**Курс:** Шаблоны проектирования приложений

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

*Баллы:*

**Цель работы (Singleton):**

Овладеть реализацией сложной версии паттерна "Одиночка" для управления единственным экземпляром класса в условиях многопоточности и взаимодействия с внешними источниками данных (например, файл, база данных).

**Описание задачи:**

Необходимо разработать систему для управления логированием в многопоточном приложении с использованием паттерна "Одиночка". Логгер должен сохранять сообщения в файл и обеспечить потокобезопасность при работе с общим ресурсом (файлом). В системе также должна быть возможность динамически изменять уровень логирования (информация, предупреждения, ошибки).

**Шаги выполнения:**

1. Создайте класс Logger, который реализует паттерн "Одиночка":
   * Приватный конструктор для предотвращения создания объекта извне.
   * Статический метод GetInstance(), возвращающий единственный экземпляр логгера.
   * Поддержка многопоточности: реализация должна быть потокобезопасной (используйте блокировки или ленивую инициализацию).
2. Добавьте функционал для логирования сообщений:
   * Метод Log(string message, LogLevel level), который сохраняет сообщения в файл.
   * Реализуйте разные уровни логирования:
     + INFO: информационные сообщения.
     + WARNING: предупреждения.
     + ERROR: ошибки.
   * В зависимости от текущего уровня логирования (который можно изменять), логгер должен записывать только сообщения, соответствующие или превышающие заданный уровень.
3. Добавьте возможность изменения уровня логирования через метод SetLogLevel(LogLevel level):
   * Уровень логирования можно менять в любой момент, и это должно влиять на фильтрацию сообщений.
4. Добавьте поддержку конфигурации логгера из внешнего источника:
   * Реализуйте возможность загрузки настроек из файла конфигурации (например, JSON или XML), где хранится информация о текущем уровне логирования и пути к файлу логов.
5. Создайте класс LogReader для чтения логов:
   * Реализуйте возможность чтения сообщений из лог-файла и отображения их на экране с фильтрацией по уровню логирования (например, только ошибки).
6. Протестируйте работу логгера в многопоточном окружении:
   * Запустите несколько потоков, каждый из которых будет логировать сообщения с разными уровнями.

**Задание:**

1. Реализуйте класс Logger с паттерном "Одиночка" и поддержкой потокобезопасности.
2. Добавьте функциональность для изменения уровня логирования и записи в файл.
3. Реализуйте чтение конфигурации логгера из файла.
4. Создайте многопоточное приложение, в котором разные потоки будут логировать сообщения с разными уровнями.
5. Реализуйте класс LogReader для чтения и фильтрации логов.
6. Убедитесь, что при любом количестве потоков всегда используется один экземпляр логгера.

**Дополнительные задачи:**

1. Добавьте поддержку ротации лог-файлов: когда файл логов достигает определенного размера, создается новый файл.
2. Расширьте функциональность: добавьте возможность логирования в несколько источников (например, файл и консоль).
3. Модифицируйте LogReader для фильтрации логов по времени или другим критериям.

**Цель работы (Builder):**

Освоить применение паттерна "Строитель" для создания сложных объектов с расширенными возможностями конфигурации и построения. Научиться применять паттерн для построения отчетов с различными форматами и опциями, включая возможность добавления динамического контента.

**Описание задачи:**

В этой работе вам предстоит разработать систему для создания отчетов различного формата (текстовый, HTML, PDF) с помощью паттерна "Строитель". Отчеты должны поддерживать следующие функциональности:

* Динамическое добавление разделов и элементов (например, заголовки, таблицы, графики).
* Возможность настройки стилей и форматов (например, цвет фона, шрифты, размеры шрифтов).
* Экспорт отчетов в несколько форматов (текстовый, HTML, PDF).

**Шаги выполнения:**

1. **Создайте интерфейс IReportBuilder, который определяет методы для создания различных частей отчета:**
   * SetHeader(string header)
   * SetContent(string content)
   * SetFooter(string footer)
   * AddSection(string sectionName, string sectionContent)
   * SetStyle(ReportStyle style)
   * GetReport(): возвращает конечный результат (отчет).
2. **Реализуйте конкретные строители для различных типов отчетов:**
   * **TextReportBuilder**: создает текстовые отчеты.
   * **HtmlReportBuilder**: создает HTML-отчеты.
   * **PdfReportBuilder**: создает PDF-отчеты (используйте стороннюю библиотеку, например, iTextSharp для PDF).
3. **Создайте класс ReportStyle, который будет содержать информацию о стилях отчета:**
   * Цвет фона
   * Цвет шрифта
   * Размер шрифта
4. **Создайте класс Report, который будет представлять результат работы строителя:**
   * Храните заголовок, содержимое, подвал и разделы.
   * Реализуйте метод Export() для экспорта отчета в нужный формат.
5. **Создайте класс ReportDirector, который будет использовать строителя для создания отчетов:**
   * Метод ConstructReport(IReportBuilder builder, ReportStyle style) для конфигурации и создания отчета.
6. **Напишите основной класс для тестирования:**
   * Создайте несколько отчетов с разными конфигурациями и форматами, используя разные строители.
   * Продемонстрируйте возможность добавления динамического контента и применения стилей.

**Задание:**

1. Реализуйте интерфейс IReportBuilder и конкретные строители для разных форматов отчетов (TextReportBuilder, HtmlReportBuilder, PdfReportBuilder).
2. Создайте класс Report для хранения и отображения отчетов, включая возможность добавления секций и применения стилей.
3. Создайте класс ReportStyle для управления стилями отчетов.
4. Создайте класс ReportDirector для управления процессом создания отчетов и использования строителей.
5. Напишите основной класс для тестирования, который создает отчеты в разных форматах и демонстрирует использование динамического контента и стилей.

**Дополнительные задачи:**

1. Добавьте поддержку дополнительных форматов отчетов (например, JSON).
2. Расширьте функциональность для работы с большими объемами данных и сложными структурами отчетов.
3. Реализуйте экспорт отчетов в файлы, используя соответствующие библиотеки для работы с форматом (например, для PDF).

**Цель работы (Prototype):**

Освоить применение паттерна "Прототип" для создания и клонирования сложных объектов. Научиться использовать этот паттерн для создания объектов с вложенными элементами и изменением их состояния.

Допустим, мы разрабатываем систему для онлайн-игры, в которой есть множество персонажей с различными характеристиками и уникальными способностями. Каждый персонаж может быть создан на основе определённых шаблонов, и игроки могут настраивать этих персонажей, меняя их атрибуты, оружие, броню, способности и т.д.

Вместо того чтобы каждый раз создавать новых персонажей с нуля, мы можем клонировать существующие экземпляры и изменять только нужные свойства. Это сэкономит время на настройку и позволит легко создавать новых персонажей.

**Задача:**

Создать систему для клонирования персонажей в онлайн-игре, используя паттерн **Прототип**. Персонажи должны поддерживать сложную структуру атрибутов, снаряжения и способностей, а также возможность глубокого копирования этих элементов.

**Требования:**

1. **Класс Character (Персонаж)** должен включать:
   * Основные характеристики: здоровье, сила, ловкость, интеллект.
   * Снаряжение: оружие и броню.
   * Способности: список магических и физических умений.
   * Способность клонировать себя, включая глубокое копирование всех вложенных объектов (например, снаряжение и способности).
2. **Класс Weapon (Оружие)** и **класс Armor (Броня)**:
   * Должны иметь свои уникальные параметры (урон для оружия, защита для брони).
   * Должны поддерживать глубокое клонирование.
3. **Класс Skill (Способность)**:
   * Должен поддерживать клонирование.

**Реализация:**

1. **Персонаж (Character)**:
   * Включает методы для глубокого клонирования всех вложенных объектов.
2. **Оружие (Weapon) и броня (Armor)**:
   * Реализуют интерфейс ICloneable.
3. **Способности (Skill)**:
   * Реализуют интерфейс ICloneable.