CSC148 Summer 2018: Lab 2

Introduction

The goals of this lab are:

- To get you familiar with composition
- To get you familiar with inheritance
- To give you practice designing a superclass.
- To give you practice implementing a subclass.

Don't hesitate to make use of other resources for this lab, including the course notes, your TAs, instructor, or other students.

General Lab Notes

1. Make sure you have lab_pyta.txt downloaded and placed in the directory (or directories) where you'll be working.
2. To use PythonTA, include the following code (if you already have a main block, just add the body to the end of it):

```
if __name__ == '__main__':
    import python_ta
    python_ta.check_all(config="lab_pyta.txt")
```

Your lab_pyta.txt should be in the same folder as the .py files you're running. PythonTA will raise errors regarding style, specifying the lines you need to fix. You should get familiar with what the errors mean, and how to fix them: this will be important for your exercises and assignments.

Designing classes

Below are the specifications for various classes you should design, as well as any subclasses or superclasses. You may organize your files however you'd like: I recommend having a separate file for each class, and a folder called **lab2** that holds all of these.

While there are multiple specifications below, you are not expected to implement all of them. At the very least, try to design and implement 1. The others are just for more practice.

Grade

Context: A grade represents a grade earned by a student in a course.

We can have 2 types of grade: A NumericGrade and a LetterGrade. Numeric grades can have a value from 0 to 100, while a NumericGrade has a letter value (e.g. "F", "D-", "D", "D+", ..., "A-", "A", "A+").

A Grade should be able to convert its value into a GPA format (e.g. a NumericGrade of 84 would be a 3.7, while a letter grade of A- would also be 3.7).

We're able to compare grades to each other. NumericGrades are equal to each other if they have the same numeric value, while LetterGrades are equal if they have the same value. You should also be able to compare 2 grades of different types by comparing the GPA equivalent.

A few things to consider:

- Grade should be a superclass of NumericGrade and LetterGrade.
    - It should have a method to convert its value into a GPA equivalent, but this should raise a NotImplementedError: the subclasses should be the ones to implement the behaviour.
        - Why should we put this method in Grade if we're not going to implement it?
- The actual conversions to GPA doesn't matter, just that you have a way to do so.
    - If you do want something that's completely accurate, see the ArtSci grading scale.

Write some client code that uses your NumericGrade and LetterGrade to test it. Additionally, try to run the following loop (replacing the constructor calls as needed):

```
list_of_grades = [NumericGrade(73), LetterGrade("A-"), LetterGrade("B-"), NumericGrade(72)]
grade_to_find = NumericGrade(72) # Create a grade that we want to find matches for
equivalent_grades = []
for grade in list_of_grades:
    if grade == grade_to_find:
        equivalent_grades.append(grade)
expected_grades = [list_of_grades[2], list_of_grades[3]]
assert equivalent_grades == expected_grades
```

You should be able to plug any grade into the list, and the comparisons should work.

Clothing (Part 1): Composition

Context: Clothes have a brand, a colour, and a price. People can purchase clothing, reducing the amount of money they have, as well as put clothing in their closet. From a closet, a person can take clothes out or look at the clothes they have by colour.

If a person tries to buy clothes without having enough money, you should raise a SadWalletError. If a person tries to take clothes out of a closet that doesn't belong, raise a NotInClosetError.

As this is practice in composition, you shouldn't be using inheritance in this part. Rather:

- A person should **have** clothes, and a person should **have** a closet. These should be attributes of a person.
- Closets **have** clothes.
- Clothes **have** a brand, colour and price.

Make classes for each of these, and implement methods for each of these classes.

Clothing (Part 2): Inheritance

Context: Continue adding to your code from Part 1. An Employee is a person who works in a Store. A store can hold a variety of clothes with certain prices, which People can purchase. Employees are also able to purchase clothes, but at a discount. In this universe, a store and a closet are both considered ClothingContainers. They both hold clothes, are both able to have clothes removed from them (although stores do so at a price) to be given to a Person, and can provide the clothes that they store which have a certain colour.

As this is practice in inheritance, you should be using inheritance here:

- An Employee **is** a Person.
- A Store **is** a ClothingContainer.
- A Closet **is** a ClothingContainer.
- ClothingContainers should have at least one method that's not implemented.