OOP Problems

Question Created By: Ahammad Nafiz
Github:
https://github.com/ahammadnafiz
Linkedin:
https://www.linkedin.com/in/ahammad-nafiz/
Personal Blog:
https://ahammadnafiz.github.io/

Problem 1: Library Management System

Design a simple library management system with the following classes:

1. **Book**:
   - Attributes: `title`, `author`, `ISBN`, `available` (boolean)
   - Methods: `check_out()`, `return_book()`
2. **Library**:
   - Attributes: `name`, `books` (list of Book objects)
   - Methods: `add_book(book)`, `remove_book(book)`, `find_book(title)`
3. **Member**:
   - Attributes: `name`, `member_id`, `borrowed_books` (list of Book objects)
   - Methods: `borrow_book(book)`, `return_book(book)`

Create a scenario that demonstrates:

- Creating multiple Book objects
- Adding books to the Library

- Creating Member objects
- Members borrowing and returning books
- Handling a case where a book is not available

Problem 2: Bank Account System

Implement a bank account system with the following classes:

1. **Account**:
   - Attributes: `account_number`, `balance`, `account_type`
   - Methods: `deposit(amount)`, `withdraw(amount)`, `display_balance()`
2. **SavingsAccount**:
   - Attributes: `account_number`, `balance`, `interest_rate`
   - Methods: `deposit(amount)`, `withdraw(amount)`, `display_balance()`, `add_interest()`
3. **CheckingAccount**:
   - Attributes: `account_number`, `balance`, `overdraft_limit`
   - Methods: `deposit(amount)`, `withdraw(amount)`, `display_balance()`
4. **Bank**:
   - Attributes: `name`, `accounts` (list of Account objects)
   - Methods: `add_account(account)`, `remove_account(account)`, `find_account(account_number)`
5. **Customer**:
   - Attributes: `name`, `accounts` (list of Account objects)
   - Methods: `open_account(account)`, `close_account(account)`

Demonstrate:

- Creating different types of accounts
- Performing transactions (deposits, withdrawals)
- Adding interest to savings accounts
- Handling overdrafts on checking accounts

Problem 3: Online Shopping Cart

Design an online shopping system focusing on the cart functionality:

1. **Product**:
   - Attributes: `name`, `price`, `stock`
   - Methods: `update_stock(amount)`
2. **CartItem**:
   - Attributes: `product` (Product object), `quantity`
   - Methods: `update_quantity(new_quantity)`
3. **ShoppingCart**:
   - Attributes: `items` (list of CartItem objects)
   - Methods: `add_item(product, quantity)`, `remove_item(product)`, `update_item_quantity(product, new_quantity)`, `get_total()`
   - Implement a `deep_copy()` method that creates a new ShoppingCart with new CartItem objects
4. **Customer**:
   - Attributes: `name`, `email`, `cart` (ShoppingCart object)
   - Methods: `add_to_cart(product, quantity)`, `remove_from_cart(product)`, `checkout()`

Demonstrate:

- Adding products to the cart
- Updating quantities
- Removing items from the cart
- Creating a deep copy of the cart
- Show how modifying the original cart doesn't affect the copy

Problem 4: Task Management System

Create a task management system with the following classes:

1. **Task**:

- Attributes: `title`, `description`, `status` (e.g., "To Do", "In Progress", "Done"), `assigned_to` (User object)
- Methods: `update_status(new_status)`, `assign(user)`

2. **User**:
   - Attributes: `name`, `email`, `tasks` (list of Task objects)
   - Methods: `assign_task(task)`, `complete_task(task)`
3. **Project**:
   - Attributes: `name`, `tasks` (list of Task objects), `team` (list of User objects)
   - Methods: `add_task(task)`, `remove_task(task)`, `add_team_member(user)`, `remove_team_member(user)`
4. **TaskManager**:
   - Attributes: `projects` (list of Project objects)
   - Methods: `create_project(name)`, `assign_task(task, user)`, `get_user_tasks(user)`

Demonstrate:

- Creating projects, tasks, and users
- Assigning tasks to users
- Updating task statuses
- Showing how tasks are shared between User and Project objects

E-commerce System: "GreenGrocer Online"

Scenario

GreenGrocer is a local produce store creating an online ordering system. They need a simple e-commerce platform that demonstrates key OOP concepts while handling basic requirements for fresh produce sales.

Classes and Their Functionalities

1. Product Class

- **Attributes**:
  - `name`

    - `price`
    - `stock`
    - `category`
- **Methods**:
    - `update_stock(amount)`: Updates the stock
    - `apply_discount(percentage)`: Applies a discount to the price

2. ShoppingCart Class

- **Attributes**:
    - `items` (dictionary of Product objects and quantities)
- **Methods**:
    - `add_item(product, quantity)`: Adds a product to the cart
    - `remove_item(product)`: Removes a product from the cart
    - `get_total()`: Calculates the total price of items in the cart
    - `create_deep_copy()`: Creates and returns a deep copy of the shopping cart

3. User Class

- **Attributes**:
    - `username`
    - `email`
    - `shopping_cart` (ShoppingCart object)
- **Methods**:
    - `add_to_cart(product, quantity)`: Adds a product to the user's shopping cart
    - `checkout()`: Processes the order and clears the cart

4. Order Class

- **Attributes**:
    - `user` (User object)
    - `items` (dictionary of Product objects and quantities)
    - `total_price`

- **Methods**:
  - `generate_invoice()`: Returns a string representation of the order

5. InventoryManager Class

- **Attributes**:
  - `products` (list of Product objects)
- **Methods**:
  - `add_product(product)`: Adds a new product to the inventory
  - `remove_product(product)`: Removes a product from the inventory
  - `get_product_by_name(name)`: Returns a product object by its name

6. EcommerceSystem Class

- **Attributes**:
  - `inventory_manager` (InventoryManager object)
  - `users` (list of User objects)
  - `orders` (list of Order objects)
- **Methods**:
  - `register_user(username, email)`: Creates and returns a new User
  - `process_order(user)`: Creates an Order from the user's cart and adds it to the orders list

Implementation Guide for "GreenGrocer Online" E-commerce System

1. Define Classes and Attributes

Product Class

- **Attributes**: Define attributes such as `name`, `price`, `stock`, and `category`.
- **Methods**: Implement methods like `update_stock(amount)` to adjust stock levels and `apply_discount(percentage)` to modify the price.

ShoppingCart Class

- **Attributes**: Design `items` as a dictionary to store Product objects and their quantities.
- **Methods**: Include methods like `add_item(product, quantity)`, `remove_item(product)`, `get_total()` to manage cart contents and calculate the total price.
- **Advanced Functionality**: Implement `create_deep_copy()` for creating a copy of the shopping cart to handle transactional states.

User Class

- **Attributes**: Specify `username`, `email`, and `shopping_cart` as a ShoppingCart object.
- **Methods**: Develop `add_to_cart(product, quantity)` to add items to the user's shopping cart and `checkout()` to process the order and clear the cart.

Order Class

- **Attributes**: Define `user`, `items` (dictionary of Product objects and quantities), and `total_price`.
- **Methods**: Include `generate_invoice()` to provide a formatted string representation of the order details.

InventoryManager Class

- **Attributes**: Utilize `products` as a list of Product objects.
- **Methods**: Implement `add_product(product)`, `remove_product(product)`, and `get_product_by_name(name)` to manage the inventory.

EcommerceSystem Class

- **Attributes**: Initialize `inventory_manager` as an InventoryManager object, `users` as a list of User objects, and `orders` as a list of Order objects.
- **Methods**: Provide `register_user(username, email)` to create and return a new User object and `process_order(user)` to facilitate order creation and management.

2. Establish Class Interactions

- **Product and Inventory Management**: Ensure InventoryManager class manages Product objects effectively, supporting addition, removal, and retrieval operations.

- **User Interaction**: Implement methods in User class to interact with ShoppingCart, facilitating item addition, removal, and order processing.
- **Order Processing**: Develop methods in EcommerceSystem to register users, manage inventory, process orders, and maintain order history.

Dungeon Explorer: Combat Edition

Classes and Functions

Item Class:

- **Attributes**:
  - `name`: Name of the item.
  - `description`: Description of the item.
  - `durability`: Durability of the item.
- **Methods**:
  - `use()`: Reduces the item's durability and provides feedback on usage.
  - `create_copy()`: Creates a duplicate of the item.

Weapon Class :

- **Additional Attributes**:
  - `damage`: Amount of damage the weapon inflicts.
- **Additional Methods**:
  - `use()`: Reduces durability and returns the damage value inflicted.
  - `create_copy()`: Creates a duplicate of the weapon.

Room Class:

- **Attributes**:
  - `name`: Name of the room.
  - `description`: Description of the room.
  - `items`: List of items present in the room.

- `connected_rooms`: Dictionary mapping directions to connected rooms (`{"north": room1, "east": room2, ...}`).
- `enemies`: List of enemies present in the room.
- `likes`: A count or rating representing how much the player likes the room.

- **Methods**:
  - `add_item(item)`, `remove_item(item)`: Add or remove items from the room.
  - `connect_room(direction, room)`: Connect a neighboring room in a specified direction.
  - `add_enemy(enemy)`, `remove_enemy(enemy)`: Add or remove enemies from the room.
  - `like()`: Increases the likes count of the room.
  - `get_copy()`: Creates a copy of the room and its contents.

Player Class:

- **Attributes**:
  - `name`: Name of the player.
  - `inventory`: List of items the player carries.
  - `current_room`: The room where the player is currently located.
  - `health`, `max_health`: Current and maximum health of the player.
- **Methods**:
  - `pick_up_item(item_name)`, `drop_item(item_name)`: Add or remove items from the player's inventory.
  - `move(direction)`: Move the player to a connected room in the specified direction.
  - `use_item(item_name)`: Use an item from the player's inventory.
  - `attack(enemy_name)`: Attack an enemy in the current room.
  - `take_damage(amount)`: Decrease player's health by a specified amount.
  - `heal(amount)`: Increase player's health by a specified amount.

Enemy Class:

- **Attributes**:
  - `name`: Name of the enemy.
  - `health`, `max_health`: Current and maximum health of the enemy.
  - `weapon`: The weapon the enemy uses for attacks.
- **Methods**:
  - `attack(player)`: Initiates an attack on the player.
  - `take_damage(amount)`: Decreases enemy's health by a specified amount.

Instructions for Implementing Dungeon Explorer: Combat Edition

Welcome, developer! Below are step-by-step instructions to guide you through the process of implementing the Dungeon Explorer: Combat Edition game. Follow these guidelines to structure your solution effectively:

1. Class Design

- **Define Classes:** Implement the necessary classes (`Item`, `Weapon`, `Room`, `Player`, `Enemy`) with appropriate attributes and methods as outlined in the problem statement.
- **Consider Composition:** Use composition where necessary (e.g., `Room` containing lists of `items`, `enemies`, and a dictionary of `connected_rooms`).

2. Class Implementation

- **Item and Weapon Classes:**
  - Implement methods for `use()` to simulate item usage and reduce durability.
  - Ensure `create_copy()` method creates duplicates of items or weapons.
- **Room Class:**
  - Implement methods to add/remove items and enemies, connect rooms, and manage room likes.
  - Create a method `get_copy()` to replicate the room and its contents for game state management.

- **Player Class:**
  - Implement methods for picking up and dropping items, moving between rooms, using items, attacking enemies, and managing health.
  - Ensure health management (`take_damage()`, `heal()`) is correctly implemented based on game mechanics.
- **Enemy Class:**
  - Implement methods for enemy attack (`attack()`) and receiving damage (`take_damage()`).