Homework 4: Office Supplies

For this assignment, you will be writing methods so that customers can successfully order and pay for office supplies from local office supply stores. You will also be writing and fixing test cases, so you can guarantee that every step within the checkout i.e., placing the order and the order being processed is accurate, and your customers are happy!

Review the starter code thoroughly before beginning this assignment, as understanding how the classes interact with each other is important. Take notes or draw a diagram if necessary.

Overview

Customer Class

The ***Customer*** class represents a customer who will order from the stores. Most of the class has been provided for you. You will write the ***place_item_order*** method.

Each *Customer* object has 3 instance variables:

name - a string representing a customer's name

employer_id - an integer representing the id of the store the customer works for (if they do not work for a store, the *employer_id* will be *None*)

account- a float showing how much money is in the customer's *account*. The default value is $15.

The *Customer* class also includes several methods:

__init__ - which initializes the customer's attributes

__str__ - which returns the customer's *name* and the amount in their *account* as a string.

deposit_money(self, amount) - which adds the passed *amount* to the customer's *account*

place_item_order(self, store, order) - which takes a store object and an 'order', which is a dictionary where the keys are **Item** objects and the values are another dictionary with keys of *quantity* and *express_order*. It returns a boolean value that indicates if the order was successfully placed or not.

Item Class

The ***Item*** class represents an item that a store can sell or that a customer can buy. The code for this class has been provided for you.

A *Item* object has 2 instance variables:

name - a string representing the name of item

cost - a float representing how much a *regular order* of an Item costs per quantity.

The *Item* class includes 2 methods:

__init__ - which initializes the *Item* object's name and cost

__str__ **-** which returns the name and cost as a string

Store Class

The ***Store*** class represents a store.. You will write several methods for this class. See the Tasks to Complete for which methods.

Each *Store* object has 4 instance variables:

name - a string which is the name of the store

store_id - an integer representing the id of the store (used for *customers* who are employees to track who they work for)

income - a float representing the income the store has collected

inventory - a dictionary which holds the item objects as the keys and the quantities in stock of each item as the values

The *Store* class also includes several methods:

__init__ - which initializes the attributes

__str__ - which returns a string with the store's name and the current menu

accept_payment(self, amount) - which takes an amount and adds it to the store's earning

calculate_item_cost(self, item, quantity, express_order, customer) - takes an *item* object, the *quantity*, whether a *express_order* has been requested, and returns the total cost of that combination

stock_up(self, item, quantity) **-** which adds the *quantity* of *item* to the inventory

process_order(self, order) - which takes a dictionary *order* where the keys are Item objects and the values are another dictionary with keys of *quantity* and *express_order*.

A few of the *Store* class and *Customer class* functions make use of an *order* dictionary. An example of an order dictionary is provided below:

```
notebook = Item('Notebook', 3.00)
pen_package = Item('Pen Package', 1.50)
pencil_package = Item('Pencil Package', 0.75)

order = {
    notebook: {
        "quantity": 3,
        "express_order": True,
    },
    pen_package: {
        "quantity": 2,
        "express_order": True,
    },
    pencil_package: {
        "quantity": 2,
        "express_order": False,
    },
}
```

Tasks to Complete

- **Complete the *Customer* Class**
  - Complete the ***place_item_order(self, store, order)*** method
    - Call the ***calculate_item_cost*** method on the *store* object to calculate the total cost of the order
    - Check if the customer has the total cost or more in their *account*. If they don't have enough money, immediately return **False**
    - Call the ***process_order*** method on the store object. If ***process_order*** returns **True**, remove the total cost from the customer's *account* and call the ***accept_payment*** method to add it to the store's income and return **True**. Otherwise, return **False**.

- **Complete the Store Class**

  - A ***calculate_item_cost(self, item, quantity, express_order, customer)*** method that takes the item object, quantity and express_order: a Boolean variable that specifies if the customer has requested an express order, and returns the total cost.
  Note: An express order of any item costs **20%** than a regular order

  - A ***stock_up(self, item, quantity)*** method that takes the item object and the quantity. It adds the quantity to the existing quantity if the item exists in the inventory dictionary or creates a new item in the inventory dictionary with the item name as the key and the quantity as the value.

  - A ***process_order(self, order)*** method that takes a dictionary that represents the order and checks that there is enough of each item in the inventory for the order and if not returns **False**. Otherwise, it subtracts the quantity of the items ordered from the quantity in the inventory and returns **True**.
  - An order should only be fulfilled if everything that is requested for it is present in the inventory.

- **Write and Fix Test Cases**

  Note: Many test cases have already been written for you. **Please do not edit any test cases other than the ones that have comments above them explicitly asking you to do so.** These test case-related tasks are also explained below:

  - Write test cases for the following scenarios for the ***place_item_order*** method in ***test_customer_place_item_order***:
    - The customer doesn't have enough money in their *account* to place the order
    - The store doesn't have enough of an item in stock (not enough in inventory)
    - The store doesn't sell the item mentioned in the order

  - Fix the test cases in ***test_customer_place_item_order_2***

  As you are working on one test case, feel free to comment out the test cases that you are not working on, but **be sure to uncomment all test cases before you turn in your homework.**

Grading Rubric (60 points)

Note that if you use hardcoding (specify expected values directly) in any of the methods by way of editing to get them to pass the test cases, or you edit any test cases other than the ones you have been directed to, you will NOT receive credit for those related portions.

- 10 points for correctly completing the ***Customer*** class function ***place_item_order***
- 30 points for correctly completing the ***Store*** class (10 points per method)
- 15 points for completing ***test_customer_place_item_order*** (5 points per scenario)
- 5 points for fixing ***test_customer_place_item_order_2***

Extra Credit (6 points)

To gain extra credit on this assignment, please edit *calculate_item_cost()* to follow this additional requirement.

Every *customer* that works at a *store* gets a 40% discount when purchasing from that store (this occurs when the customer's ***employer_id*** matches the store's ***store_id***). If the *item* is ordered as an *express order* then the employee *customer* still has to pay for the 20% upcharge before the discount is applied.