CPSC 1150 – Program Design

Assignment #5

Total Marks: 54 marks (4 marks are bonus marks)

Goals

This assignment provides students more practice on:

- Problem solving especially with methods, recursions, Arrays, Strings, and command line arguments.
- Presenting algorithms using flowchart or pseudocode
- Testing algorithms
- Writing Java programs

Problem Description

Exercise 1: Use the game (Secret Phrase game) that you have developed in Assignment 4 to complete the following parts:

Part A [12 marks]: The game in assignment 4 has limited appeal because each game contained only a few possible phrases to guess; after playing the games a few times, the user would have memorized all the phrases. Now modify your code by adding new method(s) in which any number of secret phrases can be saved to a text file before the game is played. Use a text editor such as Notepad to type any number of phrases into a file, one per line. Save the file as a text file such as **Phrases.txt**. Then, create a game that randomly selects a phrase from the file and allows the user to guess the phrase letter by letter.

Part B [10 marks]: Modify the above game so that the game is composed of more than one rounds (number of rounds can be defined and set as a constant variable in the program). In each round (like part A) the program asks user to guess a new phrase. It keeps the score of users for each round which is calculated by the length of phrase divided by the number of user's guesses. When all rounds are done, the program must show the average score and a tabular report of all rounds and its related score and phrase.

For example, the partial output of a run of 5 rounds is as follows: (Using GUI is optional)

```
Round  Target Phrase                Score
    1  The Wizard of Oz             1.23
    2  Gone With The Wind           1.64
    3  Casablanca                   1.25
    4  Chicago                      0.88
    5  Top Hat                      1.17
The average score is 1.23
```

Part C [12 marks]: Use command line arguments to make your code flexible for running different parts as shown in the following main method.

```
public static void main(String[] args) {
    if (args.length == 2 || args.length == 3) {
          final int ROUNDS = Integer.parseInt(args[0]);
          double [] scores = new double [ROUNDS];
          String [] targets = selectPhrases(ROUNDS, args);
          for (int i=0; i < ROUNDS; ++i) {
             scores[i] = runARound(targets[i]);
          }
          reportResults(targets, scores);
    } else {
       System.out.println("Usage: java SercetPhrase rounds [-l | -f filename]");
       System.out.println("rounds: a positive integer that represents the
                                        number of rounds for running program");
       System.out.println("-l: randomly selects the targets from a list of phrases");
       System.out.println("-f filename: randomly selects the targets from the filename");
       System.exit(1);
  }
}
```

Using `java SercetPhrase 1 -l` command will execute exercise 2 in Assignment 4.
Using `java SercetPhrase 1 -f filename` command will execute part A of this assignment.
Using `java SercetPhrase 10 -l` command will execute 10 rounds of exercise 2 in Assignment 4 in 10 and finally reports the scores detail and average score (like what is shown in part 2).
However, if user runs `java SercetPhrase 5 -f filename` command, the program executes 5 rounds of game where it selects its random phrases from a file named as `filename`. (Part B)
Finally, if user tries to run the program with no argument like `java SercetPhrase` the program must display the usage instruction and ends.

Assume that user passes valid input as arguments of the program. (no input validation is needed)
Use overloading functions to have the same task with different type of parameters so that fulfills the requirements of different parts. This makes it easy to modify your code.

Exercise 2: [10 marks] Partition of a list - Write a method that partitions the list using the first element, called pivot.

```
public static int partition(int[] list)
```

After partition, the elements in the list are rearranged so that all the elements before the pivot are less than or equal to the pivot and the elements after the pivot are greater than the pivot. The method returns the index where the pivot is in the new state of the list. For example, suppose the list is [5, 2, 9, 3, 6, 8]. After the partition, the list becomes [3, 2, 5, 9, 6, 8]. We do not need to sort the whole list, so think of a more efficient algorithm to find the index. The order of values before and after pivot is not important.

Exercise 3: [10 marks] Recursive method - Write a recursive method that given a string, it returns the reversed string using the following header:

```
public static String reverseDisplay(String value)
```

for example, `reverseDisplay("abcd")` returns `"dcba"`.

For Exercise 2 and 3 you need to only submit a function. Use **answers.pdf** to submit your functions along with your external documentations.

Grading

The programming questions are marked based on correctness, style of program and documentations. Add the external documentations in **answers.pdf**.

Submission

Submit a zip file (**Firstname-Surname-StudentId.zip)** including **answers.pdf, and SecretPhrase.java** on Brightspace.