**Лабораторная работа: Реализация паттерна "Мост" в системе обработки документов**

**1. Описание проблемы вашей области**

В рамках данной лабораторной работы рассматривается задача разработки системы обработки документов, которая должна поддерживать различные типы документов (например, текстовые и PDF) и способы их вывода (на экран или в файл). Основная проблема заключается в необходимости обеспечить гибкость системы: возможность добавлять новые типы документов и способы вывода без значительных изменений в существующем коде.

Без использования подходящего подхода к проектированию такая система может столкнуться с дублированием кода, сложностью в расширении и жёсткой привязкой типов документов к способам их обработки. Например, если каждый тип документа должен поддерживать вывод и на экран, и в файл, то логика вывода будет повторяться в каждом классе, что усложнит добавление новых функций, таких как вывод на принтер.

Цель работы — реализовать систему с применением паттерна "Мост" для разделения абстракции (типа документа) и реализации (способа вывода), а также сравнить её с альтернативной реализацией без паттерна.

**2. Решение: как используется паттерн в вашем проекте**

**Реализация с паттерном "Мост"**

Паттерн "Мост" применяется для разделения логики типов документов и способов их вывода. В проекте используется графический интерфейс на основе Windows Forms, где пользователь может выбрать тип документа и способ вывода через выпадающие списки, а затем запустить обработку нажатием кнопки.

* **Абстракция**: Определена через абстрактный класс Document, который содержит свойство Content (содержимое документа) и абстрактный метод Process(). Конкретные типы документов реализованы в классах TextDocument и PdfDocument, наследующих Document. Каждый из них задаёт своё содержимое и вызывает метод вывода у связанного реализатора.
* **Реализация**: Определена через интерфейс IDocumentRenderer с методом Render(string content). Конкретные реализаторы — ScreenRenderer (вывод в текстовое поле формы) и FileRenderer (сохранение в файл output.txt).
* **Связь**: Класс Document содержит защищённое поле renderer типа IDocumentRenderer, которое передаётся через конструктор. Это позволяет документам делегировать задачу вывода реализаторам.
* **Клиент**: Код в обработчике события кнопки btnProcess\_Click создаёт нужный документ и реализатор на основе выбора пользователя, затем вызывает Process().

**Программный код, реализующий паттерн**:

using System;

using System.IO;

public interface IDocumentRenderer

{

void Render(string content);

}

// Конкретные реализаторы

public class ScreenRenderer : IDocumentRenderer

{

public void Render(string content)

{

Console.WriteLine($"Вывод на экран:\n{content}");

}

}

public class FileRenderer : IDocumentRenderer

{

private readonly string filePath;

public FileRenderer(string path)

{

filePath = path;

}

public void Render(string content)

{

try

{

File.WriteAllText(filePath, content);

Console.WriteLine($"Содержимое успешно сохранено в файл: {filePath}\n---");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при сохранении в файл {filePath}: {ex.Message}");

}

}

}

public abstract class Document

{

protected IDocumentRenderer renderer;

public string Content { get; set; }

protected Document(IDocumentRenderer renderer)

{

this.renderer = renderer;

Content = "";

}

public abstract void Process();

}

// Уточнённая абстракция

public class TextDocument : Document

{

public TextDocument(IDocumentRenderer renderer) : base(renderer) { }

public override void Process()

{

Content = "Это текстовый документ.";

renderer.Render(Content);

}

}

public class PdfDocument : Document

{

public PdfDocument(IDocumentRenderer renderer) : base(renderer) { }

public override void Process()

{

Content = "Это PDF-документ с форматированием.";

renderer.Render(Content);

}

}

class Program

{

static void Main(string[] args)

{

IDocumentRenderer screenRenderer = new ScreenRenderer();

IDocumentRenderer fileRenderer = new FileRenderer("output.txt");

Document textDocOnScreen = new TextDocument(screenRenderer);

Document textDocToFile = new TextDocument(fileRenderer);

Document pdfDocOnScreen = new PdfDocument(screenRenderer);

Document pdfDocToFile = new PdfDocument(fileRenderer);

Console.WriteLine("Обработка документов:");

textDocOnScreen.Process();

textDocToFile.Process();

pdfDocOnScreen.Process();

pdfDocToFile.Process();

}

}

**Реализация без паттерна "Мост"**

В альтернативной версии без паттерна "Мост" разделение абстракции и реализации отсутствует. Вместо этого каждый тип документа (TextDocument, PdfDocument) сам содержит всю логику вывода:

* Абстрактный класс Document определяет два метода: ProcessToScreen() и ProcessToFile(string filePath).
* Подклассы реализуют оба метода, дублируя код для вывода на экран и в файл.
* Клиентский код в Main напрямую вызывает нужный метод в зависимости от требуемого способа вывода.

**Программный код без реализации паттерна:**

using System;

using System.IO;

public abstract class Document

{

public string Content { get; set; }

protected Document()

{

Content = "";

}

public abstract void ProcessToScreen();

public abstract void ProcessToFile(string filePath);

}

public class TextDocument : Document

{

public TextDocument() : base() { }

public override void ProcessToScreen()

{

Content = "Это текстовый документ.";

Console.WriteLine($"Вывод на экран:\n{Content}");

}

public override void ProcessToFile(string filePath)

{

Content = "Это текстовый документ.";

try

{

File.WriteAllText(filePath, Content);

Console.WriteLine($"Содержимое успешно сохранено в файл: {filePath}\n---");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при сохранении в файл {filePath}: {ex.Message}");

}

}

}

public class PdfDocument : Document

{

public PdfDocument() : base() { }

public override void ProcessToScreen()

{

Content = "Это PDF-документ с форматированием.";

Console.WriteLine($"Вывод на экран:\n{Content}");

}

public override void ProcessToFile(string filePath)

{

Content = "Это PDF-документ с форматированием.";

try

{

File.WriteAllText(filePath, Content);

Console.WriteLine($"Содержимое успешно сохранено в файл: {filePath}\n---");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при сохранении в файл {filePath}: {ex.Message}");

}

}

}

class Program

{

static void Main(string[] args)

{

Document textDocOnScreen = new TextDocument();

Document textDocToFile = new TextDocument();

Document pdfDocOnScreen = new PdfDocument();

Document pdfDocToFile = new PdfDocument();

textDocOnScreen.ProcessToScreen();

textDocToFile.ProcessToFile("output.txt");

pdfDocOnScreen.ProcessToScreen();

pdfDocToFile.ProcessToFile("output.txt");

}

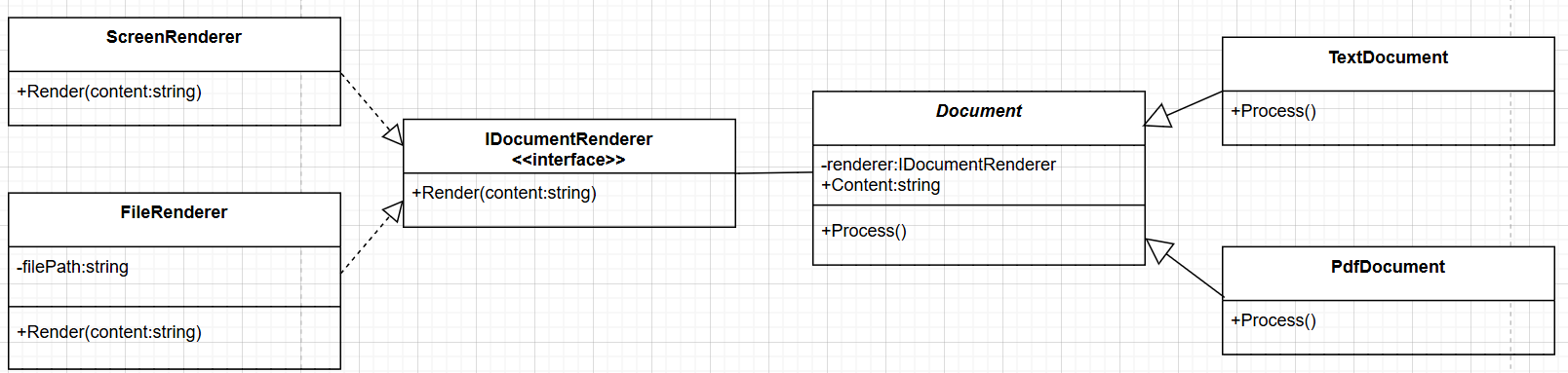
}

**3. Диаграмма классов для архитектуры приложения**

**Диаграмма классов с паттерном "Мост"**

Диаграмма классов для версии с паттерном "Мост" иллюстрирует разделение абстракции и реализации. На ней представлены:

* Интерфейс IDocumentRenderer и его реализации (ScreenRenderer, FileRenderer).
* Абстрактный класс Document и его подклассы (TextDocument, PdfDocument).
* Класс формы Form1, который выступает клиентом и использует объекты паттерна.
* Связи: реализация (от ScreenRenderer и FileRenderer к IDocumentRenderer), наследование (от TextDocument и PdfDocument к Document), ассоциация (от Document к IDocumentRenderer).

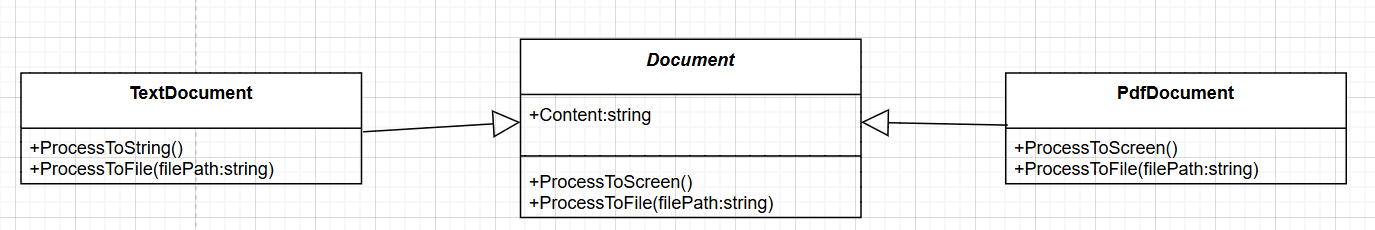
****  
Рисунок 1 – Паттерн "Мост" в архитектуре приложения

Абстракция (Document) связана с реализацией (IDocumentRenderer) через поле renderer, что позволяет гибко комбинировать типы документов и способы вывода.

**Диаграмма классов без паттерна "Мост" (опционально)**

Для сравнения представлена диаграмма без паттерна "Мост". Она проще, так как отсутствуют интерфейс и классы реализаторов:

* Абстрактный класс Document с двумя методами (ProcessToScreen, ProcessToFile).
* Подклассы TextDocument и PdfDocument, реализующие оба метода.
* Класс Program как клиент.

  
Рисунок 2 – Архитектура приложения без паттерна "Мост"

Логика вывода встроена в классы документов, что приводит к дублированию кода.

**4. Вывод: как внедрение паттерна повлияло на работу программы**

Внедрение паттерна "Мост" существенно улучшило структуру и возможности программы:

1. **Гибкость и расширяемость**:
   * В версии с паттерном добавление нового типа документа (например, WordDocument) требует только создания нового подкласса Document, а новый способ вывода (например, PrinterRenderer) — реализации IDocumentRenderer. Без паттерна каждое расширение потребовало бы изменения всех классов документов, добавления новых методов и дублирования кода.
2. **Устранение дублирования**:
   * Логика вывода (например, запись в файл с обработкой ошибок) сосредоточена в FileRenderer, а не повторяется в каждом классе документа, как в версии без паттерна.
3. **Простота использования в GUI**:
   * Паттерн позволил легко интегрировать выбор типа документа и способа вывода через интерфейс Windows Forms. Клиентский код просто комбинирует объекты, не заботясь о деталях реализации вывода.
4. **Читаемость и поддерживаемость**:
   * Код с "Мостом" более структурирован: разделение на абстракцию и реализацию делает его понятнее и проще для модификации.

Таким образом, паттерн "Мост" сделал систему обработки документов более модульной, удобной для расширения и подходящей для использования в графическом интерфейсе, что особенно важно для практического применения.