МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

Розрахункова графічна робота

з дисципліни « АППЗ .NET »

Виконав: студент 2 курсу групи № 621П

121 «Інженерія програмного забезпечення»

Любімов Максим Олександрович

Прийняв:\_\_\_\_\_\_\_\_\_\_\_\_

Національна шкала:

Кількість балів:

[Варіант та постановка задачі 3](#_Toc1413567359)

[Теоретичні відомості 4](#_Toc679235031)

[Factory method 5](#_Toc1744317833)

[призначення та застосування шаблону проектування. 5](#_Toc631863768)

[опис шаблону проектування 5](#_Toc653508603)

[Flyweight 6](#_Toc1730899715)

[призначення та застосування шаблону проектування. 7](#_Toc48706168)

[опис шаблону проектування 7](#_Toc1500475361)

[Memento 7](#_Toc594104693)

[призначення та застосування шаблону проектування. 8](#_Toc440938253)

[опис шаблону проектування 8](#_Toc21791606)

[Thread-pool 8](#_Toc2026311501)

[призначення та застосування шаблону проектування. 9](#_Toc461536375)

[опис шаблону проектування 9](#_Toc1648849273)

[Лістинг програми 9](#_Toc511527218)

[Factory-method 10](#_Toc1142858108)

[Rgr-factory-methosd.csporj 10](#_Toc360894729)

[Program.cs 10](#_Toc1681202620)

[Flyweight 10](#_Toc463262706)

[Flyweight.csproj 10](#_Toc1132126524)

[Program.cs 10](#_Toc1514334854)

[Memento 11](#_Toc1362977941)

[Memento.csproj 11](#_Toc1717206669)

[Program.cs 11](#_Toc1906879717)

[Thread-pool 12](#_Toc1867041046)

[Program.cs 12](#_Toc495549424)

[Таблиця з позначкою виконання вимог на роботу 12](#_Toc1005708935)

## Варіант та постановка задачі

Вараіант №13



Самостійно знайти в мережі Інтернет (відповідно до варіанта

завдання в табл. 2) опис шаблонів проектування (Design patterns) наступних

типів:

− що породжує (Creational patterns);

− структурного (Structural patterns);

− поведінкового (Behavioral pattern);

− паралельних обчислень (Concurrency pattern).

У репозиторії GitHub створити файл ReadMe.md і на підставі зібраного

матеріалу сформувати текстовий опис шаблону та його графічне подання у

вигляді відповідних UML-діаграм:

− статичної моделі (діаграма класів та/або діаграма модулів);

− динамічної моделі (діаграма взаємодії та/або стану);

Для побудови діаграм використовувати інструмент візуалізації Mermaid [1],

який формує зображення з текстового опису на основі мови Markdown .

На практичному етапі для кожного шаблону проектування розробити

програмний проект, який демонструє особливості застосування заданих

шаблонів проектування практично.

Для завдання підвищеного рівня складності (див. табл. 1) необхідний

шаблон проектування (design pattern) оформити у вигляді zip-файлу, який є

шаблоном проекту або елемента ( Project / Item Template ) для середовища

розробки Visual Studio [2-5].

## Теоретичні відомості

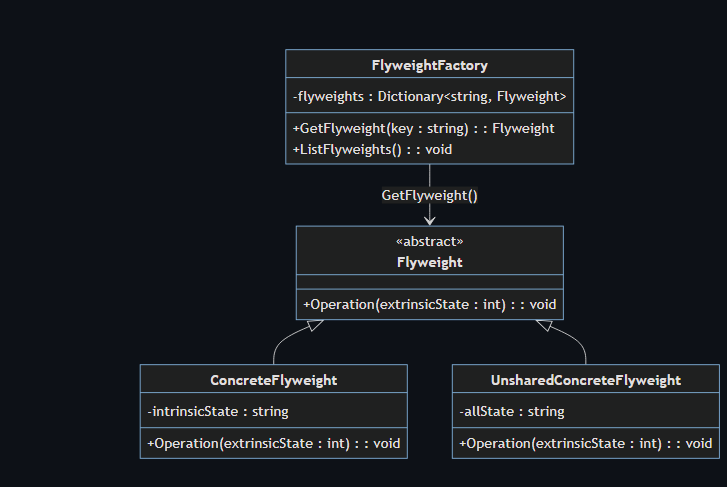
### Factory method

#### призначення та застосування шаблону проектування.

**Фабричний метод** - це паттерн проектування, який визначає загальний інтерфейс для створення об’єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об’єктів.

* Абстрактний клас Product, який визначає інтерфейс продуктів.
* Два конкретних продукти ConcreteProductA і ConcreteProductB, які наслідують Product.
* Абстрактний клас Creator, який оголошує фабричний метод FactoryMethod. Також він містить метод SomeOperation, який використовує продукт, створений фабричним методом.
* Два конкретних створювачі ConcreteCreatorA і ConcreteCreatorB, які наслідують Creator і реалізують фабричний метод для створення відповідних конкретних продуктів.
* В методі Main ми демонструємо створення конкретних продуктів за допомогою фабричного методу різних створювачів.

#### опис шаблону проектування



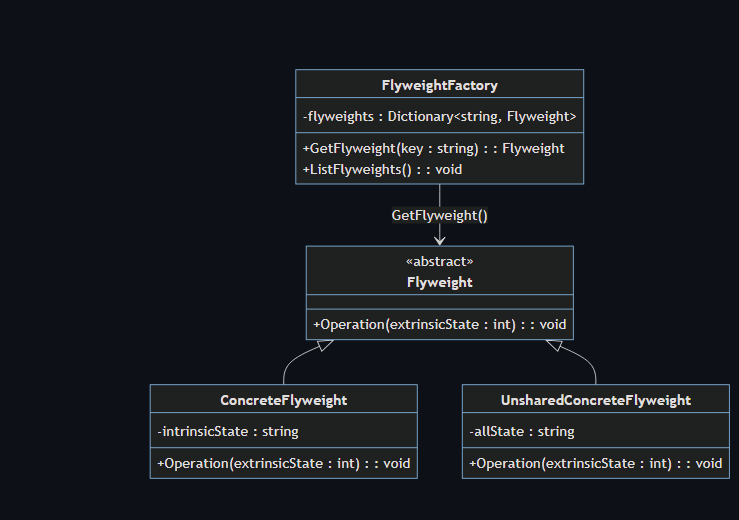
### Flyweight

#### призначення та застосування шаблону проектування.

Flyweight - це структурний паттерн проектування, який дозволяє ефективно використовувати оперативну пам’ять, розділяючи спільний стан об’єктів між ними, замість зберігання однакових даних в кожному об’єкті

* Flyweight: Абстрактний клас, який визначає інтерфейс для всіх конкретних легковаговиків.
* ConcreteFlyweight: Конкретний клас легковаговика, який зберігає внутрішній стан.
* UnsharedConcreteFlyweight: Легковаговик, який не розділяється між іншими об'єктами.
* FlyweightFactory: Фабрика, яка створює та зберігає об'єкти легковаговиків для їх повторного використання.
* Program: Демонстрація використання шаблону проектування Flyweight.

#### опис шаблону проектування



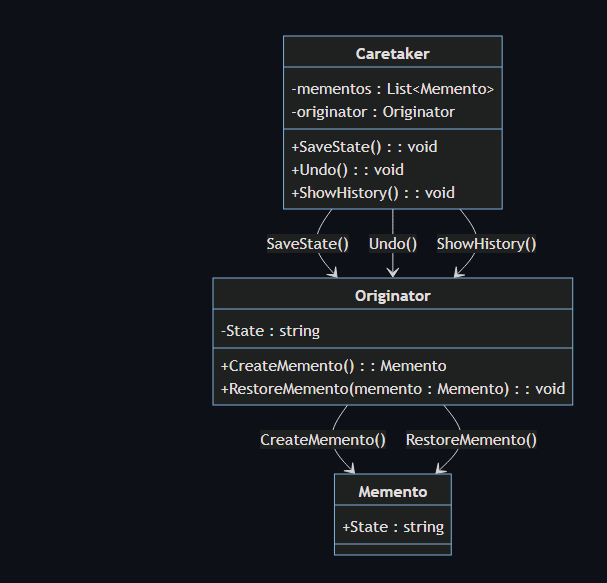
### Memento

#### призначення та застосування шаблону проектування.

*memento — це поведінковий паттерн проектування, який дозволяє зберігати та відновлювати попередні стани об’єктів, не розкриваючи деталей їх реалізації.*

* *Memento: Клас, який зберігає стан об'єкта.*
* *Originator: Клас, який створює знімки (Memento) і відновлює свій стан зі знімків.*
* *Caretaker: Клас, який керує знімками, зберігає їх і виконує операцію відновлення стану.*
* *Program: Демонстрація використання шаблону проектування Memento.*

#### опис шаблону проектування



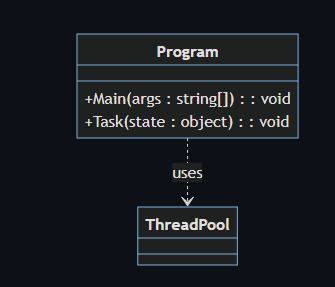
### Thread-pool

#### призначення та застосування шаблону проектування.

Thread-pool - це простий паттерн для керування одночасними запитами. Пул потоків дозволяє ефективно обробляти кілька запитів з необхідними обмеженнями. Цей паттерн зменшує накладні витрати від створення об’єктів потоків та обмежує доступні ресурси. Система може обробляти запити в межах своїх можливостей

* Встановлюємо максимальну кількість потоків у пулі за допомогою ThreadPool.SetMaxThreads.
* Додаємо 10 завдань до пулу потоків за допомогою ThreadPool.QueueUserWorkItem.
* Метод Task виконує завдання та імітує тривалу операцію за допомогою Thread.Sleep.

#### опис шаблону проектування



## Лістинг програми

### Factory-method

#### Rgr-factory-methosd.csporj

<Project Sdk="Microsoft.NET.Sdk"> <PropertyGroup> <OutputType>Exe</OutputType> <TargetFramework>net6.0</TargetFramework> <RootNamespace>rgr\_factory\_method</RootNamespace> <ImplicitUsings>enable</ImplicitUsings> <Nullable>enable</Nullable> </PropertyGroup> </Project>

#### Program.cs

**using** System; // Продукт **public** **abstract** **class** **Product** { **public** **abstract** **void** **Operation**(); } // Конкретний продукт A **public** **class** **ConcreteProductA** : **Product** { **public** **override** **void** **Operation**() { Console.WriteLine("Operation of ConcreteProductA"); } } // Конкретний продукт B **public** **class** **ConcreteProductB** : **Product** { **public** **override** **void** **Operation**() { Console.WriteLine("Operation of ConcreteProductB"); } } // Створювач **public** **abstract** **class** **Creator** { **public** **abstract** Product **FactoryMethod**(); **public** **void** **SomeOperation**() { // Використання фабричного методу для створення продукту **var** product = FactoryMethod(); product.Operation(); } } // Конкретний створювач A **public** **class** **ConcreteCreatorA** : **Creator** { **public** **override** Product **FactoryMethod**() { **return** **new** ConcreteProductA(); } } // Конкретний створювач B **public** **class** **ConcreteCreatorB** : **Creator** { **public** **override** Product **FactoryMethod**() { **return** **new** ConcreteProductB(); } } **class** **Program** { **static** **void** **Main**(string[] args) { // Використання конкретного створювача A Creator creatorA = **new** ConcreteCreatorA(); creatorA.SomeOperation(); // Використання конкретного створювача B Creator creatorB = **new** ConcreteCreatorB(); creatorB.SomeOperation(); } }

### Flyweight

#### Flyweight.csproj

<Project Sdk="Microsoft.NET.Sdk"> <PropertyGroup> <OutputType>Exe</OutputType> <TargetFramework>net6.0</TargetFramework> <RootNamespace>rgr\_flyweight</RootNamespace> <ImplicitUsings>enable</ImplicitUsings> <Nullable>enable</Nullable> </PropertyGroup> </Project>

#### Program.cs

**using** System; **using** System.Collections.Generic; // Flyweight **public** **abstract** **class** **Flyweight** { **public** **abstract** **void** **Operation**(int extrinsicState); } // ConcreteFlyweight **public** **class** **ConcreteFlyweight** : **Flyweight** { **private** string intrinsicState; **public** **ConcreteFlyweight**(string intrinsicState) { **this**.intrinsicState = intrinsicState; } **public** **override** **void** **Operation**(int extrinsicState) { Console.WriteLine($"ConcreteFlyweight: Intrinsic State = {intrinsicState}, Extrinsic State = {extrinsicState}"); } } // UnsharedConcreteFlyweight **public** **class** **UnsharedConcreteFlyweight** : **Flyweight** { **private** string allState; **public** **UnsharedConcreteFlyweight**(string allState) { **this**.allState = allState; } **public** **override** **void** **Operation**(int extrinsicState) { Console.WriteLine($"UnsharedConcreteFlyweight: All State = {allState}, Extrinsic State = {extrinsicState}"); } } // FlyweightFactory **public** **class** **FlyweightFactory** { **private** Dictionary<string, Flyweight> flyweights = **new** Dictionary<string, Flyweight>(); **public** Flyweight **GetFlyweight**(string key) { **if** (!flyweights.ContainsKey(key)) { flyweights[key] = **new** ConcreteFlyweight(key); } **return** flyweights[key]; } **public** **void** **ListFlyweights**() { Console.WriteLine($"FlyweightFactory: {flyweights.Count} flyweights in total."); **foreach** (**var** key **in** flyweights.Keys) { Console.WriteLine(key); } } } **class** **Program** { **static** **void** **Main**(string[] args) { **var** factory = **new** FlyweightFactory(); // Using shared flyweights **var** flyweight1 = factory.GetFlyweight("State1"); **var** flyweight2 = factory.GetFlyweight("State2"); **var** flyweight3 = factory.GetFlyweight("State1"); flyweight1.Operation(1); flyweight2.Operation(2); flyweight3.Operation(3); factory.ListFlyweights(); // Using unshared flyweight **var** unsharedFlyweight = **new** UnsharedConcreteFlyweight("Unshared State"); unsharedFlyweight.Operation(4); } }

### Memento

#### Memento.csproj

<Project Sdk="Microsoft.NET.Sdk"> <PropertyGroup> <OutputType>Exe</OutputType> <TargetFramework>net6.0</TargetFramework> <RootNamespace>rgr\_memento</RootNamespace> <ImplicitUsings>enable</ImplicitUsings> <Nullable>enable</Nullable> </PropertyGroup> </Project>

#### Program.cs

**using** System; **using** System.Collections.Generic; // Memento **public** **class** **Memento** { **public** string State { **get**; **private** **set**; } **public** **Memento**(string state) { State = state; } } // Originator **public** **class** **Originator** { **public** string State { **get**; **set**; } **public** Memento **CreateMemento**() { **return** **new** Memento(State); } **public** **void** **RestoreMemento**(Memento memento) { State = memento.State; } } // Caretaker **public** **class** **Caretaker** { **private** List<Memento> mementos = **new** List<Memento>(); **private** Originator originator; **public** **Caretaker**(Originator originator) { **this**.originator = originator; } **public** **void** **SaveState**() { mementos.Add(originator.CreateMemento()); } **public** **void** **Undo**() { **if** (mementos.Count > 0) { **var** memento = mementos[mementos.Count - 1]; originator.RestoreMemento(memento); mementos.RemoveAt(mementos.Count - 1); } } **public** **void** **ShowHistory**() { Console.WriteLine("History of states:"); **foreach** (**var** memento **in** mementos) { Console.WriteLine(memento.State); } } } **class** **Program** { **static** **void** **Main**(string[] args) { **var** originator = **new** Originator(); **var** caretaker = **new** Caretaker(originator); originator.State = "State1"; caretaker.SaveState(); originator.State = "State2"; caretaker.SaveState(); originator.State = "State3"; Console.WriteLine($"Current State: {originator.State}"); caretaker.Undo(); Console.WriteLine($"Restored State: {originator.State}"); caretaker.Undo(); Console.WriteLine($"Restored State: {originator.State}"); caretaker.ShowHistory(); } }

### Thread-pool

<Project Sdk="Microsoft.NET.Sdk"> <PropertyGroup> <OutputType>Exe</OutputType> <TargetFramework>net6.0</TargetFramework> <RootNamespace>rgr\_thread\_pool</RootNamespace> <ImplicitUsings>enable</ImplicitUsings> <Nullable>enable</Nullable> </PropertyGroup> </Project>

#### Program.cs

**using** System; **using** System.Threading; **class** **Program** { **static** **void** **Main**(string[] args) { // Максимальна кількість потоків у пулі ThreadPool.SetMaxThreads(5, 5); // Виконання 10 завдань у пулі потоків **for** (int i = 0; i < 10; i++) { int taskNumber = i; ThreadPool.QueueUserWorkItem(Task, taskNumber); } // Зачекати, поки всі завдання завершаться Console.WriteLine("Натиснiть будь-яку клавiшу для виходу..."); Console.ReadKey(); } **static** **void** **Task**(object state) { int taskNumber = (int)state; Console.WriteLine($"Завдання {taskNumber} розпочато у потоцi {Thread.CurrentThread.ManagedThreadId}"); // Імітація тривалої операції Thread.Sleep(1000); Console.WriteLine($"Завдання {taskNumber} завершено у потоцi {Thread.CurrentThread.ManagedThreadId}"); } }

## Таблиця з позначкою виконання вимог на роботу

