CS2110 Homework 8 - Fall 2021
Recursion

You will not need code from a previous assignment to complete this homework. You will be solving several problems concerning password verification using a recursive approach.

First, set up your project:

- Create a new Java project on Eclipse, and call it "Homework 8."

Learning Goals

In this assignment, we will practice:

1. Using a recursive approach to solve problems,
2. Main method testing.

Implementing Recursive Methods

For this assignment, you will be helping the IT department ensure that users choose properly secure passwords. However, they've asked you to apply the technique of recursion to verify passwords.

Create a class named `PasswordChanger` that provides several methods for verifying passwords as defined below. **Your method signatures should exactly match ours.** You should download and implement the `PWCInterface` interface to ensure that you have all the required methods.

- `public boolean containsDigit(String password)`
  - Write this **recursive** method that determines if the password contains at least one character that is a digit (`0` through 9). If it does, return `true`; else return `false`.
- `public boolean equalsOld(String oldPassword, String newPassword)`
  - Write this **recursive** method that determines if the new password is equal to the old password. If they are the same, return `true`; else return `false`.
- `public boolean reverseOfOld(String oldPassword, String newPassword)`
  - Write this **recursive** method that determines if the new password is equal to the reverse of the old password. If they are the same, return `true`; else return `false`.
  - For example, `reverseOfOld("password", "drowssap")` would return `true`.

- `public int numberDifferences(String oldPassword, String newPassword)`
    - Write this **recursive** method that returns the number of character positions where the two passwords differ. Note that the passwords may be of differing lengths!
    - For example:
        - `numberDifferences("abc", "abd")` returns 1
        - `numberDifferences("password", "passw0Rd")` returns 2
        - `numberDifferences("pass", "paSsword5")` returns 6
            - *There are 5 extra characters on the end of the new password that are not in the original.*
- `public boolean differentEnough(String oldPassword, String newPassword, int quality)`
    - Write this **NON-recursive** method that determines if the new password differs from the old password by at least `quality` number of characters. If so, return true; else return false.
    - *You should call another method you have already written as part of your implementation.*
- `public boolean validPasswordChange(String oldPassword, String newPassword)`
    - Finally, write this **NON-recursive** method that will determine if the user should be allowed to change the old password to the new password. A password change is allowed if the new password contains at least one digit, it is not the old password or the reverse of the old password, and it is different from the old password in at least half of the character positions. Return `true` if this password change is valid, `false` otherwise.
    - *Remember that the passwords may be of different lengths!*
    - *You should use your other methods above to determine if the password change is valid. You should **NOT** use String's methods such as `.equals()` to implement this method.*

Note: For each of these methods, an iterative solution exists. However, you will get **no points** for an iterative solution if we've asked for a recursive solution. A solution is recursive because at some point, the recursive method *will* call itself. (*This is a clue!* If your method does not call itself at some point, you may have taken an iterative approach. Be sure to collaborate on strategy with your cohort and get feedback.)

Main Method (or JUnit) Testing

In this homework, we expect you to do your own testing before submitting to Gradescope.

Note: Your main method should have enough testing to provide sufficient evidence to determine that the behavior implemented matches the behavior described above. It is *highly recommended* to write the tests before implementation as it will help you to understand the exact behavior and what is the expected output of different inputs.

*Our goal in this assignment is to encourage main method testing so that you know your code works well before submitting to Gradescope. Therefore, you will only be able to submit to Gradescope a total of **10 times** for this assignment.*

Submission Information

Method and Class Naming: You must match method names, instance variable names, and data types exactly. You must use correctly formatted Java code. Declare fields in the class definition, and create a default constructor for each class that initializes every instance variable. *Implement our provided interface to guarantee your code matches.*

Coding Style: In real-world software development, it is paramount to create readable and easily maintainable code. That is typically achieved through the use of style and commenting guidelines. Since you will be updating this code over the next few weeks, we have provided a style guide and formatting guide that we strongly encourage you to follow:

- Coding Style Guide (includes installation instructions for Eclipse)
- Eclipse Style File

Submitting: Upload your Eclipse project (the `.java` file) to the "Homework 8 - Recursion" assignment on Gradescope. You should submit `PasswordChanger.java`. This submission utilizes an autograder to check that your code follows these specifications. If it spots an issue, it will alert you, but you should **NOT** use the submission system as your testing. We encourage you to test your code during the implementation phase. Therefore, you may upload your code a **maximum of ten (10) times**.

Note: After the 10th submission, Gradescope will still allow submissions, but they will NOT be graded by the system.

Grading Rubric

The assignment will be worth a total of 100 points:

85 points - Method and implementation correctness, auto-graded using Gradescope

15 points - Code readability (organized, well-indented, readable by others)