Homework 4:
Food Truck Classes SI 206

For this assignment, you will be writing and correcting methods so that customers can successfully order and pay for food at a food truck festival using the new ordering system which has the server take your orders directly, who then passes on the orders to the trucks. 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 important. 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 who will order from the trucks. Each customer object has 2 instance variables: **name** (a string representing a customer's name) and **wallet** (a float showing how much money is in the customer's wallet). The *Customer* class also includes several methods: ***withdraw_money*** (which adds a passed amount to the **wallet**), ***make_order*** (which you will implement – see details below), **_init_**, ***order_food*** (which takes a **server**, a **truck,** the **food_name,** and the **quantity** and places an order at that truck to be delivered by that server), and ***eat_food***(which simulates a customer eating the food by printing out "Wow, that was tasty!").

Server Class

The *Server* class defines a server at the festival. Each server object has 4 instance variables: **name** (string with the name of the server), **money** (a float for the amount of money the server has), **food trucks** (a list of food trucks objects the server takes order for), and **service_fee** (a float showing the fee for delivery), alongside several methods (see the code for details).

Truck Class

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

Tasks to Complete

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

- **Create and implement the *Truck* class with the following methods**
    - A ***constructor (__init__ method)*** that initializes the instance variables **name**, **inventory, cost** per food (default = 7), and **money** (default = 700).
    - A ***process_order()*** method that takes in two values, the food name and the quantity. If the food truck has the food, it will decrease the quantity of that food in the **inventory** and increase the amount of **money** in the truck by the **cost** times the quantity.
    - A ***has_food()*** method that takes in two values, the food name and the quantity and returns 'True' if there is enough food left in the inventory.
    - A ***stock_up()*** method that takes in two values, the **food name** and the **quantity**. It will add the quantity to the existing quantity if the food exists in the **inventory** dictionary or add the food 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]. This is the current menu [INVENTORY KEYS AS LIST]. We charge $[COST] per item. We have $[MONEY] in total."

      Expected output for printing a truck object:

```
Hello, we are The Tamale Train. This is the current menu tacos, water.
We charge $10 per item. We have $700 in total.
```

- **Implement a *Main()* method**

- Create at least two inventory dictionaries with at least 3 different types of food. The dictionary keys are the food items and the values are the quantity for each item.
- Create at least 2 *Customer* objects. Each should have a unique **name** and unique amount of money in their **wallet**.
- Create at least 2 *Truck* objects. Each should have a unique **name**, **inventory, money, and cost.**
- Create at least 2 *Server* objects. Each should have a unique **name**, **money**, **food trucks**, and **service_fee**.
- Have each customer place at least one order (by calling ***order_food***).

● 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_food****,* which tests the ***has_food*** method in the *Truck* class. We have provided 3 scenarios for you to test.

- Complete ***test_order_food****,* which tests the ***order_food*** method in the *Customer* class. The ***order_food*** method places an order of food from a food truck to be delivered by a server, but only if several conditions are met: if the customer has enough money in their wallet to pay for the transaction, if the food truck has food in stock, and if the server serves that food truck.

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

  Example output for this test case:

  Don't have enough money for that :(Please withdraw more money!
  Our food truck has run out of [Food Item] :(Please try a different truck!
  Sorry, I don’t serve that food truck. Please try a different one!

- Complete **test_withdraw_money,** this tests if the customer can add money into their wallet.

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 **Truck** class (3 points per method).
-5 points for correctly completing the ***make_order*** method in the *Customer* class.
-5 points for creating the customer, server, and food truck objects in the main method and correctly placing an order for both customers.
-3 points for correcting ***test_estimated_cost***
-15 points for writing non-trivial test methods for ***test_has_food*** (5 points per scenario correctly tested).
-15 points for writing non-trivial tests for ***test_order_food*** (at least three scenarios; 5 points per scenario correctly tested).
-2 points for writing a test case for **test_withdraw_money**

Extra Credit (6 points)

To gain extra credit on this assignment, do the following: (1) Add a 20% tip for the server when a customer makes an order. (2) Fix the test case to test that your **make_order** function properly includes a tip. Earn up to 3 points for each task (for 6 total).