

Міністерство освіти і науки України
Національний технічний університет України «Київський
політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №4

з дисципліни

«Основи розробки програмного забезпечення на платформі
Microsoft.NET»

«Шаблони проектування»

Виконала ІП-21 Голованьов Г.О.

Київ 2024

Комп'ютерний практикум № 4

Шаблони проектування

Мета: ознайомитися з основними шаблонами проектування, навчитися застосовувати їх при проектуванні і розробці ПЗ.

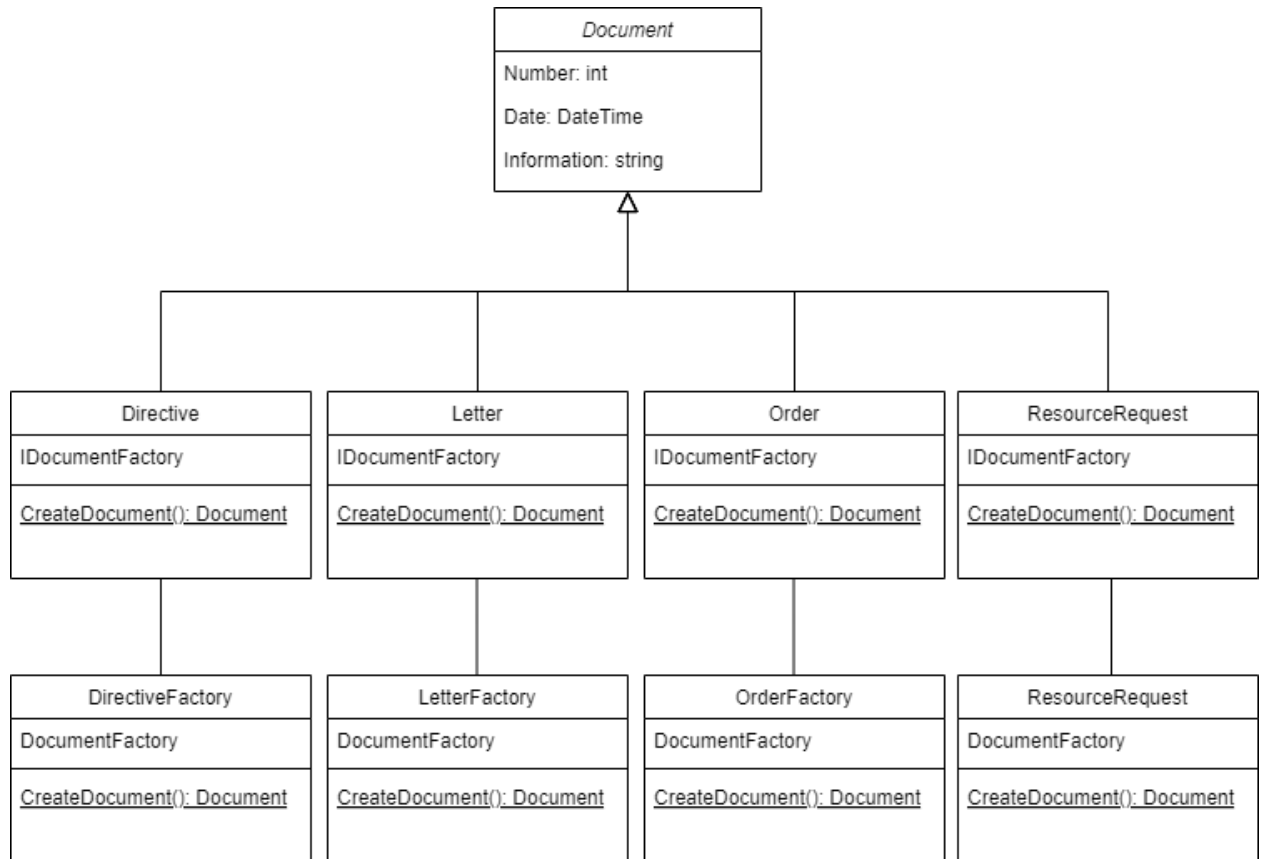
Обґрунтування обраного патерну: Factory Method

Паттерн "Фабричний метод" (Factory Method) був обраний через те, що він дозволяє динамічно визначати тип створюваних об'єктів в залежності від умов або параметрів. У нашому випадку ми маємо різні типи документів (лист, наказ, розпорядження, заявка на ресурси), які можуть бути створені в залежності від потреб користувача або вхідних даних. Використання фабричного методу дозволяє нам зберігати код гнучким і легким для розширення, оскільки нові типи документів можуть бути додані без змін у вже існуючому коді.

Опис архітектури проекту:

1. **Document (Базовий клас для всіх документів):** Містить загальні властивості для всіх типів документів, такі як номер, дата та інформація про документ.
2. **Класи конкретних документів (Letter, Order, Directive, ResourceRequest):** Класи, які реалізують функціонал конкретних типів документів. Кожен з них містить специфічні властивості для свого типу документа.
3. **IDocumentFactory (Інтерфейс фабрики документів):** Визначає метод CreateDocument(), який використовується для створення об'єктів документів.
4. **Фабрики конкретних типів документів (LetterFactory, OrderFactory, DirectiveFactory, ResourceRequestFactory):** Кожна фабрика реалізує інтерфейс IDocumentFactory і має метод CreateDocument(), який повертає конкретний об'єкт документа.

UML-діаграма



Особливості реалізації обраного патерну:

1. **Гнучкість:** Фабричний метод дозволяє динамічно вибирати тип створюваних об'єктів, що робить код більш гнучким та легким для розширення.
2. **Інкапсуляція створення об'єктів:** Клієнтський код взаємодіє тільки з інтерфейсом фабрики, не знаючи конкретної реалізації створення об'єктів, що забезпечує відокремлення від реалізації.

Текст програми

```
using System;
using System.Collections.Generic;

public abstract class Document
{
    public int Number { get; set; }
    public DateTime Date { get; set; }
    public string Information { get; set; }
}
```

```

public class Letter : Document
{
    public bool IsIncoming { get; set; }
    public string Correspondent { get; set; }
}

public class Order : Document
{
    public string Department { get; set; }
    public DateTime Deadline { get; set; }
    public string Executor { get; set; }
}

public class Directive : Document
{
    public string Department { get; set; }
    public DateTime Deadline { get; set; }
}

public class ResourceRequest : Document
{
    public string Employee { get; set; }
    public List<string> Resources { get; set; }
}

public interface IDocumentFactory
{
    Document CreateDocument();
}

public class LetterFactory : IDocumentFactory
{
    public Document CreateDocument()
    {
        return new Letter();
    }
}

public class OrderFactory : IDocumentFactory
{
    public Document CreateDocument()
    {
        return new Order();
    }
}

public class DirectiveFactory : IDocumentFactory
{
    public Document CreateDocument()
    {
        return new Directive();
    }
}

public class ResourceRequestFactory : IDocumentFactory
{
    public Document CreateDocument()
    {
        return new ResourceRequest();
    }
}

class Program
{
    static void Main(string[] args)

```

```

{
    // Creating Documents by using Factory methods
    IDocumentFactory factory = new LetterFactory();
    Document letter = factory.CreateDocument() as Letter;
    letter.Number = 1;
    letter.Date = DateTime.Now;
    letter.Information = "This is letter";
    ((Letter)letter).IsIncoming = false;
    ((Letter)letter).Correspondent = "Communal factory";

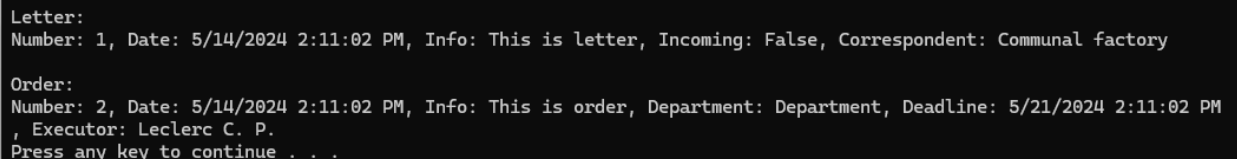
    factory = new OrderFactory();
    Document order = factory.CreateDocument() as Order;
    order.Number = 2;
    order.Date = DateTime.Now;
    order.Information = "This is order";
    ((Order)order).Department = "Department";
    ((Order)order).Deadline = DateTime.Now.AddDays(7);
    ((Order)order).Executor = "Leclerc C. P.";

    // Now we can get back to work with documents
    Console.WriteLine("Letter:");
    Console.WriteLine($"Number: {letter.Number}, Date: {letter.Date}, Info:
{letter.Information}, Incoming: {((Letter)letter).IsIncoming}, Correspondent:
{((Letter)letter).Correspondent}");

    Console.WriteLine("\nOrder:");
    Console.WriteLine($"Number: {order.Number}, Date: {order.Date}, Info:
{order.Information}, Department: {((Order)order).Department}, Deadline:
{((Order)order).Deadline}, Executor: {((Order)order).Executor}");
}
}

```

Скріншоти виконання



```

Letter:
Number: 1, Date: 5/14/2024 2:11:02 PM, Info: This is letter, Incoming: False, Correspondent: Communal factory

Order:
Number: 2, Date: 5/14/2024 2:11:02 PM, Info: This is order, Department: Department, Deadline: 5/21/2024 2:11:02 PM
, Executor: Leclerc C. P.
Press any key to continue . . .

```