Learning Journal Unit\_5

Dawid Blom

**Thursday, 06/05/2021, 8 – 11 am: Reading Assignment:** This week I read in the book about the Java Collections Framework. This framework is quite interesting because it allows us to use very powerful and complex data structures without having to implement them. Therefore, we can already create highly efficient programs by just choosing the correct the data structure for the problem. I also read about HashMaps and HashTrees, these data structures truly fascinate me. I would like to implement my own HashMap some day.

**Thursday, 06/05/2021, 12 – 13 pm: Discussion Assignment:** Choosing a specific data structure for a program can be tricky, however, I have found that when I ask myself certain questions I am able to better choose one. During this weeks discussion I was able to give some ideas on how I go about choosing a data structure for a program. I hope I was able to help some of the others and I also hope that there will be some improvements on my method of choosing.

**Friday, 07/05/2021, 12 – 4 pm: Programming Assignment:** I found the programming assignment quite challenging in the sense that the instructions was very unclear of what we’re supposed to do. However, I did a version of a spell checker that I hope will satisfy the rubric. Furthermore, I learned about a few methods that the String class has provides that was very useful in this assignment, such as the substring() method and the delimiter method.

**Friday, *07*/05/2021, 5 – 6 pm: Learning Journal:** Bellow you will find my code solution to this weeks lab, lab\_9.

/\*\*

\* Contents: SpellChecker class, as well as, the main method.

\*

\* Author: Dawid Blom.

\*

\* Date: 06/05/2021

\*/

package spell\_checker;

import java.io.\*;

import java.util.\*;

import javax.swing.JFileChooser;

/\*\*

\* A spell checking program for the English language.

\*

\* This program will read in a dictionary of words and

\* ask the user to type in a word, the program will then check

\* if the word typed in by the user is spelled correctly or not.

\* If the word is spelled correctly no suggestions are given,

\* however, if the word is miss spelled then the program will

\* provide suggestions of correctly spelled words.

\*/

public class SpellChecker

{

/\*\*

\* By calling the main method, the whole program will

\* be executed.

\*/

public static void main(String[] args)

{

spellCheckProgram();

}

/\*\*

\* By calling this method, it should provide suggestions

\* for the poorly spelled word. It will print out the miss

\* spelled word first and then print out any suggestions

\* for that word in the English language.

\*

\*

\* HOW THIS METHOD WORKS.

\* I start by creating a TreeSet<>() object.

\* I addAll() the corrections provided by the corrections()

\* method to the tree. I check if the tree is empty then there

\* will be no suggestions. However, if the tree is not empty,

\* I will print the the miss spelled word first and then

\* print all the related corrections there are for that word.

\*/

private static void suggestions(String badWord, Collection<String> dict)

{

TreeSet<String> outPutSuggestions = new TreeSet<>();

outPutSuggestions.addAll(corrections(badWord, dict));

if (outPutSuggestions.isEmpty())

{

System.out.println("No suggestion so far.");

}

else

{

System.out.print(badWord + ": ");

for (String word : outPutSuggestions) {

System.out.print(", " + word);

}

System.out.println();

}

}

/\*\*

\* By calling this method, it should check the spelling of

\* each word in the file that was provided by the user. If

\* a word is miss spelled it should provide suggestions.

\*

\*

\* HOW THIS METHOD WORKS

\* I start by creating a HashSet<>() object, I then continue on

\* into a try block where I create a Scanner to read in the file

\* provided by the user and entered as an argument, I then continue

\* on to set a delimiter for the words that are being read in. Next

\* I enter a while loop for as long as there are more words in the file.

\* I convert all the words being read in to lower case characters for easier use.

\* I check if the dictionary being used has the word being read in, in it. If it

\* does not have the word in it then I give suggestions of what the miss spelled

\* word could be. However, if the word is not miss spelled I continue on without

\* doing anything. Finally, I enter a catch block to catch a FileNotFoundException

\* in case the user did not provide a file to be checked.

\*/

private static void spellChecker(File file, Collection<String> dict)

{

Collection<String> badWords = new HashSet<>();

try

{

Scanner in = new Scanner(file);

in.useDelimiter("[^a-zA-Z]+");

while (in.hasNext())

{

String word = in.next().toLowerCase();

if (!dict.contains(word))

if (!badWords.contains(word))

{

suggestions(word, dict);

badWords.add(word);

}

}

in.close();

}

catch (FileNotFoundException e)

{

System.out.println("File could not be found: check spelling.");

}

}

/\*\*

\* By calling this method, I will check if the file provided by the user

\* contains any words that need correction. We will check by applying five different

\* strategies. By Swapping letters around, inserting a space in between letters,

\* delete certain letters, change a letter, and insert a different letter.

\*

\*

\* HOW THIS METHOD WORKS.

\* I start by creating a TreeSet<>() object.

\* I then continue to my first for-loop, this loop will only run as long

\* as the length of the word that's being check is.

\* I then check if I am out of bounds of the word or not.

\* If not, I continue to apply the swap correction and adding it to the TreeSet.

\* I then continue on to apply the space correction and adding it to the TreeSet.

\* After this, I begin my second loop to check each character in the word.

\* I continue on to apply the deleted correction and adding it the TreeSet.

\* Then I apply the changed correction and adding it to the TreeSet.

\* Finally, I apply the insert correction and adding it the TreeSet

\*

\* After all the corrections were made I return the TreeSet<>() object.

\*/

private static Collection<String> corrections(String badWord, Collection<String> dict)

{

Collection<String> correctSpelling = new TreeSet<>();

for (int i = 0; i < badWord.length(); i++)

{

if (i < badWord.length() - 1)

{

char[] ch = badWord.toCharArray();

char chTemp = ch[i];

ch[i] = ch[i + 1];

ch[i + 1] = chTemp;

String swapped = new String(ch);

if (dict.contains(swapped))

correctSpelling.add(swapped);

}

if (dict.contains(badWord.substring(0, i)) && dict.contains(badWord.substring(i)))

{

String space = badWord.substring(0, i) + ' ' + badWord.substring(i);

correctSpelling.add(space);

}

for (char ch = 'a'; ch <= 'z'; ch++)

{

String deleted = badWord.substring(0, 1) + badWord.substring(i + 1);

if (dict.contains(deleted))

correctSpelling.add(deleted);

String changed = badWord.substring(0, i) + ch + badWord.substring(i + 1);

if (dict.contains(changed))

correctSpelling.add(changed);

String inserted = badWord.substring(0, i) + badWord.substring(i);

if (dict.contains(inserted))

correctSpelling.add(inserted);

}

}

return correctSpelling;

}

/\*\*

\* By calling this method we will take the users selected

\* file and upload it to a HashSet data structure. The

\* data that will be uploaded to this collection is of

\* type String. Furthermore, this method will not take any

\* arguments.

\*

\* I will make this method private since it will

\* use data that should not be accessed by other

\* programs except, this one.

\*

\* HOW THIS METHOD WORKS.

\* I start by creating a collection HashSet<>(); names dict.

\* Then I create a variable of type File to hold the users

\* input that will be returned by the method getUserInput();.

\* After getting the user's input, we have to check if it is null

\* since the method getUserInput might return null.

\* I continue on to do exception handling in-case the

\* user provided a file that does not exist. I create a Scanner

\* object that will take the usersSelectedFile as an argument,

\* after that we run a while loop until we ran through the whole file.

\* I then create a String object named word that will take each individual

\* word and convert it toLowerCase(); I then add that word to the HasMap<>();

\* object dict. I continue this cycle until there are no more words to add.

\* I close the file for security reasons and not to further add complications.

\* After which I return the dict object for further use.

\*/

private static Collection<String> dictionary()

{

Collection<String> dict = new HashSet<>();

Scanner fileIn;

try

{

fileIn = new Scanner(new File("src/spell\_checker/words.txt"));

while (fileIn.hasNext())

{

String word = fileIn.next();

dict.add(word.toLowerCase());

}

fileIn.close();

}

catch (FileNotFoundException e)

{

System.out.println("File not found.");

}

return dict;

}

/\*\*

\* Lets the user select an input file using a standard file

\* selection dialog box. If the user cancels the dialog

\* without selecting a file, the return value is null.

\*

\* HOW THIS METHOD WORKS.

\* I start by creating a JFileChooser object named fileDialog.

\* Then I continue to set the title of the dialog box.

\* I then continue to let the user choose a file they want to

\* add, if this file does not exist in the options provided this

\* method will return null, else it will return the file selected.

\*/

public static File getUserInput()

{

JFileChooser fileDialog = new JFileChooser();

fileDialog.setDialogTitle("Select File for Input");

int option = fileDialog.showOpenDialog(null);

if (option != JFileChooser.APPROVE\_OPTION)

return null;

else

return fileDialog.getSelectedFile();

}

/\*\*

\* By Calling this method, it should initiate the program

\* by calling all the other methods related to the program.

\*

\*

\* HOW THIS METHOD WORKS.

\* I start by loading the English dictionary on the system

\* into an object called dict.

\* I then do error checking to make sure everything is all right.

\* I ask the user to provide a file that needs to be checked.

\* I continue on to do error checking if it passed all the error checking

\* I continue on to perform the spell check.

\*/

private static void spellCheckProgram()

{

Collection<String> dict = dictionary();

if (dict.isEmpty())

{

System.out.println("Please put the file 'words.txt' in this path: src/spell\_checker/words.txt");

System.exit(0);

}

System.out.println("Please enter a file to be checked.");

File file = getUserInput();

if (file == null)

{

System.out.println("No file found.");

System.exit(0);

}

else

{

System.out.println("Missed spelled words:");

spellChecker(file, dict);

}

}

}