Homework 4: Request a Service!

For this assignment, you will be writing methods so that users can successfully request and pay from a selection of service types (cleaning, gardening, tutoring, etc.), from different service vendors. You will also be writing and fixing test cases, so you can guarantee that when a user requests a specific service, it is accurately processed and users are satisfied!

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

User Class

The **User** class represents a user who will request a specific service. Most of the class has been provided for you (DO NOT CHANGE any of this code). You will write the ***request_service*** method (see **Tasks to Complete** section).

Each *User* object has 3 instance variables:

- **name -** a string representing a user's name
- **employer_id** - an integer representing the id of the service the user works for. The default value is *None*, if the user does not work for the service.
- **account** - a float showing how many credits are in the user's *account*. The default value is 15 credits.

The *User* class also includes several methods:

- ***__init__*** - which initializes the user's attributes
- ***__str__*** - which returns the user's *name* and the amount of *credits* in their account as as a string
- ***add_credits(self, credits)*** - which adds the passed number of *credits* to the user's account
- ***request_service(self, service_obj, request)*** - which takes a *service_obj* and a *request* dictionary where
    - The keys are *Service* objects and

- The values are another dictionary with keys of *duration* and *priority*. It returns a boolean value (True or False) that indicates if the request was successfully placed or not.

Service

The ***Service*** class represents a type of service they can request (cleaning, gardening, tutoring, etc). The code for this class has been provided for you.

A *Service* object has 2 instance variables:

- **name** - a string representing the name of the service
- **price** - a float representing the *regular price* of a service

The *Service* class includes 2 methods:

- ***__init__*** - which initializes the *Service* object's name and cost
- ***__str__*** - which returns the name and cost as a string

Vendor Class

The ***Vendor*** class represents various service vendors.

Each *Vendor* object has 4 instance variables:

- **name** - a string which is the name of the Vendor
- **vendor_id** - an integer representing the id of the vendor (used for *users* who are also employees to track who they work for)
- **income** - a float representing the income the vendor has collected
- **capacity** - a dictionary which holds the *Service* objects as the keys and the available number of hours for that service from this vendor as the value

The *Vendor* class also includes several provided methods (DO NOT CHANGE any of this code). You will complete additional methods in the **Tasks to Complete** section.

- ***__init__*** - which initializes the *Vendor* attributes
- ***__str__*** - which returns a string with the vendor's name and income
- ***accept_payment(self, amount)*** - which takes an amount and adds it to the vendor's total income

Some of the *Vendor* class and *User class* functions make use of a *request* dictionary. An example of a *request* dictionary is provided below:

```
self.cleaning = ServiceType('Cleaning', 25.00)
self.gardening = ServiceType('Gardening', 40.00)
self.tutoring = ServiceType('Tutoring', 50.00)
self.babysitting = ServiceType('Babysitting', 30.00)

request = {
    self.babysitting: {'duration': 2, 'priority': True},

    self.cleaning: {'duration': 16, 'priority': False},

    self.tutoring: {'duration': 1, 'priority': False}
}
```

Tasks to Complete

➔ Complete *Vendor Class*

- Complete ***calculate_service_cost(self, service_obj, duration, priority, user_obj)***, a method that takes the *service_obj* (*Service* object)*, duration (number of hours), priority* (Boolean variable that specifies if a priority request was made), and *user_obj* (*User* object)
    - It returns the cost based on the service *price* and *duration*.
    - **Note**: A priority request from any service adds a **50% surcharge.**
- Complete ***add_duration(self, service_obj, duration)****,* a method that takes the *service_obj* (*Service* object) and *duration* (number of hours).
    - If the *service_obj* is not a key in the *capacity* dictionary, create a new entry in the capacity dictionary with the *service_obj* as the key and *duration* as the value.
    - If the *service_obj* is a key in the *capacity* dictionary, add the *duration* (number of hours) to the existing value.
- Complete ***process_request(self, request)***, a method which takes a dictionary that represents the user request. The outer keys are *Service* objects and the values are another dictionary with inner keys as *duration* and *priority*. It checks if this vendor has enough available hours for the requested service using its *capacity* dictionary.
    - If not, return **False**.
    - Otherwise, it subtracts the requested hours for that *Service* object from the vendor's *capacity* dictionary, and returns **True**
        - **Note:** A request should only be fulfilled if the vendor has enough hours for that service.

➔ Complete *User Class*

- Complete the ***request_service(self, service_obj, request)*** method
    - Call the ***calculate_service_cost*** method on the *service_obj* (*Service* object) to calculate the cost of the request.
    - Check if the user has the number of credits required or more in their *account*.
        - If they don't have enough credits, return **False.**
        - Otherwise, call the ***process_request*** method on the *service_obj* (*Service* object).
            - If ***process_request*** returns **True**, remove the total number of credits from the user's *account*. Then, call the ***accept_payment*** method to add it to the service's income and return **True**.
            - Otherwise, return **False**.

➔ Write & 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 ***request_service*** method in ***test_user_request_service***:
    - The user doesn't have enough credits in their account to place the request.
    - The vendor does not have enough hours to fulfill the request (the available hours for that service are less than the requested hours).
    - The vendor does not have the requested service type in the *capacity* dictionary.
- Fix the test cases in ***test_user_request_service_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 (specifying 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 ***User*** class function ***request_service***
- 30 points for correctly completing the ***Service*** class (10 points per method)
- 15 points for completing ***test_user_request_service*** (5 points per scenario)
- 5 points for fixing ***test_user_request_service_2***

Extra Credit (6 points)

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

Every user that works for the vendor receives a **20% discount** (this occurs when the user's ***employer_id*** matches the vendor's ***vendor_id***). If the service requested is labeled as ***priority***, then the employee user still has to pay for the 50% upcharge **before** the discount is applied.