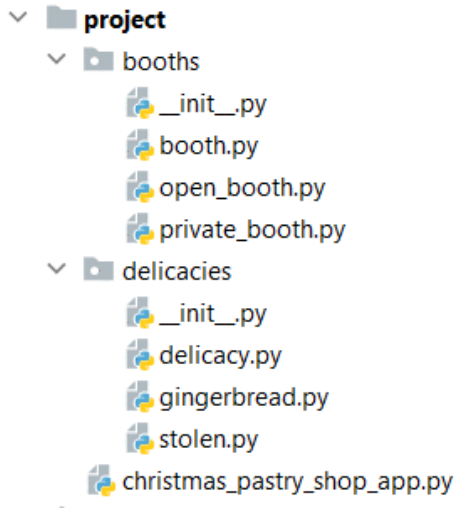
Python OOP Exam - Christmas Pastry Shop

Submit your solutions in the SoftUni judge system at https://judge.softuni.org/Contests/Practice/Index/3728#0

*As we all love delicacies, today you were chosen to build a simple Christmas pastry shop software system. This system must have support for **delicacies** and **booths** in the pastry shop.*

You will be provided with a **skeleton** that includes all the folders and files that you will need.

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



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 **in the zip file** your **`venv`**, **`.idea`**, **`pycache`**, and **`__MACOSX`** (for Mac users), so you do not exceed **the maximum allowed size** of **`16.00 KB`**.

Structure (Problem 1) and Functionality (Problem 2)

Our task is to implement the **structure and functionality** of all the classes (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 according to the requirements and the program works properly.

1. Class Delicacy

In the **`delicacy.py`** file, the class **`Delicacy`** should be implemented. It is a **base class** for any **type of delicacy,** 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 a delicacy**.
  - If the name is **an empty string or contains only white spaces**, raise a **`ValueError`** with the message: **`"Name cannot be null or whitespace!"`**
- **`portion: int`**
  - The value represents the **portion of a delicacy in grams**.
- **`price: float`**
  - The value represents the **price of a delicacy**.
  - If the price **is less or equal to 0.0,** raise a **`ValueError`** with the message: **`"Price cannot be less or equal to zero!"`**

Methods

__init__(name: str, portion: int, price: float)

- In the **`__init__`** method, all the needed attributes must be set.

details()

- Returns information about **each delicacy**.

2. Class Gingerbread

In the **`gingerbread.py`** file, the class **`Gingerbread`** should be implemented. Gingerbread is a **type of delicacy**. Each **gingerbread portion is exactly 200 grams.**

Methods

__init__(name: str, price: float)

- In the **`__init__`** method, all the needed attributes must be set.

details()

- Returns a string in the following format:

  `"Gingerbread {name}: 200g - {price - formatted to the second digit}lv."`

3. Class Stolen

In the **`stolen.py`** file, the class **`Stolen`** should be implemented. Stolen is a **type of delicacy**. Each **stolen portion is exactly 250 grams.**

Methods

__init__(name: str, price: float)

- In the **`__init__`** method, all the needed attributes must be set.

details()

- Returns a string in the following format:

  `"Stolen {name}: 250g - {price - formatted to the second digit}lv."`

4. Class Booth

In the **`booth.py`** file, the class **`Booth`** should be implemented. It is a **base class** for all different **types of booths,** and it **should not be able to be instantiated**.

Structure

The class should have the following attributes:

- **`booth_number: int`**
  - The value represents the booth's number.
- **`capacity: int`**
  - The value represents the booth's capacity.
  - It can't be **less than zero**. In these cases, raise a **`ValueError`** with the message: **`"Capacity cannot be a negative number!"`**
- **`delicacy_orders: list`**
  - Empty list that **will contain delicacies (objects) that are ordered**.
- **`price_for_reservation: float`**
  - **Initial value is 0**.
  - Each time a **booth is reserved**, the **price for a reservation** should be **set**.
- **`is_reserved: bool`**
  - Default value is **`False`**.
  - Set to **`True`** if the **booth** is **reserved**, otherwise **`False`.**

Methods

__init__(booth_number: int, capacity: int)

- In the **`__init__`** method, all the needed attributes must be set.

reserve(number_of_people: int)

- Reserves the booth depending on each booth type.

5. Class OpenBooth

In the **`open_booth.py`** file, the class **`OpenBooth`** should be implemented. The open booth is a **type of booth**.

Methods

__init__(booth_number: int, capacity: int)

- In the **`__init__`** method, all the needed attributes must be set.

reserve(number_of_people: int)

- Calculates the **price for reservation**, by **multiplying** the **price per person** by the **number of people**. The **price per person to reserve a open booth is 2.50**.
- **Set** the **price for reservation** and **reserve the booth**.

6. Class PrivateBooth

In the **`private_booth.py`** file, the class **`PrivateBooth`** should be implemented. The private booth is a **type of booth**.

Methods

__init__(booth_number: int, capacity: int)

- In the **`__init__`** method, all the needed attributes must be set.

reserve(number_of_people: int)

- Calculates the **price for reservation**, by **multiplying** the **price per person** by the **number of people**. The **price per person to reserve a private booth is 3.50**.
- **Set** the **price for reservation** and **reserve the booth**.

7. Class ChristmasPastryShopApp

In the **`christmas_pastery_shop_app.py`** file, the class **`ChristmasPastryShopApp`** should be implemented. It will contain the functionality of the project.

Structure

The class should have the following attributes:

- **`booths: list`**
  - Empty list that **will contain all booths** (objects) that are created.
- **`delicacies: list`**
  - Empty list that **will contain all delicacies** (objects) that are created.
- **`income: float`**
  - Initial value is **0.0**.
  - The value represents the **total income of the pastry shop**.

Methods

__init__()

- In the **`__init__`** method, all the needed attributes must be set.

add_delicacy(type_delicacy: str, name: str, price: float)

The method **creates** a delicacy of the given type and **adds** it to the delicacies' collection.
All delicacy **names** should be **unique**.

- If a delicacy with that name exists, raise an **Exception** with the following message: **"{delicacy name} already exists!"**
- If the delicacy type is not valid, raise an **Exception** with the following message: **"{type of delicacy} is not on our delicacy menu!"**
- Otherwise, **create** the delicacy, **add** it to the delicacies' list, and **return** the following message: **"Added delicacy {delicacy name} - {type of delicacy} to the pastry shop."**
- **Valid types** of delicacies are: **"Gingerbread"** and **"Stolen"**

add_booth(type_booth: str, booth_number: int, capacity: int)

The method **creates** a booth of the given type and **adds** it to the booths' collection.
All booth **numbers** should be **unique**.

- If a booth with that **number exists**, raise an **Exception** with the following message: **"Booth number {booth number} already exists!"**
- If the booth **type is not valid**, raise an **Exception** with the following message: **"{type of booth} is not a valid booth!"**
- Otherwise, **create** the booth, **add** it to the booths' list and **return** the following message: **"Added booth number {booth number} in the pastry shop."**
- **Valid types** of delicacies are: **"Open Booth"** and **"Private Booth"**

reserve_booth(number_of_people: int)

Finds **the first** booth that is **not reserved** and whose **capacity is enough** for the number of people provided.

- If there is **no such booth**, raise an **Exception** with the following message: **"No available booth for {number of people} people!"**
- Otherwise, **reserves** the booth and return: **"Booth {booth number} has been reserved for {number of people} people."**

order_delicacy(booth_number: int, delicacy_name: str)

Finds the **booth** with the provided number and the **delicacy** with the provided name; and **orders the delicacy** for that booth.

- If there is **no such booth**, raise an **Exception** with the following message: **"Could not find booth {booth number}!"**
- If there is **no such delicacy**, raise an **Exception** with the following message: **"No {delicacy name} in the pastry shop!"**
- Otherwise, **order** the delicacy for that booth and return: **"Booth {booth number} ordered {delicacy name}."**

leave_booth(booth_number: int)

- Finds the booth with the same booth's number (**the booth's number will always be valid**).
- **Calculates the bill** for that booth taking the **price for reservation** and all the **price of all orders**. The **bill** is **added** to the pastry shop's total **income**.
- **Removes** all the ordered **delicacies**, **frees the booth**, and sets the **price for reservation** to **0**.
- Finally returns:
  "Booth {booth number}:"
  "Bill: {bill - formatted to the second decimal}lv."

get_income()

- Returns the total income for the pastry shop for **all completed bills in the format**:

  "Income: {income - formatted to the second decimal place}lv."

Examples

Input
shop = ChristmasPastryShopApp() print(shop.add_delicacy("Gingerbread", "Gingy", 5.20)) print(shop.delicacies[0].details()) print(shop.add_booth("Open Booth", 1, 30)) print(shop.add_booth("Private Booth", 10, 5)) print(shop.reserve_booth(30)) print(shop.order_delicacy(1, "Gingy")) print(shop.leave_booth(1)) print(shop.reserve_booth(5)) print(shop.order_delicacy(1, "Gingy")) print(shop.order_delicacy(1, "Gingy")) print(shop.order_delicacy(1, "Gingy")) print(shop.leave_booth(1)) print(shop.get_income())
Output
Added delicacy Gingy - Gingerbread to the pastry shop. Gingerbread Gingy: 200g - 5.20lv. Added booth number 1 in the pastry shop. Added booth number 10 in the pastry shop. Booth 1 has been reserved for 30 people. Booth 1 ordered Gingy. Booth 1: Bill: 80.20lv. Booth 1 has been reserved for 5 people. Booth 1 ordered Gingy. Booth 1 ordered Gingy. Booth 1 ordered Gingy. Booth 1: Bill: 28.10lv. Income: 108.30lv.

Task 3: Unit Tests

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