Project 12: Refactoring the Spell Checker

Due: Monday, Apr. 20 at 11:59 PM

Description: In Project 9 we created a spell checker that used both a perfect size and an oversize array of words. The perfect size array stored the common dictionary, which never changed, and the oversize array stored the personal dictionary, which had extra space for the user to add words.

One problem with this approach is that the capacity of the oversize array is fixed. If the capacity is too small, the user will run out of space. If the capacity is too large, the program wastes memory.

A better approach is to use an oversize array of moderate capacity (say, 10 elements), and if the user runs out of space, construct a new, larger array. This is exactly how the ArrayList class works. If an element is added to a full ArrayList, the existing elements are copied to a larger array with space for the new element.

In this project, we will refactor the spell checker to use ArrayLists instead of arrays. This will require us to change the signature and body of each method. When we are finished, the program will work as it did before, but the code will be shorter and easier to understand, since the details of ArrayList objects are hidden (encapsulated) inside the ArrayList class.

Objectives: Update the spell checker to use ArrayLists of Strings, rather than normal String arrays. The new version should include the changes listed below. The exact method signatures are given in a table after the objectives.

1. (20 points) Replace the methods readFilePerfect and readFileOversize with a single method named “readFile”. This method reads a file with a given name and returns the lines in an ArrayList of Strings. If the file does not exist, readFile creates an empty text file with the given name.
2. (20 points) Refactor the method checkSpelling. The old method checks if a given String can be found in either of two arrays, one perfect size and the other oversize. The new method checks if the String can be found in either of two ArrayLists.
3. (20 points) Refactor the method writeFile. The old method writes each String in an oversize array to a file with a given name. The new method is given an ArrayList of Strings, rather than an oversize array.
4. (20 points) Refactor the main method. Store the common and personal dictionaries in ArrayLists instead of arrays. Call the refactored methods from objectives 1–3 in place of the old methods so that the program works exactly like the original version. (A user of the program will not be able to see a difference.)
5. (10 points) Remove any unnecessary import statements and local variables from the program.
6. (10 points) Use meaningful variable names, consistent indentation, and whitespace to make your code readable. Add comments to explain the overall steps of your program.

public static ArrayList<String> readFile(String fileName)
public static boolean checkSpelling(String word, ArrayList<String> common, ArrayList<String> personal)
public static void writeFile(String fileName, ArrayList<String> list)

Getting Started: To ensure that everyone starts with a complete version of the original program, we are providing you with our spell checker code. Please refactor this code, rather than the code you submitted for Project 9.

Hints and Suggestions:

1. Before modifying the code, import it into a new project (along with the common dictionary text file) and check that it runs correctly. (See the Project 9 instructions for a test case.)
2. Refer to the API documentation for the ArrayList class if you've forgotten the import statement or method signatures.
3. When writing the method readFile, note that it is no longer necessary to count the number of lines in the file. Simply create an ArrayList with the default constructor and add each word as it is read. The capacity of the list will increase automatically.
4. The program needs to sort the personal dictionary and perform binary search on both ArrayLists. The original program used methods of the Arrays class to perform these operations. These methods only work with normal arrays, not ArrayLists. To sort and search ArrayLists, we need to use methods from the Collections class instead.
5. Test each method after you finish changing it. This will require you to comment out the old code in the main method. Eclipse makes this easy: select the code you wish to comment and press CTRL+/. Alternatively, you can use the block comment symbols /* and */.

Upload Code to Zybooks: After you complete each project, you need to upload your source code to Zybooks so it can be graded. If you created the folder structure described earlier, your code is in the folder "Intro to Programming\Projects\Project 12\src". The code is contained in the file Project_12.java. Note that .java files are simply text files that contain Java code. They can be opened with any text editor (e.g., Notepad or TextEdit).

Upload your .java file to Zybooks on the Project 12 assignment page. This requires a few steps. Click the "Submit Assignment" button in the top-right corner. This will reveal a button near the bottom of the page with the text "Choose File." Click this button and browse to the location of your .java file. Finally, click the "Submit for grading" button below the "Choose on hard drive" button.

If Zybooks won't accept the file, make sure you're submitting a .java file, not a .class file. Make sure to see if the output you obtained and the output which I have mentioned match to obtain complete grade.