**Отчёт по лабораторной работе №3: Паттерн "Итератор"**

**Описание проблемы предметной области:**

В современных приложениях для сбора данных часто требуется организовать процесс заполнения сложных форм, состоящих из нескольких логически связанных блоков. Такие формы могут включать в себя различные типы полей (текстовые, выпадающие списки и т.д.) и требуют пошагового заполнения, чтобы упростить взаимодействие пользователя с системой и избежать перегрузки интерфейса.

**Решение:**

Для решения проблемы в проекте используется паттерн "Итератор", который позволяет организовать удобный и последовательный доступ к элементам сложной структуры данных (блокам формы) без нарушения принципа инкапсуляции.

Ключевым элементом решения является класс FormCollection, который отвечает за хранение блоков формы. Этот класс предоставляет метод CreateIterator(), возвращающий объект итератора. Итератор, реализованный в классе FormIterator, инкапсулирует логику последовательного обхода блоков. Он предоставляет методы HasNext() и Next(), которые позволяют определить наличие следующего блока и получить его. Благодаря этому клиентский код, например, класс MainWindow, может сосредоточиться на обработке текущего блока, не заботясь о деталях перехода между блоками.

Использование итератора делает систему гибкой и расшир

яемой. Например, можно легко изменить структуру хранения данных в FormCollection или добавить новые типы блоков, не затрагивая клиентский код. Это достигается благодаря тому, что детали обхода скрыты за интерфейсом итератора.

Кроме того, после заполнения формы данные из всех блоков собираются и передаются в класс HtmlReportGenerator, который генерирует итоговый HTML-отчёт. Удобный доступ к данным через коллекцию и итератор упрощает этот процесс.

Код программы с паттерном:

using System;

using System.Collections.Generic;

namespace IteratorPatternDemo

{

// Iterator: интерфейс для обхода элементов

public interface IIterator

{

bool HasNext();

object Next();

object Current();

}

// Aggregate: интерфейс для создания итератора

public interface IAggregate

{

IIterator CreateIterator();

}

// ConcreteIterator: реализация итератора

public class FormIterator : IIterator

{

private readonly List<FormBlock> \_blocks;

private int \_position = 0;

public FormIterator(List<FormBlock> blocks)

{

\_blocks = blocks;

}

public bool HasNext()

{

return \_position < \_blocks.Count;

}

public object Next()

{

return \_blocks[\_position++];

}

public object Current()

{

return \_blocks[\_position - 1];

}

}

// ConcreteAggregate: реализация Aggregate

public class FormCollection : IAggregate

{

private readonly List<FormBlock> \_blocks = new List<FormBlock>();

public void AddBlock(FormBlock block)

{

\_blocks.Add(block);

}

public IIterator CreateIterator()

{

return new FormIterator(\_blocks);

}

}

// FormBlock: блок формы

public class FormBlock

{

public string Title { get; }

public List<FormField> Fields { get; }

public FormBlock(string title)

{

Title = title;

Fields = new List<FormField>();

}

public void AddField(FormField field)

{

Fields.Add(field);

}

}

// FormField: поле формы

public class FormField

{

public string Label { get; }

public string Value { get; set; }

public FormField(string label)

{

Label = label;

Value = string.Empty;

}

}

// Client: демонстрация работы

class Program

{

static void Main(string[] args)

{

// Создаем коллекцию блоков формы

var formCollection = new FormCollection();

// Блок 1: Персональная информация

var personalInfoBlock = new FormBlock("Персональная информация");

personalInfoBlock.AddField(new FormField("Имя"));

personalInfoBlock.AddField(new FormField("Фамилия"));

personalInfoBlock.AddField(new FormField("Возраст"));

formCollection.AddBlock(personalInfoBlock);

// Блок 2: Профессиональные данные

var professionalInfoBlock = new FormBlock("Профессиональные данные");

professionalInfoBlock.AddField(new FormField("Университет"));

professionalInfoBlock.AddField(new FormField("Специальность"));

formCollection.AddBlock(professionalInfoBlock);

// Итерация по блокам формы

var iterator = formCollection.CreateIterator();

while (iterator.HasNext())

{

var block = (FormBlock)iterator.Next();

Console.WriteLine($"Блок: {block.Title}");

foreach (var field in block.Fields)

{

Console.WriteLine($" Поле: {field.Label}, Значение: {field.Value}");

}

}

}

}

}

**Диаграмма классов**

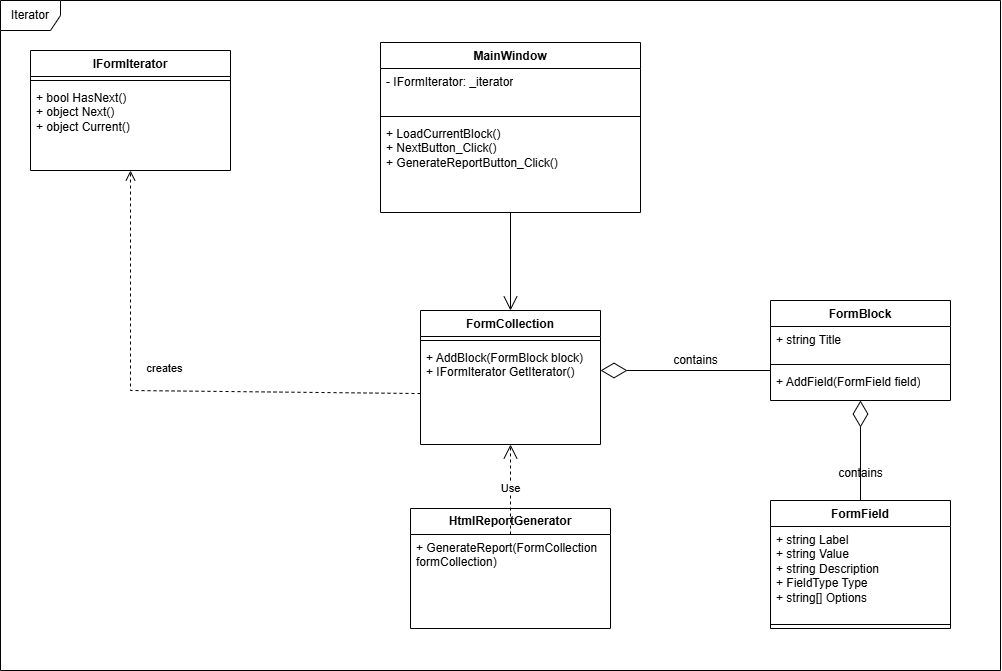


Рисунок 1 – Паттерн “Итератор” в архитектуре приложения.

**Вывод**

 Паттерн "Итератор" был успешно применён для решения задачи пошагового заполнения формы, состоящей из нескольких блоков. Использование этого паттерна позволило организовать удобный доступ к элементам сложной структуры данных, сохраняя принцип инкапсуляции и упрощая взаимодействие с коллекцией блоков формы. Логика обхода коллекции была инкапсулирована в итераторе, что упростило клиентский код и позволило сосредоточиться на обработке данных, а не на деталях их обхода