CSC110 Prog02: Words game **75 Points**

Write an application named, **Wordsa.java**. This application will be a word guessing game reminiscent of the traditional, Hangman, but no drawing of a hanging stick figure will be produced. The application will display a welcome message and then be capable of playing as many games as the user wishes in one execution run of the program. A new random word must be chosen for each game played. The user will indicate that another game is to be played by answering 'y' or 'Y' in response to the question, "Want to play again?" asked by the program after each game. *See the execution logs on Canvas*.

The secret word for each game must be randomly chosen from the file, `wordlist.txt`, found on Canvas under this assignment. This file contains **33,736** words. You may use this numeric value (33736) as a *named constant* and use this constant in generating a random integer. Then, read each word from the beginning of the file until you read the ***n*th** word, where ***n*** is the random number generated between 1 and 33,736, inclusive. For *example*, if the random number is 20,751, your code should read and ignore 20,750 words, then read the word, `nonfat`, as the secret word (number 20751) for the current game. [NOTE: After we cover chapter 7, we could use arrays to choose a word, but for now the word **must** be chosen as indicated in this paragraph.]

Once a secret word is chosen the program displays this to the user in a disguised format. The disguised word contains a dash ('−') in the corresponding position of each consonant in the word and an equal sign ('=') for each vowel. All the logs show this clearly. The disguised word therefore shows the user the length of the word, as well as the actual position of each consonant and vowel within the secret word. The letter, Y, is considered a consonant for this application.

The user begins by guessing one letter at a time. If the guessed letter is in the secret word, a new disguised word is generated showing all occurrences of the guessed letter along with dashes and equal signs for letters not yet guessed. If the guessed letter is not in the word, the incorrect guess is concatenated to the string of incorrect guesses, but only if the guessed letter is not already in this string. A correct guess and the first time an incorrect letter is guessed cause the guess count to be incremented. If a guessed letter has already been guessed either correctly or incorrectly, the guess count is NOT incremented. You can observe this in the logs and thereby get a full understanding of how this must work. Just before the prompt for each guess, which shows the disguised word, the application displays the string of letters that have already been incorrectly guessed. Note that each incorrectly guessed letter appears in the incorrect guesses string exactly once. *See the execution logs on Canvas*.

If the user guesses a character that is not a letter, a validation loop should be entered to print an error message and a "try again" prompt for another guess until the user enters a letter as the guess. You can observe this in the example execution logs. Make your program behave as you see in these logs.

When the user has guessed the final letter to complete the guessing of the secret word, the game loop should be stopped and the game-ending message displayed. For example:

```
You guessed the word:  ready in 7 guesses!
Your score was 92.3%!
```

This must be displayed *AFTER the end* of the game loop, *not inside* this loop. The guesses include all correct and incorrect letters guessed. Repeated guesses (correct or incorrect) and non-letters are not counted as guesses.

Scoring: Here is how the score is calculated. The score is the **sum** of two values: the letters percentage and the word length percentage. The letters percentage is determined by the expression:

```
100 – guessCount / 26.0 * 100
```

where `guessCount` is the number of guesses made by the user during the game just played.
The word length percentage is determined by the expression:

```
<Length of the secret word> / 26.0 * 100
```

This score should be printed to show exactly one decimal place. Following the game-ending display, the program should ask the user whether to play another game. *See the execution logs on Canvas*.

When the user answers with any character other than 'y' or 'Y', the application termination message is displayed. This includes the "Thanks for playing…" message followed by the best (maximum) score of all the user's games played during the session, followed by the user's average score. You should display the application termination message as described here in the exact format found in the execution logs. *See the execution logs on Canvas*.

Thoroughly read through the execution log below, but especially go to Canvas and carefully read through all the logs given there. These logs demonstrate all of the requirements whether these details are mentioned above or not. [Note: you should NEVER reproduce the lines in the output produced by jGrasp.] If you have any questions, ask your instructor. However, questions that are not asked until 1 or 2 days before the due date may NOT have time to be answered.

One example execution log follows. Several more such logs can be found on Canvas. Your program should look exactly like these logs, except that the secret words will be randomly chosen, so they would more than likely be different than those shown below. Also, the letters guessed would vary according to whatever the user chooses to guess. The point of these examples is that your program should have the same look and wording, the same responses to user input, and the same results as those shown below. If some detail is not mentioned in the requirements, these logs are meant to act as requirements to be followed for completing your application.

```
 ----jGRASP exec: java Prog02 ← DO NOT make your code print this line
Welcome to Words: The Word Guessing Game!
Play as many games as you like. I'll remember your top score.
and also compute your average for all games played.

- - - - - - - - - - - - - - - - - - - - -
Letters already guessed:
Guess a letter in this word: ----== ?? a
Letters already guessed: a
Guess a letter in this word: ----== ?? e
Letters already guessed: a
Guess a letter in this word: --=e== ?? i
Letters already guessed: a
Guess a letter in this word: --ie== ?? s
Letters already guessed: a
Guess a letter in this word: s-ies- ?? t
Letters already guessed: a
Guess a letter in this word: s-iest ?? h
Letters already guessed: ah
Guess a letter in this word: s-iest ?? ;
Not a letter. Guess again: .
Not a letter. Guess again: a
==> a was already guessed.
Letters already guessed: ah
Guess a letter in this word: s-iest ?? s
==> s was already guessed correctly.
Letters already guessed: ah
Guess a letter in this word: s-iest ?? l

You guessed this word: sliest in 7 guesses!
Your score: 96.2%!

Want to play again? n
Thanks for playing...
        Your best score was 96.15%!
        Your average score was 96.15%!

 ----jGRASP: operation complete.  ← DO NOT make your code print this line
```

Be sure to study the execution logs provided on Canvas also as a key part of understanding all the requirements for this program!

REMEMBER: Do not use any Java feature not already covered in the course *so far*. Avoid redundancy. Use named constants. Use descriptive names for variables. Use the proper loop construct for the job. Example: at least one game is always played: that's a job for the do..while loop! Format your code properly so that your submission is not rejected. As a guide, a solution to this assignment with a proper prologue is typically about 140 lines of code or less.

Do not forget that the prologue is essential to explain to a *non-programmer* the details involved in the application.

Ask your instructor questions *early in the assignment cycle*. Questions asked in the last few hours before the due date/time might not receive an answer.