Homework 4: Food Festival

For this assignment, you will be writing methods so that customers can successfully order and pay for food at the food festival. You will also be writing and fixing 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.

Overview

Customer Class

The ***Customer*** class represents a customer who will order from the vendors. Each customer object has 3 instance variables:

name - a string representing a customer's name

ticket_no – an integer representing a customer's ticket number which is assigned in the order the ticket was purchased

wallet - a float showing how much money is in the customer's food festival payment card. The default value is $50.

The *Customer* class also includes several methods:

__init__ - which initializes the customer's attributes

__str__ - which prints the customer's name and ticket number

reload_wallet(self, amount) - which adds a passed amount to the wallet

place_order(self, vendor, order) - which takes a vendor object and a dictionary that has food objects as the key and quantity as a value and returns a boolean value

Food Class

The ***Food*** class represents a food option at a vendor. A food object has 2 instance variables:

name - a string representing the food's name

cost - a float representing the food's cost

The *Food* class includes 2 methods:

__init__ - which initializes the food's name and cost

__str__ **-** which returns the name and cost

Vendor Class

The ***Vendor*** class represents a vendor's stall. Each vendor object has 3 instance variables:

name - a string which is the name of the vendor

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

earnings - a float representing earnings the vendor currently has

The *Vendor* class also includes several methods:

__init__ - which initializes the attributes

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

calculate_cost(self, food, quantity, customer) - takes the quantity and food name and returns the total cost

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

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

process_order(self, order) **-** which takes a dictionary that has food objects as the key and quantity as a value and returns a boolean value

Tasks to Complete

- **Complete the *Customer* Class**
  - Complete the ***place_order(self, vendor, order)*** method
    - Call the ***calculate_cost*** method on the vendor object to calculate the total cost of the order
    - Check if the customer has the total cost or more in their wallet. If they don't have enough money, print "Insufficient funds" and return **False**
    - Call the ***process_order*** method on the vendor object. If ***process_order*** returns **True**, remove the total cost from the customer's wallet and call the ***receive_payment*** method to add it to the vendor's earnings and return **True**. Otherwise, return **False**.

- **Complete the Vendor Class**

  - A ***calculate_cost(self, food, quantity, customer)*** method that takes the quantity and food name and returns the total cost. It also checks if the customer is one of the first twenty people that purchased a ticket for the festival. If so, it gives them a 25% discount.

  - A ***stock_up(self, food, quantity)*** method that takes the food name 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 has food objects as the key and quantity as a value and checks that there is enough food in the inventory for the order and if not returns **False**. Otherwise, it subtracts the food item from the quantity in the inventory and returns **True**.

- **Write and Fix 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.**

  - Write test cases for the following scenarios for the ***place_order*** method (***test_customer_place_order***):
    - The customer doesn't have enough money in their wallet to make an order
    - The vendor doesn't have enough food in stock
    - The vendor doesn't sell the food item on the order

  - Fix the test cases in ***test_customer_place_order_2***

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.
- 30 points for correctly completing the ***Vendor*** class (10 points per method).
- 15 points for completing ***test_customer_place_order*** (5 points per scenario).
- 5 points for fixing ***test_customer_place_order_2***

Extra Credit (6 points)

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

Every customer ticket number is entered into a raffle. Assuming 100 customers attended the festival, complete the ***raffle*** method in the Customer class which draws a list of 5 random, non-repeating winners, and awards the winners $25 that is added to their wallet. If the customer was selected, prints “Congratulations! You are one of the winners.” If not, print “You were not selected. Better luck next time!”