**Exercise No. 5**

You have been hired as a senior software architect for a fintech company developing a next-generation stock trading and portfolio management platform. The platform must simulate and support both retail investors and institutional clients, each with different trading privileges, execution priorities, and risk profiles. Your task is to design a robust object-oriented system that reflects real-world market behavior and trading rules while remaining scalable and extensible.

In this system, users will own accounts that can place various types of stock orders—such as market orders, limit orders, and stop-loss orders—on a simulated stock exchange. These accounts can be of two main types: RetailAccount, which faces restrictions like lower trade volumes and standard execution priority, and InstitutionalAccount, which enjoys benefits like faster execution and access to premium trading routes.

Each user maintains a portfolio that tracks owned assets (stock symbol, quantity), available cash, and overall portfolio value. Orders placed by accounts interact with a Trade Engine, which matches, executes, or cancels them based on current market prices. Market prices are fed in real-time by a Market Data Feed, which updates stock prices and notifies all active orders when a relevant change occurs. This allows the system to automatically trigger executions for conditions like stop-loss or limit orders.

The platform must also support the simulation of market events such as earnings reports, stock splits, and dividend announcements, which can affect stock valuations and portfolio performance. Each of these events should be implemented using an extensible event-handling system. For advanced users, the system should allow the implementation of custom trading strategies, and accounts should be able to execute trades based on predefined logic. Risk calculations, such as total exposure to a particular sector or stock, should also be available for every account.

The architecture must adhere to solid OOP principles, making heavy use of abstraction, encapsulation, inheritance, and polymorphism, as well as design patterns such as strategy (for trading behavior) and observer (for order-market updates). Your design should prioritize modularity, testability, and future extensibility—for example, adding features like margin trading or derivatives in the future.

**Exercise No. 6**

You are designing an object-oriented system for an **online bookstore** that sells physical books and eBooks. The system must handle customers, orders, book inventory, and digital licenses. Customers can browse books, place orders, and download eBooks if purchased. Books have common attributes like title, author, and price, but physical books also include shipping weight and stock, while eBooks include file format and download size.

Orders can consist of multiple items and should calculate the total cost, including tax and optional shipping fees. The system should support different account types (e.g., regular and premium customers), each with different discount policies. A notification system should alert customers when their order is shipped or available for download.

Design this system using solid OOP principles: **inheritance** (Book → PhysicalBook, EBook), **polymorphism** (custom discount logic), **encapsulation** (order data), and **abstraction** (interfaces for payment and notifications).