**Курс:** Шаблоны проектирования приложений

**Тема:** Модуль 07 Паттерны поведения. Стратегия. Наблюдатель

**Задача:**

Необходимо реализовать приложение на C#, которое моделирует поведение разных типов оплаты (например, банковская карта, PayPal, криптовалюта и т.д.) с использованием порождающего паттерна **Стратегия**.

Цель — предоставить гибкий способ замены алгоритмов оплаты без изменения кода клиентских классов.

**Описание:**

1. **Контекст**: Контекст будет представлять собой класс, который выполняет процесс оплаты, используя различные стратегии.
2. **Стратегии**: Каждая стратегия будет представлять собой отдельную реализацию метода оплаты, например, оплата банковской картой, через PayPal или криптовалютой.
3. **Клиент**: Клиентский код должен иметь возможность переключать стратегии оплаты по необходимости.

**Шаги выполнения:**

1. **Создайте интерфейс** IPaymentStrategy, который будет определять метод Pay
2. **Создайте несколько классов-стратегий**, которые реализуют интерфейс IPaymentStrategy, например, стратегии для оплаты банковской картой, через PayPal и с помощью криптовалюты.
3. **Создайте класс контекста** PaymentContext, который будет работать с разными стратегиями оплаты
4. **Напишите клиентский код**, который будет задавать различные стратегии оплаты и выполнять их

**Требования к выполнению:**

* Реализуйте минимум три различных стратегии оплаты.
* Реализуйте возможность выбора стратегии оплаты в зависимости от входных данных пользователя.
* При выполнении программы должна быть возможность легко переключаться между разными способами оплаты без изменения структуры программы.

**Задача:**

Необходимо реализовать приложение на C#, которое демонстрирует работу паттерна **Наблюдатель (Observer)**. Приложение будет моделировать ситуацию, когда разные подписчики получают уведомления от одного субъекта, например, обновления курсов валют или изменение цен на акции.

**Описание:**

1. **Субъект**: Субъект будет представлять объект, за состоянием которого наблюдают другие объекты. При изменении состояния субъекта, он уведомляет всех своих подписчиков.
2. **Наблюдатели**: Наблюдатели — это объекты, которые подписаны на изменения состояния субъекта. Каждый наблюдатель реагирует на изменения по-своему.
3. **Клиент**: Клиентский код должен позволять добавлять и удалять наблюдателей, а также обновлять их при изменении состояния субъекта.

**Шаги выполнения:**

1. **Создайте интерфейс** IObserver, который будет представлять наблюдателя
2. **Создайте интерфейс** ISubject, который будет представлять субъекта
3. **Реализуйте класс субъекта** CurrencyExchange, который хранит информацию о курсах валют и уведомляет наблюдателей об изменении курсов
4. **Создайте несколько классов-наблюдателей**, которые реализуют интерфейс IObserver. Каждый наблюдатель будет получать обновления и реагировать по-своему
5. **Напишите клиентский код**, который будет взаимодействовать с субъектом и наблюдателями

**Требования к выполнению:**

* Реализуйте минимум три разных наблюдателя, каждый из которых по-своему обрабатывает обновления.
* Обеспечьте возможность добавления и удаления наблюдателей.
* Реализуйте уведомление наблюдателей при изменении состояния субъекта.