CSCI 3400
Fall 2023
Lab 02: Composition, Delegation, and the Some Principles
Assigned date: 09/07/2023
Due date: 09/14/2023 (11:59 PM, EST)
Total points: 45

Learning Goal: In this lab, we practice the SOLID principles.

Given Source Code:

2 packages:
students.task01 and
students.task02

Tasks:

[10 points] Task 01: Composition, Delegation, and Inheritance

Test your implementation as stated in the main method of the **students.task01** package**.**

Problem Description:

We want to extend the functionality of a String class via a new class (say, `FunString`) so that it has the following methods:

```
public int countVowels()
//Counts the total number of vowels in a String.

public int countDigits()
//Counts a String's total number of digits (0~9).
```

Depending on the requirements, you may change the signature of these methods, but it may not be necessary. Try the following approaches, and at the end of your source file (TestMain.java, task01), **add comments** about which version works and why.

1. Inheritance:

```
class FunString extends String{
   … … …
   // implement countDigits() and countVowels() method here
   }
```

2. Composition:

```
class FunString {

String inputString;
… … …
// implement countDigits() and countVowels() method here

}
```

If you are using the composition approach, you may wonder that fn.length() is not working. As a comment at the end of TestMain, **explain why it is not working**. **Fix the issue so that fn.length() works.**

Hint: How to make `fn.length()` work:
delegate the `length()` method from the String class.

[35 points] Task 02: Are we following SOLID patterns?

Problem Description:

Test your implementation as stated in the main method of the **students.**task02 package.

a) **[5 points]** I claim that we are violating the OCP (Open-Closed Principle) in the BurgerShopA class. What do you think? Clarify your stand with a proper explanation. [Add your answer to this question as multiline comments at the end of the source code.]

b) **[10 points]** Please define a class "BurgerShopB" that implements the ElectronicOrderBook and follows the stated constraints:
   i) The burger shop doesn't provide the "orderCombo(…)" option/functionality.
   ii) The price of each burger is 8.5 dollars, and each pack of fries is 3.0 dollars.
   iii) The shop doesn't offer fries of various sizes.

c) **[10 points]** After/While implementing the BurgerShopB class, your teammate argues that the provided code is violating the ISP (interface segregation principle). Do you agree? If so, please revise the given code and adjust it to no longer violate the ISP principle. You may callyour new BurgerShopBModified.

   Note: You may create new interfaces, change the signatures and/or definitions of the methods, etc. While you use the new interfaces and redefine the BurgerShopB and BurgerShopA based on the updates, please comment out the previous code you wrote as part of part b and place them at the end of the source code.

d) **[10 points]** Find another SOLID principle that BurgerShopA or/and BurgerShopB is/are violating. Clarify your stand. [Add your answer to this question as multiline comments at the end of the source code.

Submission:

Add your name after your instructor's name and add citations (any resource you have used while writing code, for example, any website you have looked for). Submit the source codes.