Project 3: Similar Sounding Words

DUE: Sunday, April 3rd at 11:59pm

Basic Procedures

You must:

- Fill out a readme.txt file with your information (goes in your user folder, an example readme.txt file is provided)
- Have a style (indentation, good variable names, etc.) and pass the automatic style checker (see P0).
- Comment your code well in JavaDoc style AND pass the automatic JavaDoc checker (see P0).
- Have code that compiles with the command: **`javac *.java`** in your user directory.

You may:

- Add additional methods and variables, however these methods **must be private OR protected**.
- Add additional nested, local, or anonymous classes, but any nested classes must be **private**.
- Add methods **required** by parent abstract classes and methods required by interfaces you must implement.

You may NOT:

- Make your program part of a package.
- Add additional ***public*** methods, variables, or classes. You may have public methods in ***private*** nested classes.
- Use any built in Java Collections Framework classes in your program unless specified at the top of the provided templates.
- Alter any method signatures defined in this document or the template code. Note: "throws" is part of the method signature in Java, don't add/remove these.
- Add any additional import statements (or use the "fully qualified name" to get around adding import statements).
- Add any additional libraries/packages which require downloading from the internet.

Setup

- Download the `p3.zip` and unzip it. This will create a folder `section-yourGMUUserName-p3`;
- Rename the folder replacing `section` with the `001`, `002`, `005` etc. based on the lecture section you are in;
- Rename the folder replacing `yourGMUUserName` with the first part of your GMU email address;
- After renaming, your folder should be named something like: `001-jsmith-p1.`
- Complete the `readme.txt` file (an example file is included: `exampleReadmeFile.txt`)

Submission Instructions

- Make a backup copy of your user folder!
- Remove all test files, jar files, class files, etc.
- You should just submit your java files and your readme.txt
- Zip your user folder (not just the files) and name the zip `section-username-p3.zip` (no other type of archive) following the same rules for `section` and `username` as described above.
    - The submitted file should look something like this:

```
001-krusselc-p3.zip --> 001-krusselc-p3 --> JavaFile1.java
                                            JavaFile2.java
                                            ...
```

- Submit to blackboard.

Grading Rubric

Due to the complexity of this assignment, an accompanying grading rubric pdf has been included with this assignment. Please refer to this document for a complete explanation of the grading.

Overview

You are provided with a file, ***word_to_sound.txt***, that maps a given word to its corresponding sequence of "units of sound" or *unisound*s. There are 134,290 lines in the file, where each line maps one word to its corresponding sound.

```
COMBATS  K AH0 M B AE1 T S
IDIOTIC  IH2 D IY0 AA1 T IH0 K
ILIANO  IH2 L IY0 AA1 N OW0
RISKY  R IH1 S K IY0
ELECTRO  IH0 L EH1 K T R OW0
DANSFORTH'S  D AE1 N S F AO1 R TH S
SHAREWARE  SH EH1 R W EH2 R
RECYCLES  R IY0 S AY1 K AH0 L Z
LABODA  L AA0 B OW1 D AH0
…
…
ENTHUSIASTICALLY  IH0 N TH UW2 Z IY0 AE1 S T IH0 K L IY0
MCWETHY  M AH0 K W EH1 TH IY0
FLUOR  F L UW1 ER0
OGDEN'S  AA1 G D AH0 N Z
WALN  W AO1 L N
…
…
ENTER  EH1 N T ER0
UNDERSTUDY  AH1 N D ER0 S T AH2 D IY0
TRAINA  T R EY1 N AH0
MACHINIMA  M AH0 SH IY1 N IH0 M AH0
PHARAON  F EH1 R OW0 N
MENIFEE  M EH1 N IH0 F IY2
LARY  L EH1 R IY0
ADDLEMAN  AE1 D AH0 L M AH0 N
```

For example, on line 61,906 "ENTHUSIASTICALLY IH0 N TH UW2 Z IY0 AE1 S T IH0 K L IY0" indicates that when you pronounce the word "ENTHUSIASTICALLY", the *unisounds* you speak are actually "IH0 N TH UW2 Z IY0 AE1 S T IH0 K L IY0". The numbers 0, 1, and 2, are not meant to be pronounced, but are used to emphasize a given *unisound*. That is, "UW2" should be most emphasized, then comes "AE1", followed by {"IH0", "IY0"}, and finally {"N", "TH", "Z", "S", "T", "K", "L"}.

For this project, two words are considered to have similar sounds if they share (or belong to) the same *sound-group*, which is the **trailing sequence of *unisounds* starting from the last occurring most emphasized *unisound***. To clarify, consider the seven entries below in which the *sound-group* for each word is underlined:

PHOTOGRAPHER	F AH0 T AA1 G R AH0 F ER0
BIOGRAPHER	B AY0 AA1 G R AH0 F ER0
GEOGRAPHER	JH IY0 AA1 G R AH0 F ER0
DEMOGRAPHER	D IH0 M AA1 G R AH0 F ER0
PHOTOGRAPHERS	F AH0 T AA1 G R AH0 F ER0 Z
CINEMATOGRAPHER	S IH2 N IH0 M AH0 T AA1 G R AH0 F ER0
LEXICOGRAPHER	L EH2 K S IH0 K AA1 G R AH0 F ER0
ST_MARTIN	S EY1 N T M AA1 R T IH0 N

- Similarly sounding words:
  "PHOTOGRAPHER", "BIOGRAPHER", "GEOGRAPHER", and "DEMOGRAPHER" are similarly sounding since they all share the same *sound-group* "AA1 G R AH0 F ER0". Note how that "AA1" is the most emphasized *unisound*.

- Not similarly sounding words:
  "PHOTOGRAPHERS", "CINEMATOGRAPHER", and "LEXICOGRAPHER" are not similarly sounding to any other word in the above list, despite the fact that they also share "AA1 G R AH0 F ER0". For example, "CINEMATOGRAPHER" is not similar to "PHOTOGRAPHER" because its *sound-group* is "IH2 N IH0 M AH0 T AA1 G R AH0 F ER0" as opposed to "AA1 G R AH0 F ER0".
  "PHOTOGRAPHERS" is not similar to "PHOTOGRAPHER" because its *sound-group* is "AA1 G R AH0 F ER0 Z" and not "AA1 G R AH0 F ER0".

- *Sound-group* identification:
  The *sound-group* for "LEXICOGRAPHER" starts at "EH2" since it is the most emphasized *unisound*.
  Considering "ST_MARTIN", there are two (equally) most emphasized *unisounds,* namely, "EY1" and "AA1". In such a case, the *sound-group* starts with "AA1" since it occurs later than "EY1".

Deliverables

You are asked to write a program SimilarSounds that provides the following two functionalities:

1. When more than one argument is passed, i.e., "java SimilarSounds word1 word2 word3 word4 ...", the program must identify the similar sounding words amongst the ones passed on the command line. The three examples below illustrate the exact expected behavior.

   ```
   java SimilarSounds yellow mellow carmelo fellow modelo
   Output:
   "yellow" sounds similar to: "mellow" "carmelo" "fellow" "modelo"
   Unrecognized words: none

   java SimilarSounds calculated legislated hello world miscalleneous miscalculated encapsulated LIBERATED Sophisticated perculated hello
   Output:
   "calculated" sounds similar to: "legislated"
   "hello" sounds similar to: none
   "world" sounds similar to: none
   "miscalculated" sounds similar to: "encapsulated" "LIBERATED" "Sophisticated"
   Unrecognized words: "miscalleneous" "perculated"

   java SimilarSounds pontiac chevrolet honda toyota tesla zoolander chevy
   Output:
   "pontiac" sounds similar to: none
   "chevrolet" sounds similar to: none
   "honda" sounds similar to: none
   "toyota" sounds similar to: none
   "tesla" sounds similar to: none
   "chevy" sounds similar to: none
   Unrecognized words: "zoolander"
   ```

   Note how: a) the earlier words in the argument list are orderly compared to the subsequent ones; b) if a word was already deemed similar, then it will be ignored hereafter; c) the behavior is case insensitive; d) the subsequent occurrence of a given word is ignored (e.g., the second occurrence of "hello" is ignored); e) words that couldn't

be found in the database are deemed unrecognizable (e.g., "miscallenous" and "perculated"); f) in the output, the words are enclosed double quotes.
Make sure that your output adheres to the following form:

"…" sounds similar to: …
"…" sounds similar to: …
…
Unrecognized words: …

2. When only one argument is passed, i.e., "java SimilarSounds word", the program must identify all the words in the database that are similar to the word passed on the command line. The four examples below illustrate the exact expected behavior.

java SimilarSounds biology
Output:
Words similar to "biology": "ASTROLOGY" "BIOLOGY" "CHRONOLOGY" "CYTOLOGY" "DEONTOLOGY" "DOXOLOGY" "ECOLOGY" "ECOLOGY(1)" "ETHNOLOGY" "ETHOLOGY" "GEOLOGY" "GRAPHOLOGY" "HISTOLOGY" "HYMNOLOGY" "LIMNOLOGY" "MORPHOLOGY" "MYCOLOGY" "NEUROLOGY" "NUMEROLOGY" "ONCOLOGY" "ONTOLOGY" "OTOLOGY" "PATHOLOGY" "PENOLOGY" "PETROLOGY" "POMOLOGY" "PSYCHOLOGY" "SEROLOGY" "TECHNOLOGY" "THEOLOGY" "UROLOGY" "VIROLOGY" "ZOOLOGY"

java SimilarSounds dimension
Output:
Words similar to "dimension": "ASCENSION" "ATTENTION" "CONTENTION" "CONVENTION" "DECLENSION" "DETENTION" "DIMENSION" "DISSENSION" "EXTENSION" "GENTIAN" "HENSCHEN" "LAURENTIAN" "MENTION" "PENSION" "PRETENSION" "PREVENTION" "RETENTION" "SUSPENSION" "TENSION"

java SimilarSounds University
Output:
Words similar to " University": "UNIVERSITY"

java SimilarSounds hellow
Output:
Unrecognized word: "hellow"

Note how the word passed as an argument must still appear in the output. However, if it cannot be found in the database an appropriate error message should be displayed, specifically: "Unrecognized word: …".
Make sure that your output adheres to the following forms:

Words similar to "…": …
Or
Unrecognized word: …

Implementation/Classes

There are two phases to this project:
Phase1: Provide the functionality previously described, while using java.util.HashMap (you can learn more about this class in *https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html*).
Phase2: Provide the functionality previously described, while using your own hash map, specifically LinearProbingMap. All the code you wrote to implement Phase1 should be used "as is" to implement Phase2; you will merely need to: 1) implement LinearProbingMap; and 2) invoke SimilarSoundsMyHM (provided) instead of SimilarSounds.

Phase1

This phase will be built using three classes that we describe below. Template files are provided for each class, which contain additional descriptions.

- **Extractor** (Extractor.java): this class has four helper methods to be used in SimilarSounds.java. In your implementation of these methods, you are allowed to use StringTokenizer and String.split( ).

    - public static List<String> readFile(String fileName): **Provided – Do Not Change**
      Used to read the entries/lines in ***word_to_sound.txt*** and returns them as a List.

    - public static String extractWordFromLine(String line): Given a line of text, this method must return the first token (block of characters). In the context of this project, it will return the "word" whose sound is described.

    - public static String extractSoundFromLine(String line): Given a line of text, this method returns all tokens except the first one. In the context of this project, it will return the "sound" (sequence of *unisounds*).

    - public static String extractSoundGroupFromSound(String Sound): Given a string representing the sound of a word (sequence of *unisounds*), this method returns the *sound-group*, i.e., "the trailing sequence of *unisounds* starting from the last occurring most emphasized *unisound*".

- **BST<T extends Comparable<T>>** (BST.java): this is an implementation of a BST

    - public void insert(T key): This method inserts a Node<T> in the BST. You can implement it iteratively or recursively. Note: you can create private helper methods

    - public Node<T> find(T key): This method finds and returns a Node<T> in the BST. You can implement it iteratively or recursively, it should return *null* if no match is found. Note: you can create private helper methods

    - public String inorderToString(): This method returns a string in which the elements are listed in an *inorder* fashion. Your implementation must be recursive. Note: you can create private helper methods.

    - public String toString(): **Provided – Do Not Change**
      This method simply invokes "String inorderToString()", as a result the returned string will contain the elements in ascending order.

- **SimilarSounds** (SimilarSounds.java)**:** this class implements the main functionalities of this project. You are allowed to use java.util.ArrayList in this class.

    - public static void populateWordToSoundMap(List<String> lines): Given a list of all entries in ***word_to_sound.txt***, this method populates the wordToSound Map as follows: the *key* is the word, and the *value* is the "sound" (i.e., the sequence of *unisounds*). The wordToSound Map is an instance of java.util.HashMap, it is already declared as an attribute of class SimilarSounds.

    - public static void populateSoundGroupToSimilarWordsMap(List<String> lines): Given a list of all entries in ***word_to_sound.txt***, this method populates the soundGroupToSimilarWords Map as follows: the *key* is the *sound-group*, and the *value* is a BST containing all the words that share that same *sound-group*. The soundGroupToSimilarWords Map is an instance of java.util.HashMap, it is already declared as an attribute of class SimilarSounds.

    - public static void findSimilarWordsInList(String words[]): Given a list of words, e.g., [word1, word2, word3, word4], this method checks whether word1 is similar to word2, word3, and word4. Then checks

whether word2 is similar to word3 and word4, and finally whether word3 is similar to word4. The behavior/output is described in the "Deliverable" section.

- public static void findSimilarWordsTo(String theWord): Given theWord this method prints all similarly sounding words in ascending order (including theWord). The output is described in the "Deliverable" section.

Phase2

As mentioned earlier, in this phase you only need to implement LinearProbingMap and invoke SimilarSoundsMyHM (provided) instead of SimilarSounds. In fact, most methods in LinearProbingMap are already implemented except for:

- public V get(Object key) : this method returns the *value* associated with *key*, or *null* if no mapping exists for *key*.

- public V put(K key, V value): this method associates the specified *value* with the specified *key*. If the map previously contained a mapping for the key, the old value is replaced by the specified value.

The behavior should be identical to the behavior of the get() and put() methods in java.util.HashMap.

Testing

Use *run.bat* and *runMyHM.bat* to test Phase1 and Phase2, respectively. The expected output for both must be the same. For example, "java SimilarSounds dimension" and "java SimilarSoundsMyHM dimension" should both output:

```
Words similar to "dimension": "ASCENSION" "ATTENTION" "CONTENTION" "CONVENTION" "DECLENSION" "DETENTION" "DIMENSION"
"DISSENSION" "EXTENSION" "GENTIAN" "HENSCHEN" "LAURENTIAN" "MENTION" "PENSION" "PRETENSION" "PREVENTION"
"RETENTION" "SUSPENSION" "TENSION"
```

How To Handle a Multi-Week Project

While this project is given to you to work on over several weeks, you are unlikely to be able to complete it in one weekend. We recommend that you start working on Phase1 ASAP and finish it by no later than March 27. Note that all material related to Phase1 has already been covered in class prior to the midterm. The material for Phase2 will be covered in full by March 24, leaving you with ten days to finalize your project.

Requirements Summary

An overview of the requirements are listed below, please see the grading rubric for more details.

- **Implementing the classes** - You will need to implement required classes and fill the provided template files.
- **JavaDocs and Style** - You are required to write JavaDoc comments for all the required methods. In many cases, the JavaDoc comments are already partially written, you should complete them, if appropriate**.** Style requirements are the same as other projects. (Make sure to run the JavaDoc and Style checkers).