CS102 – Algorithms and Programming II
Programming Assignment 2
Spring 2025

ATTENTION:

- Compress all of the Java program source files (.java) files into a single zip file.
- The name of the zip file should follow the below convention, where you replace the variables “Sec1”, “Surname” and “Name” with your actual section, surname and name:

  CS102_Sec1_Asgn2_Surname_Name.zip
- Upload the above zip file to Moodle by the deadline (if not significant points will be taken off). You will get a chance to update and improve your solution by consulting to the TAs and tutors during the lab. You have to make the preliminary submission before the lab day. After the TA checks your work in the lab, you will make your final submission. Even if your code does not change in between these two versions, you should make two submissions for each assignment.

GRADING WARNING:

- Please read the grading criteria provided on Moodle. The work must be done individually. Code sharing is strictly forbidden. We are using sophisticated tools to check the code similarities. The Honor Code specifies what you can and cannot do. Breaking the rules will result in disciplinary action.

Chain of Words

For this assignment, you will implement a console application that finds chains of words such that each consecutive word only differs in one character. This difference could be a change in one of the characters, a deletion, or an insertion.

For example, GROW and GLOW have one character change (R -> L), so they can be chained. We delete one character to change GLOW into LOW, so they can also be chained. Similarly, to change LOW into SLOW, we add one character. Thus, one possible chain is GROW - GLOW - LOW - SLOW. However, a change of more than one character is not allowed in consecutive words; for example, we cannot chain LOW and CLOWN as this requires two additions. Similarly, FOOD and NOD cannot be chained as we need one deletion and one character change.

You will read the possible English words using the given “words.txt” text file, which contains a word in each line, sorted alphabetically. You should implement a Word class to represent a word. A different class should read the text file to create Word objects using each line, here you also need an *ArrayList<Word> allWords* to keep the Word objects you create.

After reading all the words, you should find the possible chains each word can create. To this end, you may implement a *boolean canChain(Word otherWord)* method in your Word class to check if two words can be chained. Also, add a *void addToPossibleChains(Word chainWord)*

method to add a possible chain word for this word. You should keep these possible chains in an *ArrayList<Word> possibleChains* in the Word class.

For example, GLOW could have LOW, SLOW, and GROW in its *possibleChains.* Similarly, the word SLOW will have GLOW and LOW in its *possibleChains*, but not GROW, as it cannot form a chain with SLOW. Note that the given text file contains all lowercase letters, you should make them all uppercase as you construct Word objects (read -> READ).

While checking for word pairs for possible chains, you should call *canChain* method only for pairs of words with a length difference of less than 2. If the two words have a length difference of 2 or more, they cannot form a chain, so there is no need to use the *canChain* method. If a word A can be chained with a word B, the reverse is also true, so you should call *addToPossibleChains* method for both words. After checking all the word pairs, remove any words from the *allWords* ArrayList that cannot form a chain with any other word. In other words, you should remove words whose *possibleChains* length is zero.

Note that we did not include all the English words in the given text file, only words with lengths between 3 and 7 are included to keep the problem simple.

Reading from a text file requires using the Scanner with a File input. The file should be included in your project folder so your application can find the file. You may use the following code segment to read the text file:

```
String filename = "words.txt";
Scanner sc = new Scanner(new File(filename));
while(sc.hasNextLine()) {
  String line = sc.nextLine();
  System.out.println(line);
}
sc.close();
```

You will see that trying to open a file with “new File(filename)” results in a syntax error with the message indicating an unhandled exception. You will learn more about exceptions later in the course. For now, you should know that using a File can result in an exception if the specified file is not included in the project folder; therefore, you need to handle this situation. If you include the “throws FileNotFoundException” declaration to the method signatures as the IDE suggests, this will leave handling the exception to the classes that call this method. Alternatively, you may surround the code that performs the file reading with a try-catch block. Both approaches are accepted for the assignment.

Include an option to generate a chain for a user input string using the Word objects you have. The generated chain should have at most 10 words in it, and it should not have any duplicates. For example, LOW - SLOW - GROW - GLOW - LOW - LOG is not a valid chain since it contains LOW twice. Start with the user-typed word, then choose one random word in the

chains ArrayList of this word as the next word in the chain. You will continue the process using the next word until you reach 10 words or all the words in the *chains* ArrayList of the current word is already used in the chain.

For example, suppose the user enters TREE as input. You will find the Word object corresponding to TREE, and get a random word from its *possibleChains*. Let's say we get FREE as the next word. Then, we look at *possibleChains* of FREE, but the new word we choose should be different than TREE, which is already included in our chain. For example, the next word can be FRED. You may include a *Word getChainWord(ArrayList<Word> currentChain)* method in your Word class to pick a possible chain word that is not included in the current Chain. This method should return null if there is no possible chain word when we exclude the words in the current chain. If the user input string is not included in *allWords* ArrayList, you should indicate that with a message and ask for a different word. If a chain is found using the given input word, the program should print the chain and ask for a different user input.

You will realize that looking for the possible chains takes some time. You are free to save your *allWords* after removing the words that cannot make a chain, there should be around 12K such words. In this case, you may use Java's FileWriter class to write the words line by line to a different text file, such as "filtered_words.txt"; this file will contain words that can be chained with others. This will improve your runtime for demonstrating your work to the TA. You may also keep the *possibleChains* of each word in a different text file such as "chains.txt" so that you can load the chains of each word directly from the file. For this option, you may include the word as the first element of the line and any word in its *possibleChains* separated by space characters on the same line. Then, at runtime, you will read this text file and form the Word objects by splitting the lines based on the space character. Note that these two operations involving "filtered_words.txt" and "chains.txt" are optional but highly recommended.

To write into a text file, you may use the following code. Note that while writing to a file, you also need to indicate the line ends with \n symbol. File writer may also result in exceptions; therefore, you need to throw it or surround the related block with try-catch.

```
String filename = " filtered_words.txt";
FileWriter fw = new FileWriter(filename);
fw.write("test string"+ "\n");
fw.close();
```

You should also implement a simple game where you show a chain of three, hiding the middle word; then, the user tries to guess the middle word. The user has three chances to input the correct word; otherwise, the program will reveal the hidden word. For example, suppose the randomly selected chain of three is HEAL-HEAR-EAR. The program will print it as "HEAL - ? - EAR", and the user has to guess the middle word by typing the possible answers.

Include a menu so the user can select whether to play the guessing game or generate chains with the user-typed words. Include options to return to the menu after playing the game or

generating a chain. You can include additional classes, variables, and methods to support the required functionality.

Preliminary Submission: You will submit an early version of your solution before the final submission. This version should at least include the following:

- Parts to read the input text file to find the possible word chains should be completed.

You will have time to complete your solution after you submit your preliminary solution. You can consult the TAs and tutors during the lab. Do not forget to make your final submission at the end. Even if you finish the assignment in the preliminary submission, you should submit for the final submission on Moodle.

Not completing the preliminary submission on time results in 50% reduction of this assignment's final grade.