**Курс:** Шаблоны проектирования приложений

**Тема:** Модуль 06 Порождающие паттерны. Одиночка. Строитель. Прототип/Клон

*Цель задания:*

**Цель работы (Singleton):**

Научиться применять порождающий паттерн "Одиночка" (Singleton) для создания единственного экземпляра класса и управления глобальным доступом к нему.

**Описание задачи:**

Необходимо разработать систему управления конфигурацией приложения, в которой настройки могут быть загружены только один раз при запуске программы. Для реализации этого сценария нужно использовать паттерн "Одиночка", чтобы гарантировать существование единственного экземпляра объекта конфигурации и предоставление глобального доступа к нему.

**Шаги выполнения:**

1. **Создайте класс ConfigurationManager, который будет хранить настройки приложения.**
   * Реализуйте приватный конструктор, чтобы предотвратить создание объектов вне класса.
   * Добавьте статическое поле для хранения единственного экземпляра класса.
   * Добавьте статический метод GetInstance(), который возвращает единственный экземпляр класса.
2. **Добавьте методы для работы с настройками:**
   * Реализуйте методы для загрузки, чтения и изменения настроек.
   * Настройки могут храниться в виде словаря (Dictionary<string, string>).
3. **Гарантируйте потокобезопасность создания объекта ConfigurationManager (при необходимости).**
   * Реализуйте ленивую инициализацию или используйте блокировку для обеспечения потокобезопасности.
4. **Протестируйте класс ConfigurationManager в основном классе:**
   * Продемонстрируйте, что независимо от количества вызовов метода GetInstance(), всегда возвращается один и тот же экземпляр.

**Задание:**

1. Реализуйте указанный класс ConfigurationManager в проекте.
2. Модифицируйте класс для поддержки потокобезопасности (используя блокировку или ленивую инициализацию).
3. Добавьте возможность сохранения и загрузки настроек из файла (например, текстового файла).
4. Протестируйте работу класса в многопоточном приложении, чтобы убедиться в корректной реализации одиночки.

**Дополнительные задачи:**

1. **Расширьте функционал класса**: Добавьте возможность загрузки настроек из внешних источников (например, базы данных).
2. **Добавьте обработку исключений**: В случае отсутствия настройки или попытки некорректного доступа к конфигурациям.

**Цель работы (Builder):**

Научиться применять порождающий паттерн "Строитель" для создания сложных объектов с конфигурацией и параметрами, упрощая процесс их создания и улучшая читаемость кода.

**Описание задачи:**

Необходимо разработать систему для построения различных типов отчетов (например, текстовых отчетов и HTML-отчетов) с помощью паттерна "Строитель". Каждый отчет должен иметь возможность настройки заголовка, содержимого и подвала. Паттерн "Строитель" позволит гибко создавать отчеты с различными комбинациями этих элементов.

**Шаги выполнения:**

1. **Создайте интерфейс IReportBuilder, который будет определять методы для создания различных частей отчета:**
   * SetHeader(string header)
   * SetContent(string content)
   * SetFooter(string footer)
   * GetReport(): возвращает конечный результат (отчет).
2. **Реализуйте конкретные строители, например, TextReportBuilder и HtmlReportBuilder, которые реализуют интерфейс IReportBuilder.**
   * **TextReportBuilder**: создает текстовые отчеты.
   * **HtmlReportBuilder**: создает HTML-отчеты.
3. **Создайте класс ReportDirector, который будет использовать строителя для создания отчетов:**
   * Метод ConstructReport(IReportBuilder builder) для конфигурации и создания отчета.
4. **Создайте класс Report, который будет представлять результат работы строителя.**
   * Report должен хранить заголовок, содержимое и подвал.
5. **Напишите основной класс для тестирования:**
   * Создайте несколько отчетов с разными конфигурациями, используя разные строители.
   * Выведите результат на экран.

**Задание:**

1. Реализуйте интерфейс IReportBuilder и конкретные строители для различных типов отчетов (TextReportBuilder и HtmlReportBuilder).
2. Создайте класс Report для хранения и отображения результатов работы строителя.
3. Создайте класс ReportDirector для управления процессом создания отчетов.
4. Напишите основной класс для тестирования, который создает и отображает отчеты с использованием различных строителей.

**Дополнительные задачи:**

1. Добавьте поддержку дополнительных форматов отчетов, например, XML или PDF.
2. Расширьте функциональность строителя для поддержки дополнительных опций, таких как стили или форматирование.
3. Добавьте возможность динамического изменения содержимого отчета после его создания.

**Цель работы (Prototype):**

Изучить и применить порождающий паттерн "Прототип" для создания объектов на основе их прототипов. Научиться использовать паттерн для клонирования объектов и создания новых экземпляров с измененными параметрами.

**Сценарий: Интернет-магазин с различными типами заказов**

Предположим, что у нас есть система управления заказами, где каждый заказ может иметь множество свойств (товары, доставка, скидки, методы оплаты). Вместо того чтобы каждый раз создавать заказ с нуля, мы можем использовать шаблон заказа и клонировать его, внося только минимальные изменения.

**Задача:**

Создать систему для управления заказами с возможностью клонирования заказа через паттерн **Прототип**. Каждый заказ должен включать товары, стоимость доставки, скидки и метод оплаты.

**Требования:**

1. **Класс Order (Заказ)** должен поддерживать клонирование заказа:
   * Список товаров (с ценами и количеством).
   * Стоимость доставки.
   * Применяемые скидки.
   * Метод оплаты.
2. **Класс Product (Товар)** должен содержать информацию о товаре и также поддерживать клонирование.
3. **Методы Clone()** должны поддерживать глубокое клонирование для всех вложенных объектов.

**Реализация:**

1. **Класс Order**:
   * Включает методы для глубокого клонирования всех вложенных объектов, таких как товары и скидки.
2. **Класс Product** и **класс Discount**:
   * Реализуют интерфейс ICloneable для клонирования.