Python OOP Regular Exam - 08 December 2024



*At Marlowe's Auction House, prestigious museums and private collectors competed for ancient artifacts, Renaissance masterpieces, and contemporary artworks. As the auction concluded, the artifacts, entrusted to new guardians, symbolized the timeless legacy of human expression and the profound impact of art across generations.*

***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 if it does not change the project's final result under its requirements so that the program works properly.

1. **Class BaseArtifact**

In the **base\_artifact.py** file, the class **BaseArtifact** should be implemented. It is a **base class** for any **type of artifact**,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 artifact**.
  + If the name is **an empty string or contains only white spaces**, **raise** a ValueError with the message: **"Artifact name cannot be null or empty!"**
* **price:** **float**
  + Represents the **price of each artifact**.
  + If the price of the artifact is **equal to or less than 0.0**, **raise** a ValueError with the message: **"Artifact price should be more than 0.0!"**
* **space\_required:** **int**
  + Represents the **space required by any artifact for its exhibition**.
  + Must be **between 1 and 1000**, **both inclusive**. If it is not, **raise** a ValueError with the message: **"Space required for the artifact exhibition must be between 1 and 1000!"**

**Methods**

**\_\_init\_\_(name: str, price: float, space\_required: int)**

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

**artifact\_information()**

* **A method that provides** information for each artifact.
* Remember that **each artifact type** can **implement** the **method differently**.

1. **Class RenaissanceArtifact**

In the **renaissance\_artifact.py** file, the class **RenaissanceArtifact** should be implemented. **Renaissance Artifacts** are a **type of artifact**.

**Methods**

**\_\_init\_\_(name: str, price: float, space\_required: int)**

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

**artifact\_information()**

* Provides information about the **Renaissance Artifact** in the following format:
  + The **price should be formatted to the second decimal** **place**.

**"Renaissance Artifact: {artifact\_name}; Price: {artifact\_price}; Required space: {artifact\_space\_required}"**

1. **Class ContemporaryArtifact**

In the **contemporary\_artifact.py** file, the class **ContemporaryArtifact** should be implemented. **Contemporary artifacts** are a **type of artifact**.

**Methods**

**\_\_init\_\_(name: str, price: float, space\_required: int)**

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

**artifact\_information()**

* Provides information about the **Contemporary Artifact** in the following format:
  + The **price should be formatted to the second decimal** **place**.

**"Contemporary Artifact: {artifact\_name}; Price: {artifact\_price}; Required space: {artifact\_space\_required}"**

1. **Class BaseCollector**

In the **base\_collector.py** file, the class **BaseCollector** should be implemented. It is a **base class** for any **type of collector**, 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 collector**.
  + The name must **only contain letters**, **numbers**, and **optional white spaces between them**. If it doesn't meet these criteria, **raise a ValueError** with the following message:  
    **"Collector name must contain letters, numbers, and optional white spaces between them!"**
* **available\_money:** **float**
  + Represents the **money available** to the **collector** for making purchases.
  + If the **amount becomes less than 0.0, raise** a **ValueError** with the message:

**"A collector cannot have a negative amount of money!"**

* **available\_space:** **int**
  + Represents the **space** available to the **collector** for exhibiting artifacts.
  + If the **amount becomes less than 0**, **raise** a **ValueError** with the message:

**"A collector cannot have a negative space available for exhibitions!"**

* **purchased\_artifacts: list**
  + **Initially** set to an **empty list**.
  + It will store a **collection of all the artifacts** **purchased** by each **collector**.

**Methods**

**\_\_init\_\_(name: str, available\_money: float, available\_space: int)**

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

**increase\_money()**

* **Increases** the **money** available to the **collector** by a **certain amount**.
* Remember that **each collector type** can **implement** the **method differently**.

**can\_purchase(artifact\_price: float, artifact\_space\_required: int)**

* The method **checks** whether the **collector has enough money and space to purchase a certain artifact**.
  + The **collector** should have **at least** the **same amount of money** as the **price of the artifact to buy it and at least** the **same available space** as the **artifact's required space**.
* **Returns True** if the collector can purchase the artifact, otherwise **False**.

**\_\_str\_\_()**

* **Returns** a **string** with **information** about the **collector** in the format given below.
  + The **names** of the **purchased artifacts** should be separated by a **comma and a space** **(", ")** and **sorted alphabetically in descending order**.
    - If there are **no artifacts**, **use** the string **"none"** instead.
  + The **money available should be formatted to the second decimal** **place.**

**"Collector name: {name}; Money available: {available\_money}; Space available: {available\_space}; Artifacts: {artifact\_name1, artifact\_name2, …, artifact\_namen /'none'}"**

1. **Class Museum**

In the **museum.py** file, the class **Museum** should be implemented. The Museum is a **type of collector**. Each **museum** has an **initial available money of 15000.0** andan **initial available space of 2000**.

**Methods**

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

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

**increase\_money()**

* The method **increases** the available **amount of money** by **1000.0**

1. **Class PrivateCollector**

In the **private\_collector.py** file, the class **PrivateCollector** should be implemented. The Private collector is a **type of collector**. Each **private collector** has an **initial available amount of money of 25000.0** and an **initial available space of 3000**.

**Methods**

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

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

**increase\_money()**

* The method **increases** the available **amount of money** by **5000.0**

1. **Class AuctionHouseManagerApp**

In the **auction\_house\_manager\_app.py** file, the class **AuctionHouseManagerApp** should be implemented. It will contain the functionality of the project.

**Structure**

The class should have the following attributes:

* **artifacts: list**
  + Initially an empty list to store **all artifacts objects** registered for the Auction.
* **collectors: list**
  + Initially an empty list to store **all collectors objects** participating in the Auction.

**Methods**

**\_\_init\_\_()**

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

**register\_artifact(artifact\_type: str, artifact\_name: str, artifact\_price: float, artifact\_space: int)**

The method **creates** (upon certain conditions) an **artifact** of the given type and **adds** it to the **artifacts** collection:

* **First**, check **if the artifact type is valid**, and **if not**, **raise** a **ValueError** with the following message: "**Unknown artifact type!"**
* **Then**, check if an artifact with the **same** **name** is already added to the collection regardless of its type, **do not duplicate names**,and **raise** a **ValueError** with the following message:

"**{artifact\_name} has been already registered!"**

* If none of the above cases is reached, **create and** **add** the **artifact** to the **artifacts** collection. **Return** the following message:

"**{artifact\_name} is successfully added to the auction as {artifact\_type}."**

* **Valid types** of artifacts are: **"RenaissanceArtifact"** and **"ContemporaryArtifact"**

**register\_collector(collector\_type: str, collector\_name: str)**

The method **creates** (upon certain conditions) a **collector** of the given type and **adds** it to the **collectors** collection:

* **First**, check if the **collector type** is valid, and **if** **not**, **raise** a **ValueError** with the following message: "**Unknown collector type!"**
* **Then**, check if a **collector** with the **same** **name** is already added to the list regardless of their type, **do not duplicate names**,and **raise** a **ValueError** with the following message:

"**{collector\_name} has been already registered!"**

* If none of the above cases is reached, **create and add** the **collector** to the **collectors** collection. **Return** the following message:

"**{collector\_name} is successfully registered as a {collector\_type}."**

* **Valid types** of collectors are: **"Museum"** and **"PrivateCollector"**

**perform\_purchase(collector\_name: str, artifact\_name: str)**

The method performs a purchase upon certain conditions. A **collector** with thegiven name **purchases** aspecific **artifact** with the given name **if all pre-defined conditions are met**:

* **First**,check if sucha **collector exists**:
  + Validate a collector's **existence** with the given **name** in the collection of registered **collectors**.
  + If **no collector is found**, the method **raises** a **ValueError** with the following message:

**"Collector {collector\_name} is not registered to the auction!"**

* **Then**,check if such an **artifact exists**:
  + Validate the **existence** of an **artifact** with the given **name** in the collection of **artifacts**.
  + If **no artifact is found**, the method **raises** a **ValueError** with the following message:

**"Artifact {artifact\_name} is not registered to the auction!"**

**If both objects exist**,proceed with the **purchase attempt**:

* **First**,check **if the collector has enough available space and money** to **buy and exhibit** the **artifact**:
  + If the collector **does not have enough space or money**, **the purchase is impossible**.
  + **Return** the following message: **"Purchase is impossible."**
    - **Hint**: You can use the [**can\_purchase()**](#can_purchase) method.
* **Then**, the collector **buys the artifact**.A series of **actions are required**:
  + The **artifact is removed** from the **auction's artifacts** collection.
  + The **artifact is added** to the **collector's collection** of **purchased** **artifacts**.
  + The **collector's available money is decreased accordingly**.
  + The **collector's available space is decreased accordingly**.
  + **Return** the following message and **format the price** to the **second decimal place**:

**"{collector\_name} purchased {artifact\_name} for a price of {artifact\_price}."**

**remove\_artifact(artifact\_name: str)**

The method **removes** an **artifact with the given name** if such **exists** and has **not been sold** yet.

* **First**, check if such an **artifact** **exists** in the collection of **available-for-sale artifacts** (auction's artifact collection). If **not existing**, **return** the message:

**"No such artifact."**

* If the **artifact exists** and is **available for sale**, **remove it** from the **auction's artifacts collection** and **return** the following message:

**"Removed {artifact\_type}: {artifact\_name}; Price: {artifact\_price}; Required space: {artifact\_space\_required}"**

* **Hint**: you can use the [**artifact\_information()**](#artifact_information) method.

**fundraising\_campaigns(max\_money: float)**

The method starts **fundraising campaigns** for those **collectors** whose **available money is less than or equal to** the **specified maximum amount** (**max\_money**) and **increases** it.

* **Each type** of **collector increases** their **funds** with **different amounts** (**1000.0** for **museums**, **5000.0** for private **collectors**)
  + **Hint**: Use the [**increase\_money()**](#increase_money) method.
* The method should **track the number of collectors** (zero or more) that have **increased their available money**.
* **Return** the following message:

**"{count} collector/s increased their available money."**

**get\_auction\_report()**

The method should **return** detailed **information** about the **auction**:

* Include information about **collectors and their purchased artifacts**.
  + **Sort** them by the **number** of **purchased artifacts** in **descending order**.
  + If there is more than one collector with the **same number** of **purchased artifacts**, **sort** thembythe **collector's name ascending**.
  + **Sort** the **artifact names in the collector's** collections **alphabetically** **in descending order**.
    - If there are **no artifacts**, **use** the string **"none"** instead. See the [**Examples**](#_Examples).
  + The **money available should be formatted to the second decimal** **place**.
  + **Hint**: You can use the collector's [**\_\_str\_\_()**](#str) method
* The statistics information should follow the format of the example below (**each collector's info is on a new line**):

**"\*\*Auction statistics\*\***

**Total number of sold artifacts: {count\_of\_sold\_artifacts}**

**Available artifacts for sale: {count\_of\_available\_artifacts}**

**\*\*\***

**Collector name: {name1}; Money available: {available\_money1}; Space available: {available\_space1}; Artifacts: {artifact\_name1, artifact\_name2, …, artifact\_namen}**

**Collector name: {name2}; Money available: {available\_money2}; Space available: {available\_space2}; Artifacts: {artifact\_name1, artifact\_name2, …, artifact\_namen}**

**...**

**Collector name: {namen}; Money available: {available\_moneyn}; Space available: {available\_spacen}; Artifacts: {artifact\_name1, artifact\_name2, …, artifact\_namen}"**

### **Examples**

|  |
| --- |
| **Test Code** |
| ***# Create an instance of AuctionHouseManagerApp***  manager = AuctionHouseManagerApp()  ***# Register artifacts***  print(manager.register\_artifact("RenaissanceArtifact", "Kohinoor", 5000.0, 10))  print(manager.register\_artifact("RenaissanceArtifact", "Zelda", 5000.0, 10))  print(manager.register\_artifact("RenaissanceArtifact", "Mona Lisa", 10000.0, 100))  print(manager.register\_artifact("ContemporaryArtifact", "The Scream", 2000.0, 1000))  print(manager.register\_artifact("ContemporaryArtifact", "Untitled", 32000.0, 90))  print()  ***# Register collectors***  print(manager.register\_collector("PrivateCollector", "Josh Smith"))  print(manager.register\_collector("Museum", "Louvre"))  print(manager.register\_collector("Museum", "Hermitage"))  print()  ***# Perform purchases***  print(manager.perform\_purchase("Josh Smith", "Mona Lisa"))  print(manager.perform\_purchase("Louvre", "Kohinoor"))  print(manager.perform\_purchase("Josh Smith", "Zelda"))  print(manager.perform\_purchase("Josh Smith", "The Scream"))  print(manager.perform\_purchase("Josh Smith", "Untitled"))  print()  ***# Remove artifact***  print(manager.remove\_artifact("The Scream"))  print(manager.remove\_artifact("Nonexistent"))  print()  ***# Start fund-raising campaigns***  print(manager.fundraising\_campaigns(10000.0))  print()  ***# Get auction report***  print(manager.get\_auction\_report())  print()  ***# Remove artifact***  print(manager.remove\_artifact("Untitled")) |
| **Output** |
| Kohinoor is successfully added to the auction as RenaissanceArtifact.  Zelda is successfully added to the auction as RenaissanceArtifact.  Mona Lisa is successfully added to the auction as RenaissanceArtifact.  The Scream is successfully added to the auction as ContemporaryArtifact.  Untitled is successfully added to the auction as ContemporaryArtifact.  Josh Smith is successfully registered as a PrivateCollector.  Louvre is successfully registered as a Museum.  Hermitage is successfully registered as a Museum.  Josh Smith purchased Mona Lisa for a price of 10000.00.  Louvre purchased Kohinoor for a price of 5000.00.  Josh Smith purchased Zelda for a price of 5000.00.  Josh Smith purchased The Scream for a price of 2000.00.  Purchase is impossible.  No such artifact.  No such artifact.  2 collector/s increased their available money.  \*\*Auction statistics\*\*  Total number of sold artifacts: 4  Available artifacts for sale: 1  \*\*\*  Collector name: Josh Smith; Money available: 13000.00; Space available: 1890; Artifacts: Zelda, The Scream, Mona Lisa  Collector name: Louvre; Money available: 11000.00; Space available: 1990; Artifacts: Kohinoor  Collector name: Hermitage; Money available: 15000.00; Space available: 2000; Artifacts: none  Removed Contemporary Artifact: Untitled; Price: 32000.00; Required space: 90 |

## **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 **SeniorStudent** 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.