# Python OOP Retake Exam – 16 April 2024



*In the heart of Culinarium, "Sphere" revolutionizes dining with its SphereRestaurantApp. Clients earn delightful points through the app, while Head Waiter John and his team craft a seamless dining experience.*

*As the SphereRestaurantApp tallies earnings and discounts, a unique story unfolds—blending hospitality into every shift report and client order. Sphere isn't just a restaurant, it's a digital journey in the culinary world.*

***Note: You are not allowed to change the folder and file structure and their names!***

A screenshot of a computer

Description automatically generated

**Judge Upload**

For the **first two problems**, create a **zip** file with the **project** **folder** and **upload it** to the judge system.

For the **last problem**, create a **zip** file with the **test folder** and **upload it** to the judge system.

You do not need to include **your venv, .idea, pycache, and \_\_MACOSX (for Mac users) in the zip file**, so you do not exceed **the maximum allowed size** of **16.00 KB**.

# Structure (Problem 1) and Functionality (Problem 2)

Your task is to implement all the classes' structure and functionality (properties, methods, inheritance, abstraction, etc.)

You are **free to add additional attributes** (instance attributes, class attributes, methods, dunder methods, etc.) to simplify your code and increase readability as long as it does not change the project's final result in accordance with its requirements so that the program works properly.

## Class BaseClient

In the **base\_client.py** file, the class **BaseClient** should be implemented. It is a **base class** for any **clients**,and it **should not be able to be instantiated**.

### Structure

The class should have the following attributes:

* **name:** str
  + The value represents the **name of the client**.
  + If the name is **an empty string or contains only white spaces**, raise a ValueError with the message: **"Client name should be determined!"**
* **membership\_type: str**
  + Represents the **membership type of the client**.
  + If the membership type is not **"Regular"** or **"VIP"**, raise a ValueErrorwith the message:  
    **"Invalid membership type. Allowed types: Regular, VIP."**
* **points: int**
  + Represents the **total points earned** by a client, based on the spending amount. Set the **initial value** of the property to **zero** (**0**).

### Methods

#### \_\_init\_\_(name: str, membership\_type: str)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### earning\_points(order\_amount: float)

* **Returns** the earned points from the given order's amount.
* Keep in mind that **each type of client** has specific requirements and implements the method differently.

#### apply\_discount()

* Calculatesthe **discount percentage** (based on the **earned points**) **and** determinesthe **remaining points** (the **points left** afterusing someof themfor the **discount**):
  + If clients' earned points are **greater than or equal to 100**, the **discount percentage is 10%**. The client has **used** **100 points** for this discount.
  + If clients' earned points are **between 50 (inclusive) and 100 (exclusive)**, the **discount percentage is 5%**. The client has **used 50 points** for this discount.
  + If clients' earned points are **less than 50**, the **discount percentage is 0%**. No points have been used.
* The method **returns** the **discount percentage** (integer value, e.g. for 10% discount return 10) **and the remaining points** as tuple:

**(discount\_percentage:int, remaining\_points:int)**

* Do not forget, after the discount percentage is calculated, to **subtract the points** that are **being used for the discount.**

## Class RegularClient

In the **regular\_client.py** file, the class **RegularClient** should be implemented. The regular client is a **type of client**. Their **membership** **type** is set to **"Regular"**.

### Methods

#### \_\_init\_\_(name: str)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### earning\_points(order\_amount: float)

* **Returns** the **earned points** from an order with the given amount as an **integer value**.
* The regular client earns **1** **point** for **every $10** **spent** in the restaurant.
* **Round** the **points down** to the **smallest integer value**.
* **Do not forget** to **add** the **earned points** to the **client's total earned points**.

## Class VIPClient

In the **vip\_client.py** file, the class **VIPClient** should be implemented. The VIP clientis a **type of client**. Their **membership** **type** is set to **"VIP"**.

### Methods

#### \_\_init\_\_(name: str)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### earning\_points(order\_amount: float)

* **Returns** the **earned points** from an order with the given amount as an **integer value**.
* The VIP client earns **1** **point** for **every $5** **spent** in the restaurant.
* **Round** the **points down** to the **smallest integer value**.
* **Do not forget** to **add** the **earned points** to the **client's total earned points**.

## Class BaseWaiter

In the **base\_waiter.py** file, the class **BaseWaiter** should be implemented. It is a base class for any type of waiter, and it should **not be able to be instantiated**.

### Structure

The class should have the following attributes:

* **name:** str
  + The value represents the **name of the waiter**.
  + If the name is **less than 3** or **greater than** **50** characters, raise a ValueErrorwith the message:  
    **"Waiter name must be between 3 and 50 characters in length!"**
* **hours\_worked: int**
  + Represents the number of **working hours for each waiter**.
  + If the **working hours are less than** zero (**0**), raise a ValueError with the message:

**"Cannot have negative hours worked!"**

### Methods

#### \_\_init\_\_(name: str, hours\_worked: int)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### calculate\_earnings()

* **Returns** the **total amount of the waiter's earnings**.
* Keep in mind that **each type of waiter** has specific requirements and **implements the method differently**.

#### report\_shift()

* **Returns** information about the water's shift.
* Keep in mind that **each type of waiter** can implement the **method differently**.

#### \_\_str\_\_()

* Returns a **string** with **information** about the **waiter** in the format below.  
  **"Name: {name}, Total earnings: ${total\_earnings}"**
* **Format** the **total earnings** to the **second decimal place**.

## Class FullTimeWaiter

In the **full\_time\_waiter.py** file, the class **FullTimeWaiter** should be implemented. The Full-time waiter is a **type of waiter**.

### Methods

#### \_\_init\_\_(name: str, hours\_worked: int)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### calculate\_earnings()

* The earnings of a Full-time waiter are calculated by multiplying the **waiter's working hours** by the **hourly wage of $15.0**.
* **Returns** the calculated earnings.

#### report\_shift()

* **Returns** information about the **FullTimeWaiter** shift in the following format:  
  **"{name} worked a full-time shift of {hours\_worked} hours."**

## Class HalfTimeWaiter

In the **half\_time\_waiter.py** file, the class **HalfTimeWaiter** should be implemented. The Half-time waiter is a **type of waiter**.

### Methods

#### \_\_init\_\_(name: str, hours\_worked: int)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### calculate\_earnings()

* The earnings of a Half-time waiter are calculated by multiplying the **waiter's working hours** by the **hourly wage of $12.0**.
* **Returns** the calculated earnings.

#### report\_shift()

* **Returns** information about the **HalfTimeWaiter** shift in the following format:  
  **"{name} worked a half-time shift of {hours\_worked} hours."**

## Class SphereRestaurantApp

In the **sphere\_restaurant\_app.py** file, the class **SphereRestaurantApp** should be implemented. It will contain the functionality of the project.

### Structure

The class should have the following attributes:

* **waiters: list**
  + Initially an empty list, being used to store all **waiter objects** hired at Sphere Restaurant.
* **clients: list**
  + Initially an empty list, being used to store all **client objects** who visit the Sphere Restaurant.

### Methods

#### \_\_init\_\_()

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### hire\_waiter(waiter\_type: str, waiter\_name: str, hours\_worked: int)

The method **creates** a waiter of the given type and **adds** them to the **waiters** collection.

* If the waiter's type is **not valid**, **return** the following message:

**"{waiter\_type} is not a recognized waiter type."**

* If a waiter with the same **name** is already added to the list, **do not duplicate records**, **return** the following message:

**"{waiter\_name} is already on the staff."**

* If **none of the above** cases is reached, the **waiter** is **successfully created**. Store the waiter in the appropriate collection and **return**:

**"{waiter\_name} is successfully hired as a {waiter\_type}."**

* **Valid types** of waiters are: **"FullTimeWaiter"** and **"HalfTimeWaiter"**

#### admit\_client(client\_type: str, client\_name: str)

The method **creates** a client of the given type and **adds** them to the **clients** collection. The method is responsible for **admitting a new client**.

* First, check if the **client type** is valid, and if **not**, **return** the following message:

**"{client\_type} is not a recognized client type."**

* If a client with the **same** **name** is already added to the list, **do not duplicate records**, **return** the following message:

**"{client\_name} is already a client."**

* If the above cases are **not reached**, create the correct type of **client** and add them to the appropriate collection. **Return** the following message:

**"{client\_name} is successfully admitted as a {client\_type}."**

* **Valid types** of clients are: **"RegularClient"** and **"VIPClient"**.

#### process\_shifts(waiter\_name: str)

The method is responsible for **managing the shifts of waiters**.

* If a **waiter is found**, **generate** a **shift report**. The shift report is then **returned** in the following format:  
  **"{name} worked a {full-time/half-time} shift of {hours\_worked} hours."**
* **Hint:** You can use the waiter **report\_shift()** method.
* If **no waiter is found** with the specified name, the method **returns** the following message:  
  **"No waiter found with the name {waiter\_name}."**

#### process\_client\_order(client\_name: str, order\_amount: float):

The method is responsible for **processing the client's order**.

* If the **client is found, it calculates the points earned from the order** and **returns** a message in the following format:  
  "{client\_name} earned {points\_earned} points from the order."
* Hint: You can use the earning\_points()method.
* If the **client's name does not exist**,the method **returns** the following message:   
  "{client\_name} is not a registered client."

#### apply\_discount\_to\_client(client\_name: str):

The method is responsible for **applying a discount to a client's account**.

* If the **client is successfully identified**, it **applies the discount** and returns a message in the following format:  
  "{client\_name} received a {discount\_percentage}% discount. Remaining points {remaining\_points}"
* Hint: You can use the apply\_discount() method.
* If the **client does not exist**, the method **returns** the following message:  
  "{client\_name} cannot get a discount because this client is not admitted!"

#### generate\_report()

**Returns** a detailed report for the **Sphere Restaurant**. The report includes information about **total earnings**, **the total clients unused points**, **total clients count**, and **individual details for each waiter**.

The **waiters** are **sorted** by their **total earnings in descending order**, and the information is presented in the following format:  
**"$$ Monthly Report $$**

Total Earnings: ${total\_earnings}

**Total Clients Unused Points**: {total\_client\_points}

Total Clients Count: {clients\_count}

\*\* Waiter Details \*\*

**Name: {name1}, Total earnings: ${waiter1\_total\_earnings}**

**Name: {name2}, Total earnings: ${waiter2\_total\_earnings}**

**...**

**Name: {nameN}, Total earnings: ${waiterN\_total\_earnings}"**

* The total\_earnings represent the **sum** of **all** waiters' earnings, **formatted** to the **second decimal place**.
* The **waiter\_**total\_earnings represent **each** waiter's earnings, **formatted** to the **second decimal place**.
* **Hint**: You can use the waiters **calculate\_earnings()** and **str()** methods.

## Examples

|  |
| --- |
| **Input** |
| *# Create an instance of SphereRestaurantApp*  **sphere\_restaurant\_app = SphereRestaurantApp()**  *# Hire some waiters*  **print(sphere\_restaurant\_app.hire\_waiter("FullTimeWaiter", "John", 40))**  **print(sphere\_restaurant\_app.hire\_waiter("HalfTimeWaiter", "Alice", 20))**  **print(sphere\_restaurant\_app.hire\_waiter("InvalidWaiter", "JohnDoe", 10))**  **print(sphere\_restaurant\_app.hire\_waiter("HalfTimeWaiter", "Charlie", 30))**  **print(sphere\_restaurant\_app.hire\_waiter("FullTimeWaiter", "Frank", 50))**  **print(sphere\_restaurant\_app.hire\_waiter("HalfTimeWaiter", "Alice", 60))**  *# Admit some clients*  **print(sphere\_restaurant\_app.admit\_client("InvalidClient", "JohnDoe"))**  **print(sphere\_restaurant\_app.admit\_client("VIPClient", "Eve"))**  **print(sphere\_restaurant\_app.admit\_client("VIPClient", "Lila"))**  **print(sphere\_restaurant\_app.admit\_client("RegularClient", "Bob"))**  **print(sphere\_restaurant\_app.admit\_client("VIPClient", "Eve"))**  **print(sphere\_restaurant\_app.admit\_client("RegularClient", "Oscar"))**  *# Process shifts*  **print(sphere\_restaurant\_app.process\_shifts("John"))**  **print(sphere\_restaurant\_app.process\_shifts("Alice"))**  **print(sphere\_restaurant\_app.process\_shifts("Emily"))**  **print(sphere\_restaurant\_app.process\_shifts("Frank"))**  *# Process client orders*  **print(sphere\_restaurant\_app.process\_client\_order("Bob", 100.0))**  **print(sphere\_restaurant\_app.process\_client\_order("Eve", 500.0))**  **print(sphere\_restaurant\_app.process\_client\_order("JohnDoe", 250.0))**  **print(sphere\_restaurant\_app.process\_client\_order("Bob", 750.0))**  **print(sphere\_restaurant\_app.process\_client\_order("Lila", 550.0))**  **print(sphere\_restaurant\_app.process\_client\_order("Oscar", 84.0))**  *# Apply discounts to clients*  **print(sphere\_restaurant\_app.apply\_discount\_to\_client("Lila"))**  **print(sphere\_restaurant\_app.apply\_discount\_to\_client("Eve"))**  **print(sphere\_restaurant\_app.apply\_discount\_to\_client("JohnDoe"))**  **print(sphere\_restaurant\_app.apply\_discount\_to\_client("Oscar"))**  **print(sphere\_restaurant\_app.apply\_discount\_to\_client("Bob"))**  *# Generate report*  **print(sphere\_restaurant\_app.generate\_report())** |
| **Output** |
| **John is successfully hired as a FullTimeWaiter.**  **Alice is successfully hired as a HalfTimeWaiter.**  **InvalidWaiter is not a recognized waiter type.**  **Charlie is successfully hired as a HalfTimeWaiter.**  **Frank is successfully hired as a FullTimeWaiter.**  **Alice is already on the staff.**  **InvalidClient is not a recognized client type.**  **Eve is successfully admitted as a VIPClient.**  **Lila is successfully admitted as a VIPClient.**  **Bob is successfully admitted as a RegularClient.**  **Eve is already a client.**  **Oscar is successfully admitted as a RegularClient.**  **John worked a full-time shift of 40 hours.**  **Alice worked a half-time shift of 20 hours.**  **No waiter found with the name Emily.**  **Frank worked a full-time shift of 50 hours.**  **Bob earned 10 points from the order.**  **Eve earned 100 points from the order.**  **JohnDoe is not a registered client.**  **Bob earned 75 points from the order.**  **Lila earned 110 points from the order.**  **Oscar earned 8 points from the order.**  **Lila received a 10% discount. Remaining points 10**  **Eve received a 10% discount. Remaining points 0**  **JohnDoe cannot get a discount because this client is not admitted!**  **Oscar received a 0% discount. Remaining points 8**  **Bob received a 5% discount. Remaining points 35**  **$$ Monthly Report $$**  **Total Earnings: $1950.00**  **Total Clients Unused Points: 53**  **Total Clients Count: 4**  **\*\* Waiter Details \*\***  **Name: Frank, Total earnings: $750.00**  **Name: John, Total earnings: $600.00**  **Name: Charlie, Total earnings: $360.00**  **Name: Alice, Total earnings: $240.00** |

# Task 3: Unit Tests (100 points)

You will **be provided with another skeleton** for this problem. **Open** the **new skeleton** as a **new project** and write tests for the **Restaurant** class. The class will have some methods, fields, and one constructor, all of them working properly. You are **NOT ALLOWED** to change anything in the class code. Cover the whole class with unit tests to make sure that the class is working as intended. Submit **only the test** folder.