Homework 4:
¡Medicina Express para SI 206!

For this assignment, you will be writing and correcting methods so that customers can successfully order and pay for medicine delivery from a pharmacy, using the new service Medicina Express. You will also be writing and correcting test cases, so you can guarantee that every step from the order being taken to delivery 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 imperative. Take notes or draw a diagram if necessary. We cannot emphasize how important this step is.

Overview

Customer Class

The *Customer* class defines a customer. Each customer object has 2 instance variables: **name** (a string representing a customer's name) and **money** (a float showing how much money is in the customer's account). The *Customer* class also includes several methods: ***deposit_money*** (which adds a passed amount to the **money** in the customer's account), ***make_payment*** (which you will implement – see details below), ***order_medicine*** (which takes a **driver**, a **pharmacy,** the **drug_name,** and the **quantity** and places an order at that pharmacy to be delivered by that driver), and ***take_medicine*** (which simulates a customer taking the medicine by printing out "I am starting to feel better!").

Driver Class

The *Driver* class defines a delivery driver. Each driver object has 4 instance variables: **name** (string with the name of the driver), **money** (a float for the amount of money the driver has), **pharmacies** (a list of pharmacies objects the driver will deliver from), and **delivery_fee** (a float showing the fee for delivery), alongside several methods (see the code for details).

Pharmacy Class

The Pharmacy class defines a pharmacy. Each pharmacy object has 4 instance variables: **name** (a string which is the name of the pharmacy), **inventory (**a dictionary which holds the names of the medicine as the keys and the quantities of each medicine as the values) **money** (a float for the amount of money the pharmacy has) and **cost** (the cost to the customer for each pill). You will be in charge of implementing the Pharmacy class – see details below.

Tasks to Complete

- **Complete the Customer Class**
    - Complete the ***make_payment()*** method in the *Customer* class. This method takes a **driver** and an **amount** as parameters, and has the customer pay the driver the specified amount using the ***receive_payment*** method in the driver class (i.e., it deducts money from the customer's **money** and adds it to the driver). See the test cases under ***test_make_payment*** for clues on how this method should behave.

- **Create and implement the *Pharmacy* class with the following methods**
    - A ***constructor (__init__ method)*** that initializes the instance variables **name**, **inventory, cost** per pill (default = 10), and **money** (default = 500).
    - A ***process_order()*** method that takes in two values, the medicine name and the quantity. If the pharmacy has the medicine, it will decrease the quantity of that medicine in the **inventory** and increase the amount of **money** in the pharmacy by the **cost** times the quantity.
    - A ***has_medicine()*** method that takes in two values, the medicine name and the quantity and returns 'True' if there is enough medicine left in the inventory.
    - A ***stock_up()*** method that takes in two values, the **medicine name** and the **quantity**. It will add the quantity to the existing quantity if the medicine exists in the **inventory** dictionary or add the medicine and quantity to the **inventory** dictionary otherwise.
    - A ***__str__*** method that returns a string with the information in the instance variables using the format shown below:

      "Hello, we are [NAME]. These are the drugs that we currently have in stock [INVENTORY]. We charge $[COST] per pill. We have $[MONEY] in total."

      Expected output for printing a pharmacy object:

```
Hello, we are CVS. This are the drugs we currently have in stock {'Vicodin': 10, 'Xanax': 30}.
We charge $10 per pill. We have $500 in total.
```

- **Implement a *Main()* method**
    - Create at least two inventory dictionaries with at least 2 different types of medicine. The dictionary keys are the medicine names and the values are the quantity for each medicine.
    - Create at least 2 *Customer* objects. Each should have a unique **name** and **money**.
    - Create at least 2 *Pharmacy* objects. Each should have a unique **name**, **inventory, money, and cost.**
    - Create at least 2 *Driver* objects. Each should have a unique **name**, **money**, **pharmacies**, and **delivery_fee**.
    - Have each customer place at least one order (by calling ***order_medicine***).

- **Write and Correct Test Cases**
    - Note: Many test cases have already been written for you. **Please do not edit test cases outside of the ones below.** As you are working on one test case, feel free to comment out the test cases that you are not working on.
    - ***test_estimated_cost*** fails. What are the correct numbers to make this test pass? Correct the mistakes in this test.
    - Complete ***test_has_medicine,*** which tests the ***has_medicine*** method in the pharmacy class. We have provided 3 scenarios for you to test.
    - Complete ***test_order_medicine,*** which tests the ***order_medicine*** method in the *Customer* class. The ***order_medicine*** method places an order of medicine from a pharmacy to be delivered by a driver, but only if several conditions are met: if the customer has enough money to pay for the transaction, if the pharmacy has medicine in stock, and if the delivery driver can deliver from that pharmacy.

      When writing tests for ***test_order_medicine***, please write comments for each test case describing what scenarios you are testing, similar to the comments in ***test_has_medicine***

      Example output for this test case:

      Don't have enough money for that :(Please add more money to your account!
      Pharmacy has run out of [Medicine Name] :(Please try a different pharmacy!
      Sorry, this service doesn't deliver from that pharmacy. Please try a different pharmacy!

```
Testing if customer doesn't have enough money
Don't have enough money for that :( Please add more money to your account!
Testing if Pharmacy doesn't have medicine left in stock
Our pharmacy has run out of Vicodin :( Please try a different pharmacy!
Testing if driver doesn't deliver from that pharmacy
Sorry, this service doesn't deliver from that pharmacy. Please try a different pharmacy!
ok
```

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.

Note - use the provided methods you will earn credit if you implement the functionality instead.

-15 points for correctly implementing the **Pharmacy** class (3 points per method).
-5 points for correctly completing the ***make_payment*** method in the *Customer* class.
-5 points for creating the customer, driver, and pharmacy objects in the main method and correctly placing an order for both customers.
-5 points for correcting ***test_estimated_cost***
-15 points for writing non-trivial test methods for ***test_has_medicine*** (5 points per scenario correctly tested).
-15 points for writing non-trivial tests for ***test_order_medicine*** (at least three scenarios; 5 points per scenario correctly tested).

Extra Credit (6 points)

It turns out that whoever wrote this program forgot we are in the US, a land where everyone tips 20%. To gain extra credit on this assignment, do the following: (1) rewrite ***make_payment()*** to account for this 20% tip. The customer should pay an extra 20% tip to the Medicine Express driver IF the customer has enough money to afford it. (2) Rewrite the appropriate test cases to test this new behavior. Earn up to 3 points for each task (for 6 total).