CS603 Programming Assignment 3

Due by end of day on **Wednesday, 10/11**

1. Getting Started

In this assignment, you will be writing a game that makes use of arrays. A class called **RandomLetters** has been posted along with this assignment. You will be using a method in this class for generating an array containing four letters (as char data) chosen from the letters ranging from 'a' to 'f'. Download **RandomLetters.java** to the **same folder** as your homework assignment code. You may need to **refresh the folder** for the file to appear in Eclipse: select the folder and either press the F5 function key or select File > Refresh from the menu. **Do not make any changes to the RandomLetters class and do not pass it in**. Only submit the java program you are writing.

2. Programming Assignment

Mastermind (28 points)

The Mastermind game involves Player A guessing the color and sequence of four pegs arranged by Player B and hidden from Player A's view. Player B provides feedback on the number of pegs whose color and location were guessed correctly (i.e., "exact matches"), and the number of pegs whose color was correctly guessed but whose location was incorrect (i.e., "partial matches"). The game ends when Player A has either correctly guessed the color and location of each peg or has used up the number of allowed guesses.

In the version you will be writing, your program (using a method provided for you in another class) will generate a sequence of four letters, where each letter is in the range from 'a' to 'f.' The player will have up to **10** chances to guess the correct sequence, with your program providing feedback after each guess. The game will end when the player has either guessed correctly or has used up all 10 guesses.

In the following sample interactions, the player's input appears in **bold**. Instructions on how to play the game are provided once, at the start of a round of play. The player is then asked for each guess by number (Guess 1:, Guess 2:, etc.). Note that the player enters the letters in **one line with one space between each letter**.

Sample 1: player wins – correct sequence "**e d a b**"

```
You have up to 10 chances to guess a sequence of four letters ranging from a to f.

Enter four letters, separated by a space.

Guess 1: d e a b
exact (correct letter and position): 2
partial (correct letter, incorrect position): 2

Guess 2: e d a b

You won!
```

A letter may appear more than once in the generated sequence, and the player can repeat letters in a guess. For repeated guesses, an exact match will take precedence over a partial match. For example, if the player guesses that the letter 'a' appears twice but is in the sequence only once, and if one of the guesses is an exact match, then that guess will be the one that counts (see sample 2).

Sample 2: player wins – correct sequence “**a b c d”** (note repeated letters in the guesses)

```
You have up to 10 chances to guess a sequence of four letters ranging from a to f.

Enter four letters, separated by a space.

Guess 1: e b b f
exact (correct letter and position): 1
partial (correct letter, incorrect position): 0

Guess 2: b e f b
exact (correct letter and position): 0
partial (correct letter, incorrect position): 1

Guess 3: e a a f
exact (correct letter and position): 0
partial (correct letter, incorrect position): 1

Guess 4: c e c a
exact (correct letter and position): 1
partial (correct letter, incorrect position): 1

Guess 5: a b f c
exact (correct letter and position): 2
partial (correct letter, incorrect position): 1

Guess 6: a b c d

You won!
```

If the player **wins**, there is **no final feedback** concerning exact and partial matches, just a message stating “**You won!**” as shown in the prior two samples. If the player **loses**, the **exact and partial matches are shown**, and the **correct sequence** is displayed, as shown below.

Sample 3: player loses - correct sequence “**a d d e**” (note repeated letters in both the sequence and the guesses)

```
You have up to 10 chances to guess a sequence of four letters ranging from a to f.

Enter four letters, separated by a space.

Guess 1: a b c d
exact (correct letter and position): 1
partial (correct letter, incorrect position): 1

Guess 2: a a a a
exact (correct letter and position): 1
partial (correct letter, incorrect position): 0

Guess 3: a a b b
exact (correct letter and position): 1
partial (correct letter, incorrect position): 0

Guess 4: a c c c
exact (correct letter and position): 1
partial (correct letter, incorrect position): 0
```

```
Guess 5: a d e f
exact (correct letter and position): 2
partial (correct letter, incorrect position): 1

Guess 6: a d f f
exact (correct letter and position): 2
partial (correct letter, incorrect position): 0

Guess 7: a d e d
exact (correct letter and position): 2
partial (correct letter, incorrect position): 2

Guess 8: a e d d
exact (correct letter and position): 2
partial (correct letter, incorrect position): 2

Guess 9: d a e d
exact (correct letter and position): 0
partial (correct letter, incorrect position): 4

Guess 10: d e a d
exact (correct letter and position): 0
partial (correct letter, incorrect position): 4

You lost. The correct code was: a d d e
```

Code requirements:

1. This assignment is about arrays, which must therefore be used **at a minumum** for storing the letters to be guessed as well as the letters the player guesses. Store the **correct sequence** and the **guessed sequence** to arrays of type char and **use those arrays in determining exact and partial matches**. Note that representing those sequences as String objects and using String methods or as List objects and using list methods for finding exact and partial matches are not acceptable solutions.
2. Initially, hardcode the sequence of letters for the player to guess. For example:

   ```
   char[] correctSequence = {'a', 'e', 'b', 'd'};
   ```

   This will make it possible to test your logic with a known sequence that doesn't change (until you want it to).

   Once your logic is working, use the static `generateSequence()` method in the `RandomLetters` class, which returns an array of type char containing four lowercase letters between 'a' and 'f.' Store the value returned to your array (correctSequence in the case above).

   When testing with a generated sequence, it is recommented that your program **prints the contents of the array that was returned** by the method so you can test your code with known values. **For your final version, comment out any code you added to print the generated sequence**.
3. The player is allowed a maximum of **10** incorrect guesses. Store this value to a named **constant**. For testing purposes, you may want to allow a smaller number of guesses, but be sure to set the value to 10 for final testing and submission.
4. After prompting for four letters, you can use **next().charAt(0)** within a loop to read in one character for each loop iteration. Store **the letters guessed by the player** to an **array**.

5. Display the number of **exact** and **partial** matches **except for when the user wins** (in which case there must be four exact matches). If the **player loses**, show the **number of exact and partial matches as well as the correct sequence**.

Additional requirements:

You can define as many static methods as you would like, but the following two are required:

1. Define a **static method** called **arrayToLowerCase** that is passed an array of any length of type char and replaces any uppercase characters in that array with the corresponding lowercase letters. For example, if passed an array containing {'a', 'B', 'T', 'g', 'D' , 'X'}, the contents of that array should be updated to {'a', 'b', 't', 'g', 'd' , 'x'}. The return type of this method should be **void**, as the contents of the array passed to the method will be changed so no need to explicitly return it. See section 9.5 of the textbook on static methods defined in the `Character` class for help with this functionality. Modify your code to make use of this method for converting the player's guess to lowercase values.
2. Define a **static method** called **verifyLetters** that is passed (1) an array of any length of type char in which all the letters are in lowercase, (2) a starting character , and (3) an ending character. It returns a boolean value of true if every character in the array is between the starting and the ending values, inclusive, or a value of false otherwise. While you will be using this method only for verifying that input falls between one particular range (i.e., a through f) , it must be written as a more **general purpose method** that would work for any range of letters. You may assume that all character data passed to this method are in lowercase.

   For example, if the arguments passed to the method are (['c', 'b', 'v', 'd', 'e', 'h'], 'b', 'h'), then a value of false would be returned, because the character 'v' is not within the range of letters from 'b' to 'h.'

   Guesses with invalid input should **not be counted against the player**, as shown in the following output. If invalid input is entered, the player must be **prompted again** until only valid characters are entered.

Sample 4: validating input

```
You have up to 10 chances to guess a sequence of four letters ranging from a to f.

Enter four letters, separated by a space.

Guess 1: a t b c
*** Enter only letters between 'a' and 'f' ***

Guess 1: E E E E
exact (correct letter and position): 1
partial (correct letter, incorrect position): 0

Guess 2: e 1 1 1
*** Enter only letters between 'a' and 'f' ***

Guess 2: e g g g
*** Enter only letters between 'a' and 'f' ***

Guess 2: E c f c

You won!
```

3. Grading

Your program must compile and run to receive any credit. You can receive partial credit if a portion of your program is working, but only if the program compiles without syntax errors. The grade for this assignment will be determined on the following basis:

- 6 points: two required static methods.
- 4 points: Prompting for input, including showing the number of the guess, and re-prompting if invalid data was entered (using the static methods as part of the data validation process).
- 12 points: determining the number of exact and partial matches for each guess.
- 4 points: determining when the game is over and displaying the required output.
- 2 points: meeting the style requirements, including the use of comments, meaningful identifiers, and constant values when appropriate, and having the program appearance reflect its logical structure.

4. Submitting Your Assignment

When you are satisfied that your program meets all requirements, submit your **java** file electronically by following the submit link from the Blackboard Assignments page for Assignment 3. You **should not be submitting SecretCode.java** or any other files. Please remember to comment out a package statement if one appears in your code.