**Курс:** Шаблоны проектирования приложений

**Тема:** Модуль 08 Паттерны поведения. Команда. Шаблонный метод. Посредник

*Баллы:*

**Цель:**

Изучить и углубленно проработать паттерн **Команда (Command)** на языке C#. Ваша задача — разработать систему "Умный дом", которая поддерживает несколько типов устройств, отмену и повтор команд, а также макрокоманды (группы команд, выполняемые последовательно).

**Описание задачи:**

Вам предстоит разработать систему управления умным домом, где различные устройства (свет, кондиционер, телевизор и т.д.) управляются через интерфейс команд. Программа должна поддерживать:

* Выполнение отдельных команд (например, включение/выключение устройства).
* Возможность отмены последней команды и повтор последней отмененной команды.
* Макрокоманды — выполнение нескольких команд за один раз.
* Возможность назначения разных команд на виртуальные кнопки пульта (клавиши).

**Структура программы:**

1. **Интерфейс ICommand** — общий интерфейс для всех команд.
2. **Классы устройств** — реализация действий для различных устройств.
3. **Классы команд** — инкапсуляция действий для каждого устройства.
4. **Класс RemoteControl** — объект, управляющий выполнением команд и их отменой.
5. **Макрокоманда** — команда, выполняющая несколько других команд.
6. **Клиентский код** — демонстрация всех возможностей системы.

**Шаги выполнения:**

1. Создайте интерфейс ICommand, который будет реализован всеми командами
2. Реализуйте несколько устройств (например, свет, телевизор, кондиционер)
3. Реализуйте конкретные команды для управления этими устройствами
   1. Команды для управления светом
   2. Команды для управления кондиционером
   3. Команды для управления телевизором
4. Реализуйте класс **RemoteControl**, который будет хранить команды и управлять их выполнением и отменой
5. Реализуйте **макрокоманды**, которые позволяют выполнять несколько команд за один вызов
6. Напишите клиентский код для демонстрации всех возможностей

**Задания:**

1. **Реализуйте код по шагам выше.**
2. **Тестирование:**
   * **Проверьте работу всех команд для отдельных устройств.**
   * **Убедитесь, что функция отмены работает корректно.**
   * **Проверьте работу макрокоманд.**
3. **Расширение функционала:**
   * **Добавьте больше устройств (например, умные шторы, музыкальный плеер).**
   * **Добавьте возможность записи макрокоманд с пульта управления.**
4. **Обработка ошибок:**
   * **Обработайте ситуации, когда команда для слота не назначена (например, если пользователь нажимает кнопку для пустого слота).**

**Цель:**

Изучить и реализовать паттерн **Шаблонный метод (Template Method)** на языке C#. Ваша задача — разработать систему отчетов для различных типов документов (например, PDF, Excel, HTML), где процесс генерации отчетов выполняется по общему шаблону, но с возможностью настройки уникальных шагов для каждого типа отчета.

**Описание задачи:**

Вам нужно создать систему, которая позволяет генерировать различные отчеты (например, PDF-отчет, Excel-отчет, HTML-отчет) на основе общего алгоритма. Основная структура отчетов одинакова, но некоторые шаги могут быть уникальными для каждого типа отчета (например, форматирование, структура данных, сохранение в файл). Программа должна поддерживать:

* Общий шаблон для создания отчета.
* Возможность переопределять отдельные шаги в конкретных реализациях (например, форматирование данных, генерация заголовков).
* Использование перехватываемых методов (hooks), позволяющих добавлять опциональные действия в процессе создания отчета.

**Структура программы:**

1. **Абстрактный класс ReportGenerator** — содержит шаблонный метод для генерации отчетов и определяет последовательность шагов.
2. **Конкретные классы PdfReport, ExcelReport, HtmlReport** — реализуют уникальные шаги для каждого типа отчета.
3. **Клиентский код** — демонстрирует процесс создания отчетов различных типов.

**Шаги выполнения:**

1. Создайте абстрактный класс ReportGenerator, который будет содержать общий алгоритм генерации отчетов
2. Реализуйте класс **PdfReport**, который будет представлять логику для PDF-отчета
3. Реализуйте класс **ExcelReport**, который будет представлять логику для Excel-отчета
4. Реализуйте класс **HtmlReport**, который будет представлять логику для HTML-отчета
5. Напишите клиентский код для демонстрации работы программы

**Задания:**

1. **Реализуйте код по шагам выше.**
2. **Тестирование**:
   * Проверьте процесс генерации PDF, Excel и HTML отчетов.
   * Убедитесь, что возможность сохранения отчета работает корректно для Excel.
3. **Расширение функционала**:
   * Добавьте новый тип отчета, например, **CSV-отчет**, с уникальными шагами форматирования и создания.
   * Внедрите возможность отправки отчета по электронной почте вместо его сохранения.
4. **Обработка ошибок**:
   * Добавьте проверку пользовательского ввода в методе CustomerWantsSave, чтобы обрабатывать некорректные ответы.
   * Реализуйте журналирование шагов процесса генерации отчета.

**Вопросы для самопроверки:**

1. В чем преимущества использования паттерна "Шаблонный метод" при создании отчетов?
2. Как можно расширять систему для добавления новых типов отчетов, не изменяя существующий код?
3. Какие методы следует сделать абстрактными, а какие можно оставить как перехватываемые (hooks)?

**Цель:**

Изучить и реализовать паттерн **Посредник (Mediator)** на языке C#. Ваша задача — создать чат-систему, в которой пользователи могут общаться друг с другом через посредника. Посредник будет координировать взаимодействия между пользователями, включая отправку сообщений, добавление новых участников и управление каналами.

**Описание задачи:**

Необходимо создать систему чатов, в которой пользователи общаются через посредника. Посредник будет управлять отправкой сообщений, добавлением участников в чат и уведомлением всех пользователей о входе или выходе других участников. Ваша система должна поддерживать:

* Возможность отправки сообщений пользователями через посредника.
* Добавление и удаление пользователей из чата.
* Уведомление всех участников о подключении и отключении других пользователей.
* Поддержка различных каналов чатов, чтобы пользователи могли общаться в отдельных группах.

**Структура программы:**

1. **Интерфейс IMediator** — определяет контракт для взаимодействия между пользователями.
2. **Класс ChatMediator** — конкретный посредник, который координирует взаимодействие между пользователями.
3. **Интерфейс IUser** — определяет методы для взаимодействия пользователей с посредником.
4. **Классы User** — представляет конкретных участников чата.
5. **Клиентский код** — демонстрирует работу чат-системы с несколькими пользователями и каналами.

**Шаги выполнения:**

1. Создайте интерфейс IMediator, который будет определять основные методы для посредника
2. Создайте класс **ChatMediator**, который будет реализовывать интерфейс *IMediator* и управлять взаимодействием между пользователями
3. Создайте интерфейс IUser, который будет представлять участников чата
4. Реализуйте класс **User**, представляющий конкретных пользователей чата
5. Создайте расширение для поддержания нескольких каналов чатов. Реализуйте класс **ChannelMediator**, чтобы один посредник мог управлять отдельными каналами
6. Напишите клиентский код для демонстрации работы посредника с несколькими пользователями и каналами

**Задания:**

1. **Реализуйте код по шагам выше.**
2. **Тестирование**:
   * Проверьте отправку сообщений между пользователями в разных каналах.
   * Убедитесь, что уведомления о подключении и отключении пользователей работают корректно.
3. **Расширение функционала**:
   * Реализуйте функционал для отправки приватных сообщений между пользователями, не затрагивая остальных участников чата.
   * Добавьте возможность кросс-канальной отправки сообщений, чтобы один пользователь мог отправить сообщение в другой канал.
   * Добавьте команду для администраторов чата, чтобы они могли временно блокировать пользователей.
4. **Обработка ошибок**:
   * Реализуйте проверку на существование канала при отправке сообщения, если канала не существует — создайте его или выдайте ошибку.
   * Обработайте ситуацию, когда пользователь пытается отправить сообщение в канал, в котором он не состоит.