Scenario: E-Commerce Platform

You are building an e-commerce platform where users can browse products, add them to a cart, apply discounts, and place orders. The system should be modular, extensible, and maintainable.

Goal 1: Single Responsibility Principle (SRP)

- **Task:** Create a `Product` class that only handles product-related attributes (e.g., `productId`, `name`, `price`).
- **Deliverables:**
  - A `Product` class with methods to get and set product details.
  - A separate `ProductPrinter` class to display product information.
- **Objective:** Understand how SRP ensures that a class has only one reason to change.

Goal 2: Open/Closed Principle (OCP)

- **Task:** Implement a discount system where new discount types can be added without modifying existing code.
- **Deliverables:**
  - An interface `Discount` with a method `applyDiscount(double price)`.
  - Two classes `FlatDiscount` and `PercentageDiscount` implementing the `Discount` interface.
  - A `Product` class that can apply any discount type without modifying its code.
- **Objective:** Learn how OCP allows extending functionality without changing existing code.

Goal 3: Liskov Substitution Principle (LSP)

- **Task:** Create a hierarchy of payment methods (e.g., `CreditCard`, `PayPal`) that can be used interchangeably.
- **Deliverables:**
  - A base class `PaymentMethod` with a method `pay(double amount)`.
  - Two subclasses `CreditCard` and `PayPal` that override the `pay` method.
  - A `PaymentProcessor` class that can process payments using any `PaymentMethod` subclass.
- **Objective:** Understand how LSP ensures that subclasses can replace their base class without breaking functionality.

Goal 4: Interface Segregation Principle (ISP)

- **Task:** Design interfaces for user roles (e.g., `Customer`, `Admin`) to ensure they only have relevant methods.
- **Deliverables:**
  - An interface `Customer` with methods like `viewProducts()` and `addToCart()`.
  - An interface `Admin` with methods like `addProduct()` and `removeProduct()`.
  - A class `User` that implements only the interfaces relevant to their role.
- **Objective:** Learn how ISP prevents classes from implementing unnecessary methods.

Goal 5: Dependency Inversion Principle (DIP)

- **Task:** Implement a notification system where the e-commerce platform can send notifications via email or SMS without tightly coupling the classes.
- **Deliverables:**
  - An interface `NotificationService` with a method `sendNotification(String message)`.
  - Two classes `EmailNotification` and `SmsNotification` implementing the `NotificationService` interface.
  - A `OrderService` class that depends on the `NotificationService` interface, not concrete implementations.
- **Objective:** Understand how DIP promotes loose coupling and easier testing.

Goal 6: Design Pattern - Singleton

- **Task:** Ensure that the e-commerce platform's `Cart` class is a singleton, as each user should have only one cart.
- **Deliverables:**
  - A `Cart` class with a private constructor and a static method `getInstance()`.
  - A demonstration of how multiple calls to `getInstance()` return the same `Cart` object.
- **Objective:** Learn how the Singleton pattern ensures a single instance of a class.

Goal 7: Design Pattern - Observer

- **Task:** Implement a notification system where users are notified when a product they are interested in is back in stock.
- **Deliverables:**
  - A `Product` class that maintains a list of `Observer` objects.
  - An `Observer` interface with a method `update(String productName)`.
  - A `User` class that implements the `Observer` interface and receives updates.
  - A demonstration of how users are notified when a product's stock status changes.
- **Objective:** Understand how the Observer pattern facilitates event-driven communication.

Collaborative Task

- **Task:** Combine all the individual tasks into a single e-commerce platform.
- **Deliverables:**
  - A working e-commerce platform with:
    - Products managed by `Product` class.
    - Discounts applied using `Discount` interface.
    - Payments processed using `PaymentMethod` hierarchy.
    - User roles defined by `Customer` and `Admin` interfaces.
    - Notifications sent via `NotificationService` interface.
    - A singleton `Cart` for each user.
    - Users notified of stock updates using the Observer pattern.
- **Objective:** Learn how SOLID principles and design patterns work together to create a modular and maintainable system.

Timeline

1. **Session 1:** Each intern works on their individual task.
2. **Session 2:** Interns collaborate to integrate their tasks into a single platform.
3. **Session 3:** Code review, testing, and final presentation.

Learning Outcomes

- Understanding of SOLID principles and their importance in software design.
- Hands-on experience with common design patterns (Singleton, Observer).
- Improved collaboration and problem-solving skills.

- Ability to apply theoretical concepts to real-world scenarios.

This task plan ensures that each intern gains a deep understanding of SOLID principles and design patterns while contributing to a cohesive project.