**Курс:** Шаблоны проектирования приложений

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

*Баллы:*

**Цель работы (Singleton):**

Научиться применять порождающий паттерн "Одиночка" для создания единственного экземпляра класса и управления глобальным доступом к этому объекту. Реализовать потокобезопасную версию одиночки и протестировать ее работу в многопоточном окружении.

**Описание задачи:**

Реализовать систему логирования на основе паттерна "Одиночка". Логгер должен предоставлять доступ к единственному экземпляру класса и обеспечивать потокобезопасную запись логов в файл. Лог-сообщения должны быть разделены по уровням логирования (информация, предупреждение, ошибка).

**Шаги выполнения:**

1. **Создайте класс Logger, который реализует паттерн "Одиночка"**:
   * Приватный конструктор для предотвращения создания объектов извне.
   * Статическое поле для хранения единственного экземпляра класса.
   * Статический метод GetInstance(), который возвращает единственный экземпляр класса.
2. **Реализуйте методы для логирования сообщений**:
   * Метод Log(string message, LogLevel level) для записи сообщений в файл.
   * Реализуйте несколько уровней логирования: INFO, WARNING, ERROR.
3. **Реализуйте потокобезопасность**:
   * Сделайте метод GetInstance() потокобезопасным (используйте блокировку lock или ленивую инициализацию).
4. **Добавьте возможность изменения уровня логирования**:
   * Добавьте метод SetLogLevel(LogLevel level), чтобы логгер записывал сообщения только соответствующего или более высокого уровня.
5. **Протестируйте систему логирования в многопоточном приложении**:
   * Создайте несколько потоков, каждый из которых будет писать логи с разными уровнями логирования.

**Задание:**

1. Реализуйте класс Logger с поддержкой паттерна "Одиночка".
2. Добавьте функционал для логирования сообщений с учетом уровня логирования.
3. Обеспечьте потокобезопасность создания экземпляра класса Logger.
4. Протестируйте работу логгера в многопоточном приложении с различными уровнями логирования.

**Дополнительные задачи:**

1. Добавьте возможность изменения пути к файлу логов через метод SetLogFilePath(string path).
2. Реализуйте чтение логов из файла и отображение их на экране.

**Цель работы (Builder):**

Изучить и применить порождающий паттерн "Строитель" для создания сложных объектов с конфигурацией и параметрами. Научиться использовать этот паттерн для построения отчетов с разными форматами и опциями.

**Задача:**

Создать приложение для сборки компьютеров с использованием порождающего паттерна Строитель. Приложение должно позволять создавать компьютеры с различными конфигурациями, такими как процессор, оперативная память, жесткий диск, видеокарта, и операционная система. Система должна позволять строить как базовые офисные компьютеры, так и мощные игровые машины.

**Требования:**

1. **Класс продукта Computer**:
   * Должен включать свойства для конфигурации компьютера:
     + Процессор (CPU)
     + Оперативная память (RAM)
     + Жесткий диск (HDD/SSD)
     + Видеокарта (GPU)
     + Операционная система (OS)
   * Должен иметь метод ToString(), который выводит полную конфигурацию компьютера.
2. **Интерфейс строителя IComputerBuilder**:
   * Методы для добавления каждого компонента компьютера (CPU, RAM, HDD/SSD, GPU, OS).
   * Метод для возврата собранного объекта компьютера.
3. **Конкретные строители**:
   * Реализовать двух строителей:
     + **Офисный компьютер** (OfficeComputerBuilder) с минимальными параметрами (например, простой процессор, 8GB RAM, HDD, встроенная видеокарта).
     + **Игровой компьютер** (GamingComputerBuilder) с высокими параметрами (например, мощный процессор, 32GB RAM, SSD, дискретная видеокарта).
4. **Директор ComputerDirector**:
   * Должен управлять процессом сборки компьютера. Он должен принимать строителя и вызывать методы в определённом порядке для сборки компьютера.
5. **Клиент**:
   * Написать клиентский код, который создает офисный и игровой компьютер, используя паттерн Строитель. В результате программа должна вывести на экран конфигурации обоих компьютеров.

**Реализация:**

1. **Класс продукта:**

*public class Computer*

*{*

*public string CPU { get; set; }*

*public string RAM { get; set; }*

*public string Storage { get; set; }*

*public string GPU { get; set; }*

*public string OS { get; set; }*

*public override string ToString()*

*{*

*return $"Компьютер: CPU - {CPU}, RAM - {RAM}, Накопитель - {Storage}, GPU - {GPU}, ОС - {OS}";*

*}*

*}*

1. **Интерфейс строителя:**

*public interface IComputerBuilder*

*{*

*void SetCPU();*

*void SetRAM();*

*void SetStorage();*

*void SetGPU();*

*void SetOS();*

*Computer GetComputer();*

*}*

1. Конкретные строители:

*public class OfficeComputerBuilder : IComputerBuilder*

*{*

*private Computer \_computer = new Computer();*

*public void SetCPU() => \_computer.CPU = "Intel i3";*

*public void SetRAM() => \_computer.RAM = "8GB";*

*public void SetStorage() => \_computer.Storage = "1TB HDD";*

*public void SetGPU() => \_computer.GPU = "Integrated";*

*public void SetOS() => \_computer.OS = "Windows 10";*

*public Computer GetComputer() => \_computer;*

*}*

*public class GamingComputerBuilder : IComputerBuilder*

*{*

*private Computer \_computer = new Computer();*

*public void SetCPU() => \_computer.CPU = "Intel i9";*

*public void SetRAM() => \_computer.RAM = "32GB";*

*public void SetStorage() => \_computer.Storage = "1TB SSD";*

*public void SetGPU() => \_computer.GPU = "NVIDIA RTX 3080";*

*public void SetOS() => \_computer.OS = "Windows 11";*

*public Computer GetComputer() => \_computer;*

*}*

1. Директор

*public class ComputerDirector*

*{*

*private IComputerBuilder \_builder;*

*public ComputerDirector(IComputerBuilder builder)*

*{*

*\_builder = builder;*

*}*

*public void ConstructComputer()*

*{*

*\_builder.SetCPU();*

*\_builder.SetRAM();*

*\_builder.SetStorage();*

*\_builder.SetGPU();*

*\_builder.SetOS();*

*}*

*public Computer GetComputer()*

*{*

*return \_builder.GetComputer();*

*}*

*}*

1. Клиентский код:

*class Program*

*{*

*static void Main(string[] args)*

*{*

*// Создаем офисный компьютер*

*IComputerBuilder officeBuilder = new OfficeComputerBuilder();*

*ComputerDirector director = new ComputerDirector(officeBuilder);*

*director.ConstructComputer();*

*Computer officeComputer = director.GetComputer();*

*Console.WriteLine(officeComputer);*

*// Создаем игровой компьютер*

*IComputerBuilder gamingBuilder = new GamingComputerBuilder();*

*director = new ComputerDirector(gamingBuilder);*

*director.ConstructComputer();*

*Computer gamingComputer = director.GetComputer();*

*Console.WriteLine(gamingComputer);*

*}*

*}*

**Дополнительные задания:**

1. Добавьте больше опций для конфигурации компьютеров (например, тип охлаждения, блок питания).
2. Реализуйте дополнительные строители для других типов компьютеров (например, сервер или компьютер для работы с графикой).
3. Добавьте валидацию данных, чтобы гарантировать, что некоторые компоненты совместимы друг с другом.

**Цель работы (Prototype):**

Изучить и применить порождающий паттерн "Прототип" для создания и клонирования объектов. Научиться использовать этот паттерн для работы с объектами, которые могут содержать вложенные элементы.

**Описание задачи:**

В этой лабораторной работе вам предстоит создать систему для управления различными типами документов, используя паттерн "Прототип". Система должна позволять клонировать документы с вложенными элементами, такими как разделы и изображения.

**Задание:**

1. **Создайте интерфейс IPrototype, который будет определять метод клонирования:**
   * IPrototype Clone()
2. **Реализуйте классы, представляющие разные типы документов и вложенные элементы, реализующие интерфейс IPrototype:**
   * **Document**: класс, представляющий основной документ. Он должен содержать заголовок, содержимое и коллекции вложенных элементов (разделов и изображений).
   * **Section**: класс, представляющий раздел документа с заголовком и текстом.
   * **Image**: класс, представляющий изображение документа с URL.
3. **В каждом классе реализуйте методы для установки и получения данных, а также метод Clone() для клонирования объектов:**
   * Для **Document**: управление заголовком, содержимым и коллекциями секций и изображений.
   * Для **Section** и **Image**: управление текстом и URL.
4. **Создайте класс DocumentManager, который будет использовать прототипы для создания и клонирования документов.**
   * Метод CreateDocument(IPrototype prototype) для клонирования документа на основе прототипа.
5. **Напишите основной класс для тестирования:**
   * Создайте и клонируйте документы, изменяя их содержимое и добавляя новые элементы.
   * Продемонстрируйте использование клонирования для создания и модификации документов.

**Задание:**

1. Реализуйте интерфейс IPrototype с методом Clone().
2. Создайте классы Section, Image и Document, реализующие интерфейс IPrototype.
   * Реализуйте методы для установки и получения данных.
   * Реализуйте метод Clone() для клонирования объектов.
3. Создайте класс DocumentManager для управления созданием и клонированием документов.
4. Напишите основной класс для тестирования, который создаст и клонирует документы, изменяя их содержимое и добавляя новые элементы.

**Дополнительные задачи:**

1. Добавьте поддержку дополнительных типов вложенных элементов, например, таблиц или графиков.
2. Расширьте функциональность, добавив возможность клонирования документов с изменением их структуры.
3. Реализуйте метод для сохранения и загрузки документов из файлов, используя клонирование.