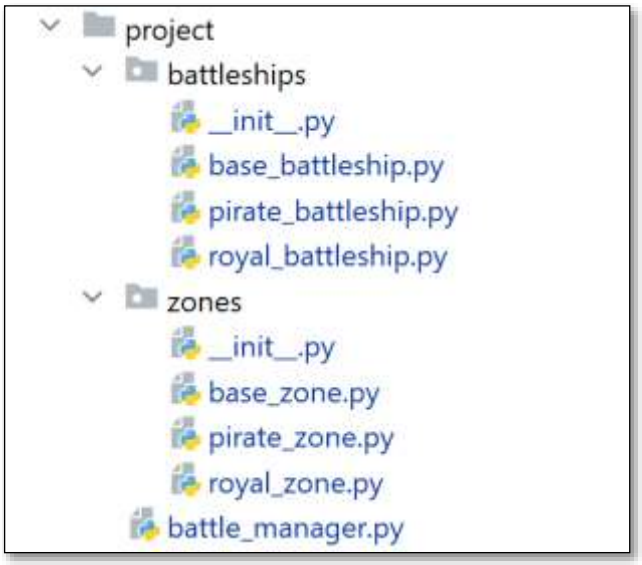
Python OOP Retake Exam - 14 August 2024

*Welcome to the **BattleZones** simulation system, where **Royal** and **Pirate Battleships** engage in strategic combat within **designated battle zones**. Each battle zone, whether a **Royal Zone** or **Pirate Zone**, hosts battleships, each with defined attributes and behaviors. Manage these zones and battleships through a centralized **Battle manager**, ensuring thrilling and tactical warfare.*

You are provided with a **skeleton** that includes all folders and files you need.

Note: You are not allowed to change the folder and file structure or 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.

Do not 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)

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

You can **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 following its requirements and proper workflow.

1. Class BaseBattleship

In the **`base_battleship.py`** file, the class **`BaseBattleship`** should be implemented. It serves as a **base class** for **any type** of **battleship** and should **not be instantiated directly**.

Structure

The class should have the following attributes:

- **`name: str`**
  - The value represents the **name** of the **battleship**.
  - The name should contain **letters only**, if not **raise** a **`ValueError`** with the message: **`"Ship name must contain only letters!"`**
- **`health: int`**
  - The value represents the **health** of the **battleship**.
  - If the health **drops below `0`**, set it to **`0`.**
- **`hit_strength: int`**
  - The value represents the **damage** the **battleship inflicts**.
- **`ammunition: int`**
  - The value represents the **ammunition** of the **battleship**.
- **`is_attacking: bool`**
  - The value determines whether the ship **is attacking or being attacked**.
  - The **initial value** is set to **`False`.**
- **`is_available: bool`**
  - The value determines whether the ship **is currently participating in a battle**.
  - The **initial value** is set to **`True`** (not participating).

Methods

__init__(name: str, health: int, hit_strength: int, ammunition: int)

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

take_damage(enemy_battleship: BaseBattleship)

- When the **ship is attacked**, it **takes damage (decreasing `health`) equal** to the **hit strength** of the **attacking enemy ship**.

attack()

- When the **ship attacks**, its **ammunition decreases**. **Each type** of **battleship** implements this method **differently**.

2. Class RoyalBattleship

In the **`royal_battleship.py`** file, the class **`RoyalBattleship`** should be implemented. The **Royal Battleship** is a **type** of **`BaseBattleship`**. Each **Royal Battleship initially** has **`100` units** of **ammunition**.

Methods

__init__(name: str, health: int, hit_strength: int)

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

attack()

- The method **reduces** the **Royal Battleship's ammunition amount by 25 units**. If the value **drops below zero**, **set it to zero** (**`0`**).

3. Class PirateBattleship

In the **`pirate_battleship.py`** file, the class **`PirateBattleship`** should be implemented. The **Pirate Battleship** is a **type** of **`BaseBattleship`**. Each **Pirate Battleship initially** has **`80` units** of **ammunition**.

Methods

__init__(name: str, health: int, hit_strength: int)

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

attack()

- The method **reduces** the **Pirate Battleship's ammunition amount by 10 units**. If the value **drops below zero**, **set it to zero** (**`0`**).

4. Class BaseZone

In the **`base_zone.py`** file, the class **`BaseZone`** should be implemented. It serves as a **base class** for **any type** of **battle zone** and should **not be instantiated directly**.

Structure

The class should have the following attributes:

- **`code: str`**
  - The value represents the **code of the zone**.
  - The **code must contain only digits**, if not **raise** a **`ValueError`** with the message: **`"Zone code must contain digits only!"`**
- **`volume: int`**
  - The value represents the **zone's volume** (capacity).
- **`ships: list`**
  - A **list** containing **battleships** (objects) each **zone** has.
  - **Initially** set to an **empty list**.

Methods

__init__(code: str, volume: int)

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

get_ships()

- **Returns** a **list of all battleships** in the **zone**, **ordered** by **ship's hit strength descending**, then by **ship name ascending**.

zone_info()

- Returns **detailed information** about the zone. Keep in mind that **each** type of **zone implements** the method **differently**.

5. Class RoyalZone

In the **royal_zone.py** file, the class **RoyalZone** should be implemented. A **Royal Zone** is a **type** of **BaseZone**. The **Royal Zone has** an **initial volume** of **10 ships**.

Methods

__init__(code: str)

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

zone_info()

- The method **returns detailed information** about the **battle zone**, in the following format (each row on a new line):

  ```
  "@Royal Zone Statistics@
  Code: {zone_code}; Volume: {zone_current_volume}
  Battleships currently in the Royal Zone: {battleships_total_count},
  {pirateships_count} out of them are Pirate Battleships.
  #{Battleship_name1}, …, {Battleship_namen}#"
  ```

- **Order** the **ships** by **ship's hit strength descending**, then by **ship name ascending**.
- **Return** the **ship names** (if any) **surrounded by hashtags** (**'#'**), **separated by comma and space** **', '**. If there are **no ships** - **skip** the line. See the **Examples**

  Hint: You can use the **get_ships()** method.

6. Class PirateZone

In the **pirate_zone.py** file, the class **PirateZone** should be implemented. A **Pirate Zone** is a **type** of **BaseZone**. The **Pirate Zone has** an **initial volume** of **8 ships**.

Methods

__init__(code: str)

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

zone_info()

- The method **returns detailed information** about the **battle zone,** in the following format (each row on a new line):

  ```
  "@Pirate Zone Statistics@
  Code: {zone_code}; Volume: {zone_current_volume}
  Battleships currently in the Pirate Zone: {battleships_total_count},
  {royalships_count} out of them are Royal Battleships.
  #{Battleship_name1}, …, {Battleship_namen}#"
  ```

- **Order** the **ships** by **ship's hit strength descending**, then by **ship name ascending**.
- **Return** the **ship names** (if any) **surrounded by hashtags** (`'#'`), **separated by comma and space** `', '`. If there are **no ships** - **skip** the line. See the **Examples**

  Hint: You can use the `get_ships()` method.

7. BattleManager

In the **`battle_manager.py`** file, the class **`BattleManager`** should be implemented. It will **manage** the **battles** and **interactions** between **ships** in the battle **zones**.

Structure

The class should have the following attributes:

- **`zones: list`**
  - A list **containing all zones** (objects) assigned to host battles.
  - **Initially** an **empty list**.
- **`ships: list`**
  - A list **containing all battleships** (objects) intending to participate in battles.
  - **Initially** an **empty list**.

Methods

__init__()

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

add_zone(zone_type: str, zone_code: str)

The method **creates** a **zone object** of the **given type** and **code** and **adds** it to the **`zones`** collection.

- **First**, check If the **type** is a valid one, if **not valid**, **raise** an **`Exception`** with the following message:

  `"Invalid zone type!"`
  - **Valid types** of **zones** are: **`"RoyalZone"`** and **`"PirateZone"`**.
- **Then**, check if a zone with the **given code** is **already in the collection**. If such a zone **exists**, **raise** an **`Exception`** with the following message:

`"Zone already exists!"`

- Otherwise, **create** the **zone**, **add** it to the **zones list**, and **return** the following message:

  `"A zone of type {zone_type} was successfully added."`

add_battleship(ship_type: str, name: str, health: int, hit_strength: int)

The method **creates** a **ship object** of the **given type** with the given attributes and **adds** it to the **`ships`** collection. All **ship names** will be **unique**.

- **First**, check if the **ship type** is **valid**: **`'RoyalBattleship'`** or **`'PirateBattleship'`**.

  If not, **raise** an **`Exception`** with the following message:

  `"{ship_type} is an invalid type of ship!"`

- Otherwise, **create** the ship object, **add** it to the **`ships`** list, and **return** the following message:
  `"A new {ship_type} was successfully added."`

add_ship_to_zone(zone: BaseZone, ship: BaseBattleship)

The method adds the provided **ship object** to the **given zone** (object). The **zone and ship** will **always exist**.

- **First**, check if the zone has **enough volume** to **allow** the **ship to participate**. If not, **return** the following message:

  `"Zone {code} does not allow more participants!"`

- **Then**, check if the **ship's health is greater than zero**, and **if not**, **return** the message:

  `"Ship {name} is considered sunk! Participation not allowed!"`

- **Next**, check if the **ship is available**, and **if not**, **return** the message:

  `"Ship {name} is not available and could not participate!"`

- If **none of the above** is reached, the **ship can participate**:
  - **Check** the **ship type** and **compare** it with the **zone type**.
    - When a **ship** from an **enemy type** participates in an **enemy zone**, **it becomes a target:**
      - **Mark** it as **being under attack** (the **`is_attacking`** value remains **`False`**).

        An **enemy zone/ship** example: **`RoyalBattleship`** enters **`PirateZone`** or **`PirateBattleship`** enters a **`RoyalZone`**.

    - When a **ship** from the **same type** enters the **zone**, **it becomes an attacker** (will attack enemy ships):
      - **Mark** it as an **attacker** (set the **`is_attacking`** value to **`True`**).
  - **Add** the ship to the **zone's `ships` collection**.
    - Mark the **ship** as **unavailable** so it could not be added to other zones.
    - **Decrease** the **zone's volume**.
  - **Return** the following message:

    `"Ship {name} successfully participated in zone {zone_code}."`

remove_battleship(ship_name: str)

The method **removes the battleship** with the given **name** from the **battle manager ship's collection**.

- **First**, check if a ship with the given **name** exists in the **battle manager ships collection**. If not, **return** the following message:

  `"No ship with this name!"`

- **Then**, check if the ship **participates** in a **zone** (`is_available` value). If so, **return** the following message:

  `"The ship participates in zone battles! Removal is impossible!"`

- If the ship can be **removed successfully**, **remove** it from the **battle manager ships collection**, and **return** the following message:

  `"Successfully removed ship {ship_name}."`

start_battle(zone: BaseZone)

The method **initiates a battle between two of the participating battleships** in the **given zone** (always existing object):

- **First**, check if there are **at least two battleships**: **one attacker** and **one target** (enemy, being attacked), depending on their `is_attacking` value. If not, **return** the message:

  `"Not enough participants. The battle is canceled."`

- **Battle Rules** (in case there are **two ships** - **one attacker** and **one target**):
  - **Select** the **most powerful battleship** (based on **hit strength**) that **corresponds** to the **zone in type**, **marked** as **attacker** (`is_attacking=True`).

    Example: **`RoyalBattleship`** and **`RoyalZone`** **correspond** in **type**

  - The **opponent** will be the **healthiest enemy battleship** (based on **health**) that does **not correspond in type** (marked as **being attacked**).

    Example: **`PirateBattleship`** and **`RoyalZone`** do **not correspond** in **type**

  - **Perform** the **battle** using the **appropriate methods**: `attack()` and `take_damage()`.
    - The **attacking ship performs** an **attack** (`attack()` method, decreasing its **ammunition**), while the **enemy ship takes damage** (`take_damage()` method, decreasing its **health**).

- **Result**:
  - **First**, check if the enemy **ship's health drops to 0**. If so, it is **removed** from the **zone** and the **manager's ships collection**. **Return** the following message:

    `"{ship_name} lost the battle and was sunk."`

  - **Then**, check if the attacking **ship runs out of ammunition**. If so, it is **removed** from the **zone** and the **manager's ships collection**. **Return** the following message:

    `"{ship_name} ran out of ammunition and leaves."`

- Otherwise, **both ships remain** in the **zone**. **Return** the following message:

  "Both ships survived the battle."

- **Constraints:**
  - There will always be **only one ship** with **maximum hit strength** (if any) and **only one with maximum health** (if any).
  - The method performs **one battle per call**.
  - When there is **a sunk ship**, the **other will have ammunition left**. There **won't** be a **case** when **both** ships shall be **removed**.

Note: Use the **`attack()`** and **`take_damage()`** methods to perform the battle properly.

get_statistics()

The method **returns** up-to-date **statistics** for **all zones** in the **battles manager collection** and the **battleships currently available** (not participating in zones).

- **Return** the **available ship names** (if any) in their **current order**, **surrounded by hashtags** (**'#'**), **separated by comma and space ', '**. If there are **no available ships** - **skip** the line.
- **Order** zones by **zone code ascending**.
- **Return** information for **each zone**, **generated** by its **designated method** **`zone_info()`**.
- The **output string** should **contain** the **above-described information**, on new lines as follows**:**

  "Available Battleships: {available_ships_count}

  #{available_ship_name$_1$}, …, {available_ship_name$_n$}#

  *****Zones Statistics:*****

  Total Zones: {zones_count}

  {zone$_1$_info}

  …

  {zone$_n$_info}"

- **Note**: Use the zone's **`zone_info()`** method to generate the statistics properly.

Examples

Test Code

`# Initialize the BattleManager`
`battle_manager = BattleManager()`
 	
`# Add zones`
`print(battle_manager.add_zone("RoyalZone", "001"))`
`print(battle_manager.add_zone("PirateZone", "002"))`
`print()`
 	
`# Add battleships`
`print(battle_manager.add_battleship("RoyalBattleship", "RoyalOne", 50, 50))`
`print(battle_manager.add_battleship("RoyalBattleship", "RoyalTwo", 80, 45))`
`print(battle_manager.add_battleship("PirateBattleship", "PirateOne", 50, 50))`

```
print(battle_manager.add_battleship("PirateBattleship", "PirateTwo", 70, 60))
print(battle_manager.add_battleship("RoyalBattleship", "RoyalThree", 100, 100))
print(battle_manager.add_battleship("PirateBattleship", "PirateThree", 90, 90))
print()

# Retrieve battleships and zones
royal_zone = battle_manager.zones[0]
pirate_zone = battle_manager.zones[1]

royal_one = battle_manager.ships[0]
royal_two = battle_manager.ships[1]
pirate_one = battle_manager.ships[2]
pirate_two = battle_manager.ships[3]

# Add ships to zones
print(battle_manager.add_ship_to_zone(royal_zone, royal_one))
print(battle_manager.add_ship_to_zone(royal_zone, pirate_one))
print(battle_manager.add_ship_to_zone(pirate_zone, royal_two))
print(battle_manager.add_ship_to_zone(pirate_zone, pirate_two))
print()

# Remove a battleship
print(battle_manager.remove_battleship("RoyalTwo"))
print(battle_manager.remove_battleship("Nonexistent"))
print()

# Start battle in RoyalZone
print(battle_manager.start_battle(royal_zone))
print()

# Start battle in PirateZone
print(battle_manager.start_battle(pirate_zone))
print()

# Start another battle in RoyalZone
print(battle_manager.start_battle(royal_zone))
print()

# Retrieve updated statistics
print(battle_manager.get_statistics())
print()

# Remove a battleship
print(battle_manager.remove_battleship("RoyalThree"))
```

Output

```
A zone of type RoyalZone was successfully added.
A zone of type PirateZone was successfully added.

A new RoyalBattleship was successfully added.
A new RoyalBattleship was successfully added.
A new PirateBattleship was successfully added.
A new PirateBattleship was successfully added.
A new RoyalBattleship was successfully added.
A new PirateBattleship was successfully added.

Ship RoyalOne successfully participated in zone 001.
Ship PirateOne successfully participated in zone 001.
Ship RoyalTwo successfully participated in zone 002.
Ship PirateTwo successfully participated in zone 002.
```

```
The ship participates in zone battles! Removal is impossible!
No ship with this name!

PirateOne lost the battle and was sunk.

Both ships survived the battle.

Not enough participants. The battle is canceled.

Available Battleships: 2
#RoyalThree, PirateThree#
***Zones Statistics:***
Total Zones: 2
@Royal Zone Statistics@
Code: 001; Volume: 8
Battleships currently in the Royal Zone: 1, 0 out of them are Pirate Battleships.
#RoyalOne#
@Pirate Zone Statistics@
Code: 002; Volume: 6
Battleships currently in the Pirate Zone: 2, 1 out of them are Royal Battleships.
#PirateTwo, RoyalTwo#

Successfully removed ship RoyalThree.
```

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 **`Furniture`** 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.