Homework 4:

Food Market

For this assignment, you will be writing and correcting methods so that customers can successfully order and pay for food at the food market using the new collective ordering system which has waiters take orders (and payment) and pass the orders on to the restaurants. You will also be writing and correcting test cases, so you can guarantee that every step from the order being taken to 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. We cannot emphasize how important this step is.

Overview

Customer Class

The *Customer* class represents a customer who will order from the restaurants. Each customer object has 2 instance variables: **name** (a string representing a customer's name) and **card** (a float showing how much money is in the customer's debit card). The *Customer* class also includes several methods: ***__init__***, ***load_card*** (which adds a passed amount to the **card**), ***submit_order*** (which you will implement – see details below), ***validate_order*** (which takes a **waiter**, a **restaurant,** the **item_name,** and the **quantity** and places an order with that waiter to be delivered to that restaurant), and ***__str__*** *(which prints the customer's information).*

Waiter Class

The *Waiter* class represents a waiter at the market. A waiter object has 4 instance variables: **name** (a string representing a waiter's name), **directory** (a list of restaurants), a **wallet** (total they make in tips), and a **service_fee** (their tip amount). The *Waiter* class includes several methods: ***__init__, has_restaurant*** *(which returns whether the restaurant is in the waiter's* **directory***),* ***add_restaurant*** *(which adds a new restaurant to the waiter's current* **directory***),* ***estimated_cost*** *(which returns the estimated cost of an order, namely the cost of the foods plus the server's own service fee)* ***receive_payment*** *(which takes the customer's money and adds it to the restaurant's* **earning***),* ***place_order*** *(which passes the order, including the ordered food items and quantity, to the restaurant and this function returns the cost of this order (quantity ** **cost**)), and lastly, the ***__str__*** *method (which returns a string representing the waiter, see the starter code for details).*

Restaurant Class

The *Restaurant* class represents a vendor's restaurant . Each restaurant object has 4 instance

variables: **name** (a string which is the name of the restaurant), **inventory** (a dictionary which holds the names of the food as the keys and the quantities of each food as the values), **earnings** (a float for the amount of earnings the restaurant currently has) and **cost** (the cost to the customer for each food. For simplicity, the cost will be the same for all foods in the same restaurant). There is a ***process_order*** method that takes the food **name** and the **quantity**. If the stall has enough food, it will decrease the quantity of that food in the **inventory**. There is a ***__str__*** method that returns a string with the information in the instance variables. We also wrote the constructor for you. You will be in charge of completing the rest of Restaurant class – see details below.

Tasks to Complete

- **Complete the *Customer* Class**

    - Complete the ***submit_order*** method in the *Customer* class. This method takes a **waiter**, a **restaurant** and an **amount** as parameters, and has the customer pay the waiter the specified amount using the ***receive_payment*** method in the waiter class (i.e., it deducts money from the customer's **card**).

- **Complete the *Waiter* Class**

    - Complete the ***recieve_payment*** method in the *Waiter* class. This method takes **restaurant** and **money** parameters, and adds a **service_fee** to the waiter's **wallet** then adds the **money** minus the **service_fee** to the restaurant's **earnings**

- **Complete the *Restaurant* class with the following methods**

    - A ***process_order*** method that takes the food **name** and the **quantity**. If the restaurant has enough food (hint: use ***has_item)***, it will decrease the quantity of that food in the **inventory**.

    - A ***stock_up*** method that takes the **food name** and the **quantity**. It will add 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.

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

    - Create at least *3 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 on their **card**.
    - Create at least 3 *Restaurant* objects. Each should have a unique **name**, **inventory** (use the inventory that you just created)**,** and **cost.**
    - Create at least 2 *Waiter* objects. Each should have a unique **name** and **directory (a list of restaurants)**.
    - Have each customer place at least one order (by calling ***validate_order***) and try all cases in the ***validate_order*** function above. See starter code for hints of all cases.

- **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, but be sure to uncomment all test cases before you turn in your homework.
    - Complete ***test_has_item****,* which tests the ***has_item*** method in the *Restaurant* class. We have provided 3 scenarios for you to test (please refer to the starter code).
    - ***test_validate_order*** has three test scenarios. The first is given to you, but it has an error. Correct the assert statement and write at least one assert statement for each case. The ***validate_order*** method places an order of items from a restaurant to be carried out by a waiter, but only if several conditions are met: if the customer has enough money on their card to pay for the transaction and if the restaurant has enough items in stock. Hint: when testing this method think about which values either do or don't change.
    - Complete ***test_load_card*****,** this tests if the customer can add money into their card

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.

- 10 points for correctly implementing the ***Restaurant*** class (5 per method).
- 5 points for correctly completing the ***submit_order*** method in the *Customer* class.
- 5 points for correctly completing ***recieve_order*** method in *Waiter* class
- 5 points for creating the customer, waiter, and restaurant objects in the **main** method and correctly placing an order for each customer.
- 15 points for writing non-trivial test methods for ***test_has_item*** (5 points per scenario correctly tested).
- 15 points for fixing and writing non-trivial tests for ***test_validate_order*** (at least three scenarios; 5 points per scenario correctly tested).
- 5 points for writing a test case for ***test_load_card***

Extra Credit (6 points)

To gain extra credit on this assignment, please complete the following task:

For every 10^{th} customer that places an order at a waiter, run a lucky draw with a 5% probability of giving the customer a $10 reward on their card.