**Курс:** Шаблоны проектирования приложений

**Тема:** Модуль 09 Структурные паттерны. Декоратор. Адаптер

**Задача:**

Необходимо разработать систему управления заказами в кафе, где различные добавки (например, молоко, сироп, сливки и т.д.) могут быть добавлены к базовому напитку (например, кофе). Используя паттерн "Декоратор", создайте архитектуру, которая позволяет динамически добавлять новые "декорации" (добавки) к напиткам.

**Требования:**

1. Создайте базовый интерфейс или абстрактный класс Beverage, который будет описывать поведение напитка.
2. Создайте несколько классов напитков, реализующих интерфейс Beverage (например, Espresso, Tea).
3. Реализуйте абстрактный класс-декоратор BeverageDecorator, который наследуется от Beverage и содержит ссылку на объект Beverage. Этот класс должен делегировать вызовы методов объекту, который декорируется.
4. Создайте конкретные классы-декораторы для различных добавок (например, Milk, Sugar, WhippedCream), которые наследуются от BeverageDecorator.
5. Каждый декоратор должен изменять поведение метода Cost() для расчета стоимости напитка с учетом добавки.
6. Напишите клиентский код, в котором создается объект напитка, к которому последовательно добавляются различные декораторы (добавки). Выведите в консоль итоговое описание и стоимость напитка.

**Ожидаемый результат:**

* Программа должна позволять гибко добавлять любое количество добавок к базовому напитку.
* Должна корректно рассчитываться итоговая стоимость напитка с учетом всех добавок.

**Задания:**

1. Реализуйте предложенную архитектуру.
2. Добавьте несколько новых типов напитков (например, Latte, Mocha).
3. Добавьте несколько новых типов добавок.
4. Протестируйте систему с различными комбинациями напитков и добавок.

**Задача:**

Вам необходимо интегрировать стороннюю систему оплаты в уже существующую систему интернет-магазина. Однако интерфейс сторонней системы отличается от интерфейса, используемого в вашем приложении. Ваша задача — применить паттерн "Адаптер" для решения этой проблемы.

**Требования:**

1. **Существующая система:**
   * Есть интерфейс IPaymentProcessor, который используется для обработки платежей в интернет-магазине.
   * В проекте уже реализован класс PayPalPaymentProcessor, который обрабатывает платежи через PayPal.
2. **Сторонняя система:**
   * Сторонний процессор платежей называется StripePaymentService, у которого другой интерфейс для проведения транзакций.
3. **Задача:**
   * Необходимо адаптировать StripePaymentService к интерфейсу IPaymentProcessor с помощью паттерна "Адаптер".
   * Реализовать класс StripePaymentAdapter, который позволит использовать StripePaymentService как часть существующей системы.

**Структура задания:**

1. Создайте интерфейс IPaymentProcessor, который содержит метод ProcessPayment(double amount).
2. Реализуйте класс PayPalPaymentProcessor, который будет реализовывать интерфейс IPaymentProcessor и симулировать процесс оплаты.
3. Создайте сторонний класс StripePaymentService, у которого есть метод MakeTransaction(double totalAmount).
4. Реализуйте класс адаптера StripePaymentAdapter, который будет реализовывать интерфейс IPaymentProcessor и адаптировать вызов метода MakeTransaction из StripePaymentService к интерфейсу IPaymentProcessor.
5. Напишите клиентский код, который может использовать оба процессора (PayPal и Stripe) через интерфейс IPaymentProcessor.

**Ожидаемый результат:**

* В клиентском коде вы должны иметь возможность работать с обеими системами оплаты через единый интерфейс IPaymentProcessor, используя либо PayPalPaymentProcessor, либо StripePaymentAdapter.

**Задания:**

1. Реализуйте предложенную архитектуру, используя паттерн "Адаптер".
2. Добавьте в проект еще один сторонний платежный сервис с другим интерфейсом и создайте для него адаптер.
3. Протестируйте работу системы с обоими платежными процессорами.