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.

A close up of a text

Description automatically generated

**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**

A diagram of a document

Description automatically generated

**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;

interface IDocument

{

string GetInfo();

int Number { get; set; }

DateTime Date { get; set; }

string Info { get; set; }

}

abstract class Document : IDocument

{

public int Number { get; set; }

public DateTime Date { get; set; }

public string Info { get; set; }

public abstract string GetInfo();

}

class Letter : Document

{

public string Correspondent { get; set; }

public override string GetInfo()

{

return $"Letter: Number = {Number}, Date = {Date}, Info = {Info}, Correspondent = {Correspondent}";

}

}

class Order : Document

{

public string Department { get; set; }

public DateTime Deadline { get; set; }

public string Responsible { get; set; }

public override string GetInfo()

{

return $"Order: Number = {Number}, Date = {Date}, Info = {Info}, Department = {Department}, Deadline = {Deadline}, Responsible = {Responsible}";

}

}

class Directive : Document

{

public string Department { get; set; }

public DateTime Deadline { get; set; }

public override string GetInfo()

{

return $"Directive: Number = {Number}, Date = {Date}, Info = {Info}, Department = {Department}, Deadline = {Deadline}";

}

}

class ResourceRequest : Document

{

public string Employee { get; set; }

public List<string> Resources { get; set; }

public override string GetInfo()

{

return $"ResourceRequest: Number = {Number}, Date = {Date}, Info = {Info}, Employee = {Employee}, Resources = {string.Join(", ", Resources)}";

}

}

abstract class DocumentFactory

{

public abstract IDocument CreateDocument(string type);

}

class ConcreteDocumentFactory : DocumentFactory

{

public override IDocument CreateDocument(string type)

{

switch (type)

{

case "Letter":

return new Letter();

case "Order":

return new Order();

case "Directive":

return new Directive();

case "ResourceRequest":

return new ResourceRequest();

default:

throw new ArgumentException("Invalid document type");

}

}

}

class DocumentManager

{

private DocumentFactory factory;

public DocumentManager(DocumentFactory factory)

{

this.factory = factory;

}

public void CreateAndShowDocument(string type)

{

IDocument document = factory.CreateDocument(type);

// Налаштування документа

document.Number = new Random().Next(1, 1000);

document.Date = DateTime.Now;

document.Info = "This is a test document.";

// Виведення інформації про документ

Console.WriteLine(document.GetInfo());

}

public void CreateDocumentWithInput(string type)

{

IDocument document = factory.CreateDocument(type);

// Отримання даних від користувача

Console.WriteLine("Enter document number:");

document.Number = int.Parse(Console.ReadLine());

Console.WriteLine("Enter document date (yyyy-mm-dd):");

document.Date = DateTime.Parse(Console.ReadLine());

Console.WriteLine("Enter document info:");

document.Info = Console.ReadLine();

switch (type)

{

case "Letter":

var letter = document as Letter;

Console.WriteLine("Enter correspondent:");

letter.Correspondent = Console.ReadLine();

break;

case "Order":

var order = document as Order;

Console.WriteLine("Enter department:");

order.Department = Console.ReadLine();

Console.WriteLine("Enter deadline (yyyy-mm-dd):");

order.Deadline = DateTime.Parse(Console.ReadLine());

Console.WriteLine("Enter responsible person:");

order.Responsible = Console.ReadLine();

break;

case "Directive":

var directive = document as Directive;

Console.WriteLine("Enter department:");

directive.Department = Console.ReadLine();

Console.WriteLine("Enter deadline (yyyy-mm-dd):");

directive.Deadline = DateTime.Parse(Console.ReadLine());

break;

case "ResourceRequest":

var resourceRequest = document as ResourceRequest;

Console.WriteLine("Enter employee name:");

resourceRequest.Employee = Console.ReadLine();

resourceRequest.Resources = new List<string>();

Console.WriteLine("Enter resources (comma separated):");

var resources = Console.ReadLine().Split(',');

foreach (var resource in resources)

{

resourceRequest.Resources.Add(resource.Trim());

}

break;

default:

throw new ArgumentException("Invalid document type");

}

// Виведення інформації про документ

Console.WriteLine(document.GetInfo());

}

}

class Program

{

static void Main()

{

DocumentFactory factory = new ConcreteDocumentFactory();

DocumentManager manager = new DocumentManager(factory);

while (true)

{

Console.WriteLine("Enter document type (Letter, Order, Directive, ResourceRequest) or 'exit' to quit:");

string type = Console.ReadLine();

if (type.ToLower() == "exit")

{

break;

}

try

{

manager.CreateDocumentWithInput(type);

}

catch (Exception ex)

{

Console.WriteLine($"Error: {ex.Message}");

}

}

}

}

# Скріншоти виконання

A screen shot of a computer

Description automatically generated