**Отчёт по лабораторной работе: Применение паттерна "Мост"**

**1. Описание проблемы предметной области**

В рамках данной лабораторной работы требовалось разработать консольное приложение для обработки различных типов документов (текстовых файлов формата .txt и PDF-документов) с возможностью вывода их содержимого различными способами — на экран или в файл. Основная проблема заключалась в необходимости обеспечить гибкость и расширяемость системы. Приложение должно поддерживать добавление новых типов документов (например, DOCX) и новых способов вывода (например, сохранение в JSON) без усложнения структуры кода и без дублирования логики обработки и вывода.

Без применения паттернов проектирования реализация такой системы могла бы привести к следующим сложностям:

* Повторение кода для каждого типа документа и способа вывода.
* Увеличение сложности при добавлении новых типов документов или способов вывода, так как пришлось бы изменять существующие классы.
* Нарушение принципа единственной ответственности, поскольку классы документов одновременно отвечали бы и за извлечение содержимого, и за его отображение.

Таким образом, целью было создать архитектуру, которая минимизирует эти проблемы и обеспечивает модульность и масштабируемость.

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

Для решения описанной проблемы был применён паттерн проектирования **"Мост" (Bridge)**, который позволяет разделить абстракцию (типы документов) и реализацию (способы вывода), обеспечивая их независимое изменение и комбинирование.

**Структура решения**

* **Абстракция**: Представлена абстрактным классом DocumentProcessor, который определяет общий интерфейс для обработки документов. У этого класса есть подклассы:
  + TextDocumentProcessor — для работы с текстовыми файлами (.txt);
  + PdfDocumentProcessor — для работы с PDF-файлами (используется библиотека iText7 для извлечения текста).
* **Реализация**: Определена интерфейсом IDocumentRenderer, который задаёт контракт для вывода содержимого документа. Конкретные реализации:
  + ScreenRenderer — выводит содержимое на экран (в консоль);
  + FileRenderer — сохраняет содержимое в файл.

**Как это работает**

Класс DocumentProcessor содержит ссылку на объект типа IDocumentRenderer, который передаётся через конструктор. Метод Process() в каждом подклассе DocumentProcessor извлекает содержимое документа (например, текст из .txt или PDF) и передаёт его в метод Render() объекта IDocumentRenderer. Таким образом:

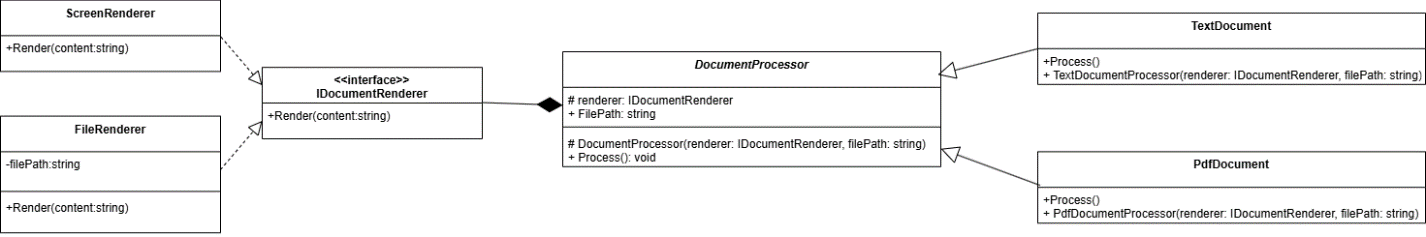
* TextDocumentProcessor и PdfDocumentProcessor отвечают только за извлечение текста;
* ScreenRenderer и FileRenderer — за способ вывода текста.

**Преимущества использования паттерна**

* **Независимость**: Добавление нового типа документа (например, DocxDocumentProcessor) не требует изменений в классах рендеринга.
* **Расширяемость**: Новый способ вывода (например, JsonRenderer) можно добавить, просто реализовав интерфейс IDocumentRenderer, не трогая классы документов.
* **Модульность**: Логика обработки и вывода разделена, что упрощает поддержку и тестирование.

**3. Диаграмма классов для архитектуры приложения с применением паттерна**

Для наглядного представления архитектуры приложения ниже описана диаграмма классов с использованием паттерна "Мост".

Рисунок 1 – Архитектура приложения с паттерном "Мост"

На рисунке 1 изображена структура классов приложения, где паттерн "Мост" используется для разделения абстракции обработки документов и реализации способов вывода. Основные элементы:

* **DocumentProcessor** (абстрактный класс):
  + Свойство: IDocumentRenderer renderer (ассоциация с интерфейсом реализации);
  + Абстрактный метод: void Process().
* **TextDocumentProcessor**:
  + Наследуется от DocumentProcessor;
  + Реализует Process() для чтения текста из .txt.
* **PdfDocumentProcessor**:
  + Наследуется от DocumentProcessor;
  + Реализует Process() для извлечения текста из .pdf.
* **IDocumentRenderer** (интерфейс):
  + Метод: void Render(string content).
* **ScreenRenderer**:
  + Реализует IDocumentRenderer;
  + Выводит текст в консоль.
* **FileRenderer**:
  + Реализует IDocumentRenderer;
  + Сохраняет текст в файл.

**Связи**:

* DocumentProcessor имеет композицию с IDocumentRenderer.
* TextDocumentProcessor и PdfDocumentProcessor наследуются от DocumentProcessor.
* ScreenRenderer и FileRenderer реализуют IDocumentRenderer.

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

Внедрение паттерна "Мост" существенно улучшило архитектуру приложения и его дальнейшую разработку. Основные эффекты:

1. **Улучшенная гибкость и расширяемость**:
   * Добавление нового типа документа (например, DocxDocumentProcessor) требует лишь создания нового подкласса DocumentProcessor, что не затрагивает существующие классы рендеринга.
   * Новый способ вывода (например, JsonRenderer) можно добавить, реализовав интерфейс IDocumentRenderer, без необходимости менять классы обработки документов.
2. **Снижение дублирования кода**:
   * В версии без паттерна каждый класс документа содержал бы методы для всех способов вывода (например, ProcessToScreen() и ProcessToFile()), что привело бы к повторению кода. С паттерном "Мост" логика вывода вынесена в отдельные классы рендеринга, что устраняет дублирование.
3. **Чёткое разделение ответственности**:
   * Классы DocumentProcessor отвечают только за извлечение содержимого, а классы IDocumentRenderer — за его отображение или сохранение. Это делает код более модульным, понятным и соответствующим принципу единственной ответственности.
4. **Упрощение тестирования**:
   * Обработка документов и способы вывода могут тестироваться независимо друг от друга, что облегчает поиск ошибок и поддержку кода.

**Сравнение с версией без паттерна**

Без использования паттерна "Мост" архитектура приложения была бы менее эффективной:

* Каждый класс документа (TextDocument, PdfDocument) содержал бы методы вроде ProcessToScreen() и ProcessToFile(), что привело бы к дублированию логики вывода.
* Добавление нового способа вывода потребовало бы изменения всех классов документов, что нарушило бы принцип открытости/закрытости (Open/Closed Principle).
* Код стал бы сложнее для понимания и поддержки из-за смешивания логики обработки и вывода.

Применение паттерна "Мост" устранило эти недостатки, сделав приложение более гибким, масштабируемым и удобным для дальнейшей разработки.