**Отчет по использованным паттернам**

В процессе разработки системы для интернет-магазина были использованы несколько паттернов проектирования для обеспечения гибкости, расширяемости и упрощения кода. Каждый паттерн был выбран в зависимости от специфики задачи, что позволило эффективно решать поставленные цели.  
  
1. Адаптер ("Adapter") — Структурный паттерн ("Structural pattern"):  
 - Паттерн Адаптер был использован для интеграции различных платёжных систем (например, PayPal и Credit Card) в систему.  
 - Этот паттерн позволил создать единый интерфейс для разных платёжных систем, обеспечив возможность легко добавлять новые способы оплаты в будущем без изменения основной логики системы.  
 - В частности, был создан абстрактный класс PaymentAdapter, который реализуют конкретные адаптеры, такие как PayPalPaymentAdapter и CreditCardPaymentAdapter.  
  
2. Фабричный метод ("Factory Method") — Порождающий паттерн ("Creational pattern"):  
 - Паттерн Фабричный метод был применён для создания различных типов уведомлений (например, email, SMS).  
 - Этот паттерн позволяет легко добавлять новые типы уведомлений без изменения основной логики создания уведомлений в системе.  
 - В системе был использован класс NotificationFactory, который предоставляет метод create\_notification, создающий соответствующий объект уведомления в зависимости от типа (email или SMS).  
  
3. Цепочка обязанностей ("Chain of Responsibility") — Поведенческий паттерн ("Behavioral pattern"):  
 - Паттерн Цепочка обязанностей использовался для организации обработки заказов, где каждый этап обработки (проверка наличия товара, оплата, отправка) обрабатывается отдельным объектом в цепочке.  
 - Это позволяет гибко управлять процессом обработки и добавлять новые этапы без изменений в существующем коде.  
 - В системе был использован класс OrderProcessingChain, который связывает несколько обработчиков заказов, таких как PaymentHandler, StockHandler и ShippingHandler.  
  
4. Стратегия ("Strategy") — Поведенческий паттерн ("Behavioral pattern"):  
 - Паттерн Стратегия был использован для реализации различных способов оплаты. Каждый платёжный метод (PayPal, Credit Card) является отдельной стратегией для обработки платежей.  
 - Это позволяет легко добавлять новые платёжные методы, не изменяя код, который использует эти методы.  
 - Например, были реализованы классы PayPalPaymentAdapter и CreditCardPaymentAdapter, которые инкапсулируют логику обработки платежей для соответствующих платёжных систем.  
  
5. Одиночка ("Singleton") — Порождающий паттерн ("Creational pattern"):  
 - Паттерн Одиночка был использован для создания уникального экземпляра объекта, например, OrderFactory.  
 - Этот паттерн гарантирует, что будет только один экземпляр класса и предоставляет глобальную точку доступа к этому экземпляру.  
 - В системе был использован класс OrderFactory, который был реализован как одиночка для создания объектов заказов.  
  
6. Декоратор ("Decorator") — Структурный паттерн ("Structural pattern"):  
 - Паттерн Декоратор позволяет динамически добавлять новые функциональные возможности объектам без изменения их структуры.  
 - В данной системе, например, можно было бы использовать декораторы для добавления новых функциональностей к уведомлениям, например, добавление логирования или других проверок перед отправкой.  
 - В текущей реализации системы декораторы могут быть использованы для расширения функциональности уведомлений, добавляя дополнительные шаги перед отправкой уведомлений.  
  
7. Компоновщик ("Composite") — Структурный паттерн ("Structural pattern"):  
 - Паттерн Компоновщик используется для работы с объектами, которые могут быть как простыми, так и сложными (состоящими из других объектов). Он позволяет обрабатывать группы объектов одинаково, как и одиночные объекты.  
 - В данном случае, Компоновщик можно было бы использовать для обработки пакетов товаров, где один товар — это одиночный объект, а несколько товаров — это составной объект (например, бандл).  
 - Паттерн позволяет легко управлять группами товаров и их ценами, как единым целым.  
  
Каждый из этих паттернов был использован для того, чтобы решить конкретные задачи в системе:  
- Адаптер обеспечил единый интерфейс для работы с различными платёжными системами.  
- Фабричный метод упростил создание объектов уведомлений в зависимости от их типа.  
- Цепочка обязанностей позволила элегантно организовать обработку заказа с разными этапами.  
- Стратегия позволила эффективно добавлять новые платёжные методы, не изменяя существующий код.  
- Одиночка гарантировал, что фабрика заказов имеет только один экземпляр в системе.  
- Декоратор и Компоновщик могут быть использованы для расширения системы с учётом добавления новых функциональностей и группировки объектов.  
  
Эти паттерны улучшили архитектуру системы, обеспечив её гибкость, расширяемость и возможность легкого добавления новых функций.