Project 9: Spell Checker

Due: Monday, Oct. 28 at 11:59 PM

Description: Many software programs include a spell checker that highlights misspelled words. A basic spell checker works by comparing user input to the words stored in a dictionary. If a given word is found, the checker assumes it is spelled correctly. Otherwise, the checker assumes it is incorrectly spelled.[1]

Good spell checkers use two dictionaries to check spellings: a dictionary of common words and a personal dictionary. The common dictionary contains typical words and is rarely, if ever, updated. The personal dictionary contains words not stored in the common dictionary that the user deems as correct. This may include words in other languages or technical jargon like "mutator" or "polymorphism." The personal dictionary may be updated by the user at any time.

In this project, we will create a spell checker! The checker uses a perfect size array to store a common dictionary and an oversize array to store a personal dictionary. The user enters words at the console, and the program looks for each word in the arrays. If a word is found, the user is told the spelling is correct. If a word is not found, the user can add it to the oversize array.

When the user quits, the words in the oversize array are saved to a text file. This allows the spell checker to remember the personal dictionary the next time it runs.

Objectives: Your program will be graded according to the rubric below.

1. (10 points) Write the method readFilePerfect. This method reads a text file with a given name and returns a perfect size array that contains each line of the file.
2. (20 points) Write the method readFileOversize. This method reads a text file with a given name, adds each line to an element of an oversize array, and returns the number of lines. The method creates an empty file with the given name if one does not exist.
3. (20 points) Write the method checkSpelling. This method checks if a given word can be found in either of two arrays, one perfect size and the other oversize.
4. (20 points) Write the method writeFile. This method writes each element of an oversize array to a file with a given name.
5. (20 points) Write the main method. This method uses the methods from objectives 1–4 to implement a working spell checker.
6. (10 points) Use meaningful variable names, consistent indentation, and whitespace (blank lines and spaces) to make your code readable. Add comments where appropriate to explain the overall steps of your program.

[1] This strategy has numerous well-known shortcomings. In particular, the spell checker cannot tell if a word is being used properly, only whether it exists in the dictionary. For instance, if you type "here" when you mean "hear," the spell checker will not catch this mistake.

Sample Output: Below is an example run of the program. The user input is in bold. Format your print statements so your output matches if given the same input.

```
Spell Checker
-------------
Enter a word or 'quit' to stop: compiler
The word is spelled correctly.

Enter a word or 'quit' to stop: polymorphism
The word was not found in the dictionary.
Would you like to add it to your personal dictionary (yes/no)?
yes

Enter a word or 'quit' to stop: polymorphism
The word is spelled correctly.

Enter a word or 'quit' to stop: quit
Goodbye!
```

Common Dictionary: The common dictionary is stored in the file "common-dictionary.txt" with one word on each line.[2] Add this file to the root of your project folder, just as you did with the data file in Project 8.

Method Descriptions: In projects 7 and 8, we gave you the signatures of the methods you had to write. In this project, you need to determine the signatures yourself. Below are the names and descriptions of the methods needed to implement the spell checker. Before you begin coding each method, think carefully about what its parameters and return type should be.

1) `readFilePerfect`

Given the name of a text file, this method returns the lines of the file as elements of a perfect size String array. The main method calls readFilePerfect to read the common dictionary text file.

- Use a Scanner to read the lines of the file. The Scanner must be constructed with a File object as the argument, rather than System.in.
- The File class is not part of the java.lang package, so it must be imported with the statement "import java.io.File;".
- Any method that constructs a File object must be able to throw a file-not-found exception. To allow readFilePerfect to do this, add the statement "throws FileNotFoundException" after the close parenthesis of the method signature.

[2] The common dictionary was created from the file "google-10000-english-usa-no-swears.txt" hosted in the following GitHub repository: https://github.com/first20hours/google-10000-english. The file contains the 10,000 most common English words (excluding swear words) found in Google's Trillion Word Corpus. The common dictionary was created by removing words with less than five letters and sorting the result.

- The file-not-found exception must be imported with the statement "import java.io.FileNotFoundException;".
- This method is essentially identical to a method you wrote in Project 8. Refer to that project if you get stuck here.

2) `readFileOversize`

Given the name of a text file and a String array, this method adds the lines of the file to successive elements of the array (starting from the first element) and returns the number of lines. The main method calls readFileOversize to read the personal dictionary text file.

- When the spell checker runs for the first time, the personal dictionary will not exist. Include the following code at the start of the method to create an empty text file:[3]

  ```
  File file = new File(filename);
  file.createNewFile();
  ```

  The variable filename should contain the name passed to the method. Note that the object used to create the file can also be used to construct a Scanner that reads the file.
- Any method that calls createNewFile must be able to throw an input-output exception. To allow readFileOversize to throw both this and a file-not-found exception, add the statement "throws FileNotFoundException, IOException" after the close parenthesis of the method signature.
- The input-output exception must be imported with the statement "import java.io.IOException;".
- The code that reads the file and stores the lines in an array is similar to that of readFilePerfect. However, this method is simpler because the file does not need to be read twice. (The body of the method should have only one loop.) Simply add each line to the next empty element of the array. This is an advantage of using an oversize array.
- If the given array is not large enough to store all the lines of the file, readFileOversize should add as many lines as will fit.

3) `checkSpelling`

Given a perfect size array of words, an oversize array of words, and a word, this method returns true if the word is in either array. Otherwise, it returns false. The main method calls checkSpelling to see if a word is in either dictionary.

- One way to write this method is to loop through both arrays and compare each element with the given word. (If you do this, use the equals method, not the == operator.)
- A better way to write the method is to use binary search, which is significantly faster than sequential search. Note that different methods in the Arrays class are needed to perform binary search on perfect size and oversize arrays. (See the documentation.)

[3] The method createNewFile will do nothing if a file with the given name already exists.

4) `writeFile`

Given the name of a text file and an oversize String array, this method writes the (nonempty) array elements to successive lines of the file. The main method calls writeFile to save the personal dictionary before the program ends.

- Use a PrintWriter object to write each element to the file. Construct the PrintWriter with a File object as the argument:

  ```
  PrintWriter writer = new PrintWriter(new File(filename));
  ```

  The variable filename should contain the name passed to the method. Note the similarities between this statement and the code that constructs a Scanner to read a file.

- The PrintWriter class is not part of the java.lang package, so it must be imported with the statement "import java.io.PrintWriter;".

- The PrintWriter class has a method named "println" that is similar to the method System.out.println. The PrintWriter method, however, can write its argument to a file, rather than the console. For example, the following statement writes the message "Hello!" using the PrintWriter constructed above:

  ```
  writer.println("Hello!");
  ```

- After you write the array elements to the file (using a loop), you must close the PrintWriter to save the file. Otherwise, the data will be lost. This can be done with the close method, just like closing a Scanner.

- writeFile must be able to throw a file-not-found exception.

5) `main`

The main method uses the methods described above to implement the spell checker. (Note that the main method must be able to throw any exception that a method it calls can throw.) Below is an ordered list of the steps that the method needs to perform.

1. Call the method readFilePerfect to read the file "common-dictionary.txt" and store the words in a perfect size array.
2. Call the method readFileOversize to read the file "personal-dictionary.txt" and store the words in an oversize array. (If the method is implemented correctly, it will create the file when the program first runs.) The oversize array must be constructed in the main method. Give the array a capacity of 100.
3. Use a loop to prompt and read words from the user until they enter "quit."
4. For each word that is entered, call the method checkSpelling to see if it is in the dictionaries. If the word is found, tell the user it is spelled correctly.

5. If the word is not found and there is space in the oversize array, ask the user if they want to add the word to the personal dictionary. If the user enters “yes,” add the word to the next empty spot in the array. (Don’t forget to increment the size variable.)
6. If you are using binary search in checkSpelling, you must sort the personal dictionary array after adding a word.[4] (See the Arrays class API documentation for a sort method that works with oversize arrays.)
7. When the user quits the program, call the method writeFile to save the personal dictionary to the file “personal-dictionary.txt”.

Submission Instructions: Submit your source code to Canvas in a .zip file. Below are instructions for creating this file in Eclipse.

1. Click “File” in the top-left corner and then “Export…” from the drop-down menu.
2. In the Export window, click the arrow next to “General,” then click “Archive File,” then click the “Next” button.
3. Click the arrow next to your project folder and then check the box next to the src folder.
4. In the “To archive file” field, browse to a convenient location to create the .zip file (e.g., your desktop) and name the file “Project9”. Click the “Finish” button.
5. Upload the .zip file to Canvas.

[4] Sorting takes longer than both binary and sequential search. For this reason, we only want to sort the array after a new word is added, since the array will remain sorted until then.