**Курс:** Шаблоны проектирования приложений

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

*Баллы:*

**Задача:**

Реализуйте систему управления отчетностью для интернет-магазина с применением паттерна "Декоратор". Система должна поддерживать генерацию различных отчетов по продажам и пользователям с возможностью добавления дополнительных функциональных возможностей, таких как фильтрация по датам, сортировка по определенным критериям, экспорт данных в различные форматы (например, в CSV и PDF). В основе должна быть гибкая архитектура, которая позволит динамически добавлять новые "декорации" без изменения исходного кода базовых классов.

**Структура работы:**

**1. Базовые требования:**

* **Интерфейс IReport:** Интерфейс для генерации отчетов, содержащий метод Generate(), возвращающий строку с данными отчета.
* **Классы отчетов:**
  + SalesReport — класс, представляющий отчет по продажам.
  + UserReport — класс, представляющий отчет по пользователям.

**2. Декораторы:**

* **Фильтр по датам:** Декоратор DateFilterDecorator добавляет фильтрацию данных по указанному диапазону дат.
* **Сортировка данных:** Декоратор SortingDecorator добавляет возможность сортировки данных по определенному критерию (например, по дате или сумме продажи).
* **Экспорт:**
  + Декоратор CsvExportDecorator добавляет возможность экспорта отчета в формат CSV.
  + Декоратор PdfExportDecorator добавляет возможность экспорта отчета в PDF.

**3. Дополнительные требования:**

* Добавьте возможность применения нескольких декораторов к одному отчету.
* Реализуйте клиентский код, который создает отчет и применяет к нему декораторы в зависимости от требований пользователя.
* Генерируйте данные отчетов с фиктивной информацией для демонстрации работы.

**Детали реализации:**

1. **Интерфейс для отчетов:** Создайте интерфейс IReport с методом Generate(), который возвращает строку.
2. **Классы отчетов:** Реализуйте классы отчетов, такие как SalesReport и UserReport, которые наследуют IReport и возвращают соответствующие данные.
3. **Абстрактный декоратор:** Создайте абстрактный класс ReportDecorator, который наследует IReport и содержит ссылку на базовый отчет.
4. Декораторы: Фильтрация по датам, Сортировка данных, Экспорт в CSV, Экспорт в PDF
5. Клиентский код

**Задания:**

1. Реализуйте предложенную архитектуру.
2. Добавьте новые декораторы, такие как фильтрация по сумме продаж или пользователям с определенными характеристиками.
3. Создайте механизм динамического выбора декораторов в зависимости от пользовательских запросов.
4. Протестируйте систему с различными комбинациями отчетов и декораторов.

**Задача:**

Вам необходимо разработать систему для мониторинга логистики и обработки заказов в крупной сети складов. На текущий момент система поддерживает взаимодействие с внутренней службой доставки. В то же время необходимо интегрировать сторонние службы логистики, каждая из которых имеет свой собственный интерфейс. Для решения этой задачи необходимо использовать паттерн "Адаптер".

**Описание:**

1. В системе есть существующий интерфейс для внутренней службы доставки:
   * IInternalDeliveryService, который реализован в классе InternalDeliveryService.
2. Вам нужно интегрировать несколько сторонних логистических служб:
   * Сторонние службы логистики имеют собственные интерфейсы и методы, такие как ExternalLogisticsServiceA и ExternalLogisticsServiceB.
3. Вы должны разработать адаптеры для каждой из сторонних служб:
   * Адаптеры должны преобразовывать интерфейсы внешних служб логистики к единому интерфейсу внутренней системы.
4. Кроме того, необходимо создать фабрику, которая будет динамически возвращать нужный тип службы доставки в зависимости от требований пользователя (внутренняя или внешняя служба).

**Структура работы:**

**1. Основные компоненты:**

* **Интерфейс IInternalDeliveryService:** Интерфейс для внутренней службы доставки с методами DeliverOrder(string orderId) и GetDeliveryStatus(string orderId).
* **Класс InternalDeliveryService:** Класс, который реализует интерфейс IInternalDeliveryService и симулирует процесс доставки заказа.

**2. Сторонние службы логистики:**

* **ExternalLogisticsServiceA:** Сторонняя служба, которая имеет метод ShipItem(int itemId) и TrackShipment(int shipmentId) для отслеживания отправки.
* **ExternalLogisticsServiceB:** Сторонняя служба с методами SendPackage(string packageInfo) и CheckPackageStatus(string trackingCode).

**3. Адаптеры:**

* **LogisticsAdapterA:** Адаптер, который позволяет интегрировать ExternalLogisticsServiceA в систему через интерфейс IInternalDeliveryService.
* **LogisticsAdapterB:** Адаптер для интеграции ExternalLogisticsServiceB.

**4. Фабрика:**

* **DeliveryServiceFactory:** Фабрика, которая возвращает нужный тип службы доставки (внутреннюю или стороннюю) в зависимости от пользовательских требований.

**Детали реализации:**

1. **Интерфейс и класс для внутренней службы доставки**
2. Сторонние логистические службы
3. Адаптеры для внешних служб логистики: Адаптер для ExternalLogisticsServiceA, Адаптер для ExternalLogisticsServiceB
4. Фабрика для выбора службы доставки
5. Клиентский код

**Задания:**

1. Реализуйте предложенную архитектуру, используя паттерн "Адаптер".
2. Добавьте еще одну стороннюю логистическую службу с уникальным интерфейсом и создайте для нее адаптер.
3. Расширьте функционал фабрики для обработки дополнительных типов служб доставки.
4. Реализуйте обработку ошибок и логгирование процесса доставки в адаптерах.
5. Добавьте функционал для расчета стоимости доставки, который также должен поддерживаться через адаптеры.