CSCI 3400
Fall 2021
Lab 02: Composition, Delegation, and the Decorator Pattern
Assigned date: 09/01/2021
Due date: 09/06/2021 (11:59 PM, EST)
Total points: 60

Learning Goal: In this lab, we practice the decorator pattern.

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 so that it has the following methods:

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

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

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.

[50 points] Task 02: The Decorator Pattern

This problem is inspired by Reference [1]. Use **package *students.task02*** for implementation.

Problem Description:

The word has spread about your successful implementation of the order management system (OMS) for Starbuzz Inc. You received an email from the manager of TastyPizza Inc. who heard about your success. He wants you to implement a new OMS for his pizza delivery shop. For the first milestone, he has defined several requirements that need to be fulfilled by your OMS:

• The shop offers Pizza.
• All pizzas have a price and have specific nutrition facts.
• Every Pizza contains at least cheese and tomato sauce (Pizza Margherita). It can be combined with additional toppings (ham, onions, etc.), ordered multiple times.
• Pizzas with certain topping combinations are named, e.g., Hawaiian Pizza for the base pizza with ham and pineapple. Those names should be contained in the bill. You do not need to auto-detect names for such combinations if they are not explicitly requested in the order.
• In addition to the normal size, Pizza can be ordered in family size.
The manager sent you the menu card of TastyPizza for a better understanding of their available items:

Pizza	**Calories**	**Price**
Pizza Margherita (tomato, cheese)	1104	4.99
Hawaiian Pizza (tomato, cheese, ham, pineapple)	1024	6.49
Salami Pizza (tomato, cheese, salami)	1160	5.99
Family Size for Pizza	x1.95	+ 4.15

Toppings	**Calories**	**Price**
Cheese	92	0.69
Ham	35	0.99
Onions	22	0.69
Pineapple	24	0.79
Salami	86	0.99

Your tasks for this exercise are:

• Implement an OMS library in Java for TastyPizza, Inc that provides the requested features.
• Your library design should use immutable objects, i.e., orders and pizzas are never changed in place. For example, adding a topping to a pizza creates a new pizza object.
• Use the Decorator Pattern to implement the different variations of Pizza (additional toppings, family-sized).
• Create the pizza instances mentioned in the main(…) method of task02 package.

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.

Reference:

1. http://stg-tud.github.io/sedc/Lecture/ws16-17/exercises/ex07/ex07.pdfhttps://www.javatpoint.com/decorator-pattern