
                         typos++;
                         String lineOutput = String.format("Line
%02d: ", lineNumber);
                         System.out.println( lineOutput + line);
                         for(int i = 0; i < line.indexOf(word) +
lineOutput.length(); i++) System.out.print(' ');
                         for(int i = 0; i < word.length(); i++)
System.out.print('^');
                         System.out.println();
                    }
```

```
                }
            }

            System.out.println();
            System.out.println(typos + " typos found in " + lineNumber
+ " lines.");
    }


    /**
     * @param in InputStream that you want returned as a String
     * @return String value of in
     */
    private static String inputStreamToString(InputStream in) throws
IOException {
        StringBuilder output = new StringBuilder();
        byte[] b = new byte[4096];
        for (int n; (n = in.read(b)) != -1;) {
            output.append(new String(b, 0, n));
        }
        return output.toString();
    }

    /**
     * @param elements to search for
     * @param search for this element
     * @param c is the comparator that will return zero when they
"match."
     * @return
     */
    private static <T> boolean itterativeSearch(T[] elements, T
search, java.util.Comparator<? super T> c){
    for( T element : elements ) if(c.compare(element, search) == 0)
return true;
    // default to false
    return false;
    }

    /**
     * @param elements to search (sorted)
     * @param search for this value
     * @param c is the Comparator instance to use
     * @return -1 if not found, otherwise the index of the element
     */
    private static <T> int binarySearch(T[] elements, T search,
java.util.Comparator<? super T> c){
    int start = 0;
    int stop  = elements.length - 1;
```

```
    while( start != stop ){
           int half = ((stop - start)/2) + start;
           int res = c.compare(elements[half], search);

           if( res == 0 )      return half;
           else if( stop - start <= 1 ) return -1;
           else if( res > 0 ) start = half;
           else if( res < 0 ) stop  = half;
    }

    return -1;
    }
}
```

Conclusion

This application allowed me to apply generics and apply 'Comparator's. I also had some fun using a binary search to find the words instead of an iterative search. Though I would have liked to implement a graphing algorithm instead, giving me $O(m)$ performance (with m being the characters in the string to find) instead of $O(log(n) + m)$ with n being the size of the dictionary. I was also surprised to learn that Java does not update the last modified time of files when you close them, but must be done by the programmer (via File#setLastModified).