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 робота**

з дисципліни « Архітектура та проектування ПЗ »

*назва дисципліни*

на тему: ШАБЛОНИ ПРОЕКТУВАННЯ

Виконав: студент 2 курсу групи № 621п

напряму підготовки (спеціальності)

121 інженерія программного забезпечення

(шифр і назва напряму підготовки (спеціальності))

Бобро М.Ю.

(прізвище й ініціали студента)

Прийняв: ст. викладач каф. 603

Сьомочкін М. О.

(посада, науковий ступінь, прізвище й ініціали)

Національна шкала:

Кількість балів:

Харків – 2024

**Розрахунково-графічна робота   
«Шаблони проектування»**

**Мета роботи**: Вивчення стандартних ситуацій у процесі розробки складних програмних проектів та застосування шаблонів проектування (***Design patterns***) для їх вирішення.

**Завдання**. 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].

# **ВИКОНАННЯ РОБОТИ**

**Варіант 3**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Creational pattern*** | ***Structural pattern*** | ***Behavioral pattern*** | ***Concurrency pattern*** |
| Multiton | Adapter or Wrapper | Protocol stack | Asynchronous method invocation |

**Multiton (Creational Pattern)**

**Описание:**

Multiton является вариацией Singleton, позволяя иметь несколько уникальных экземпляров класса, гарантируя, что для каждого ключа будет создан только один экземпляр. Это полезно, когда требуется несколько экземпляров для различных контекстов, но каждый экземпляр должен быть единственным в своем роде для данного ключа.

**Основные составные части:**

Класс Multiton: управляет экземплярами и гарантирует их уникальность для каждого ключа.

**UML-діаграми (рисунок 1 – 2):**

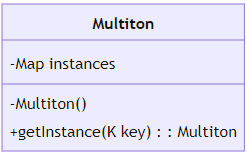


Рисунок 1. Static model

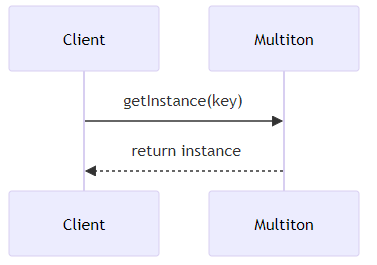


Рисунок 2. Dynamic model

2. Adapter or Wrapper (Structural Pattern)

\*\*Описание:\*\*

Adapter позволяет объектам с несовместимыми интерфейсами работать вместе. Он оборачивает объект, чтобы он соответствовал нужному интерфейсу.

\*\*Основные составные части:\*\*

- \*\*Target:\*\* определяет интерфейс, который клиент использует.

- \*\*Adapter:\*\* реализует интерфейс Target и оборачивает объект Adaptee.

- \*\*Adaptee:\*\* представляет существующий интерфейс, который нужно адаптировать.

\*\*Источник:\*\* [Adapter Pattern](https://sourcemaking.com/design\_patterns/adapter)

**UML-діаграми (рисунок 3 – 4):**

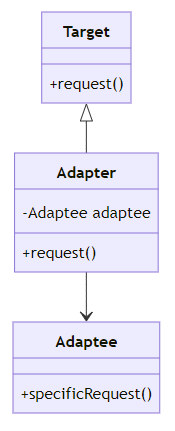


Рисунок 3. Static model

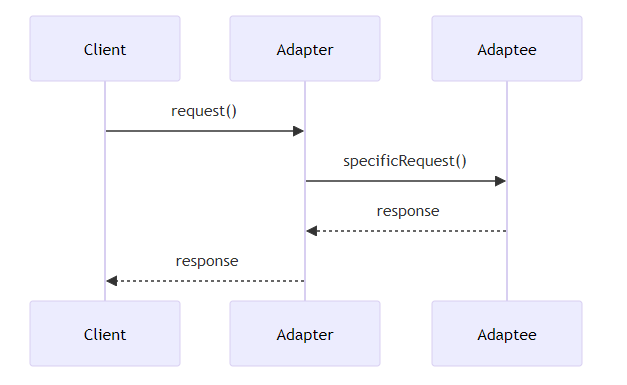


Рисунок 4. Dynamic model

3. Protocol Stack (Behavioral Pattern)

\*\*Описание:\*\*

Protocol Stack организует протоколы связи в стек, где каждый уровень отвечает за отдельный аспект коммуникации.

\*\*Основные составные части:\*\*

- \*\*ProtocolStack:\*\* управляет уровнями протоколов.

- \*\*ProtocolLayer:\*\* представляет отдельный уровень протокола.

\*\*Источник:\*\* [Protocol Stack Pattern](<https://en.wikipedia.org/wiki/Protocol_stack>)

**UML-діаграма (рисунок 5 – 6):**

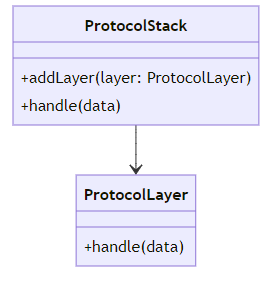


Рисунок 5. State model

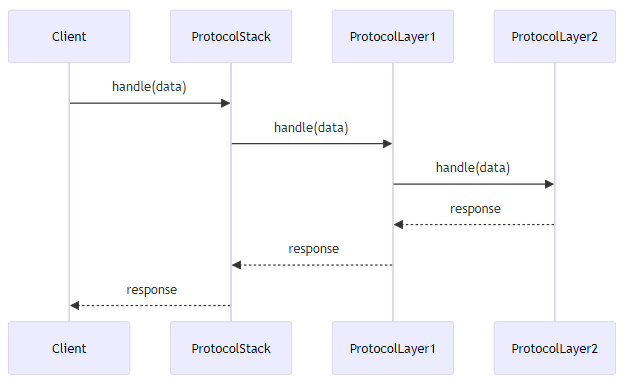


Рисунок 6. Dynamic model

4. Asynchronous Method Invocation (Concurrency Pattern)

\*\*Описание:\*\*

Asynchronous Method Invocation (AMI) позволяет асинхронно вызывать методы, не блокируя текущий поток выполнения.

\*\*Основные составные части:\*\*

- \*\*AsyncInvoker:\*\* отвечает за асинхронный вызов метода.

- \*\*Method:\*\* представляет метод, который будет вызван асинхронно.

\*\*Источник:\*\* [Asynchronous Method Invocation](https://sourcemaking.com/design\_patterns/asynchronous\_method\_invocation)

**UML-діаграми (рисунок 7 – 8):**

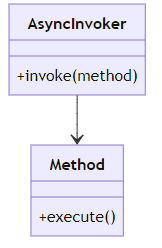


Рисунок 7. Static model

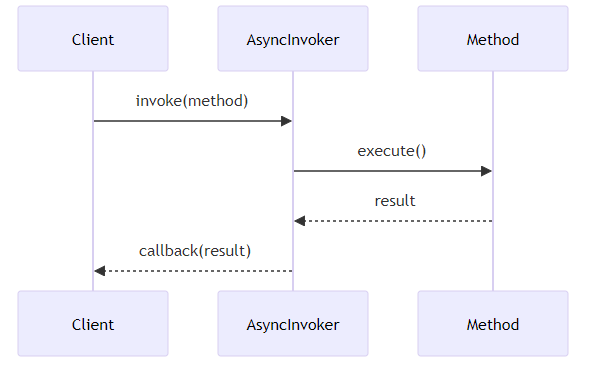


Рисунок 8. Dynamic model

# **ЛІСТИНГ ПРОГРАМ**

Multiton (Creational Pattern)

Программный проект на C#

using System;

using System.Collections.Generic;

public class Multiton

{

private static Dictionary<string, Multiton> \_instances = new Dictionary<string, Multiton>();

private string \_key;

private Multiton(string key)

{

\_key = key;

}

public static Multiton GetInstance(string key)

{

if (!\_instances.ContainsKey(key))

{

\_instances[key] = new Multiton(key);

}

return \_instances[key];

}

public void Display()

{

Console.WriteLine("Instance with key: " + \_key);

}

}

class Program

{

static void Main()

{

Multiton instance1 = Multiton.GetInstance("key1");

Multiton instance2 = Multiton.GetInstance("key2");

Multiton instance3 = Multiton.GetInstance("key1");

instance1.Display(); // Output: Instance with key: key1

instance2.Display(); // Output: Instance with key: key2

instance3.Display(); // Output: Instance with key: key1

}

}

Реализация на дополнительном языке (Python)

class Multiton:

\_instances = {}

def \_\_new\_\_(cls, key):

if key not in cls.\_instances:

cls.\_instances[key] = super(Multiton, cls).\_\_new\_\_(cls)

return cls.\_instances[key]

def \_\_init\_\_(self, key):

self.key = key

def display(self):

print(f"Instance with key: {self.key}")

# Использование

instance1 = Multiton('key1')

instance2 = Multiton('key2')

instance3 = Multiton('key1')

instance1.display() # Output: Instance with key: key1

instance2.display() # Output: Instance with key: key2

instance3.display() # Output: Instance with key: key1

Adapter or Wrapper (Structural Pattern)

Программный проект на C#

using System;

public class Adaptee

{

public string SpecificRequest()

{

return "Adaptee's specific request";

}

}

public interface ITarget

{

string Request();

}

public class Adapter : ITarget

{

private Adaptee \_adaptee;

public Adapter(Adaptee adaptee)

{

\_adaptee = adaptee;

}

public string Request()

{

return \_adaptee.SpecificRequest();

}

}

class Program

{

static void Main()

{

Adaptee adaptee = new Adaptee();

ITarget adapter = new Adapter(adaptee);

Console.WriteLine(adapter.Request());

}

}

Реализация на дополнительном языке (Python)

python

class Adaptee:

def specific\_request(self):

return "Adaptee's specific request"

class Adapter:

def \_\_init\_\_(self, adaptee):

self.adaptee = adaptee

def request(self):

return self.adaptee.specific\_request()

# Использование

adaptee = Adaptee()

adapter = Adapter(adaptee)

print(adapter.request()) # Output: Adaptee's specific request

Protocol Stack (Behavioral Pattern)

Программный проект на C#

using System;

using System.Collections.Generic;

public abstract class ProtocolLayer

{

public abstract string Handle(string data);

}

public class ProtocolStack

{

private List<ProtocolLayer> \_layers = new List<ProtocolLayer>();

public void AddLayer(ProtocolLayer layer)

{

\_layers.Add(layer);

}

public string Handle(string data)

{

foreach (var layer in \_layers)

{

data = layer.Handle(data);

}

return data;

}

}

public class Layer1 : ProtocolLayer

{

public override string Handle(string data)

{

return $"Layer1 processed {data}";

}

}

public class Layer2 : ProtocolLayer

{

public override string Handle(string data)

{

return $"Layer2 processed {data}";

}

}

class Program

{

static void Main()

{

ProtocolStack stack = new ProtocolStack();

stack.AddLayer(new Layer1());

stack.AddLayer(new Layer2());

string result = stack.Handle("data");

Console.WriteLine(result); // Output: Layer2 processed Layer1 processed data

}

}

Реализация на дополнительном языке (Python)

class ProtocolLayer:

def handle(self, data):

pass

class ProtocolStack:

def \_\_init\_\_(self):

self.layers = []

def add\_layer(self, layer):

self.layers.append(layer)

def handle(self, data):

for layer in self.layers:

data = layer.handle(data)

return data

class Layer1(ProtocolLayer):

def handle(self, data):

return f"Layer1 processed {data}"

class Layer2(ProtocolLayer):

def handle(self, data):

return f"Layer2 processed {data}"

# Использование

stack = ProtocolStack()

stack.add\_layer(Layer1())

stack.add\_layer(Layer2())

print(stack.handle("data")) # Output: Layer2 processed Layer1 processed data

Asynchronous Method Invocation (Concurrency Pattern)

Программный проект на C#

using System;

using System.Threading;

using System.Threading.Tasks;

public class AsyncInvoker

{

public async Task Invoke(Func<string> method, Action<string> callback)

{

var result = await Task.Run(method);

callback(result);

}

}

class Program

{

static void Main()

{

AsyncInvoker invoker = new AsyncInvoker();

invoker.Invoke(SomeMethod, Callback).Wait();

}

static string SomeMethod()

{

return "Method executed";

}

static void Callback(string result)

{

Console.WriteLine(result);

}

}

Реализация на дополнительном языке (Python)

import threading

class AsyncInvoker:

def invoke(self, method, callback):

def run():

result = method()

callback(result)

thread = threading.Thread(target=run)

thread.start()

def some\_method():

return "Method executed"

def callback(result):

print(result)

# Использование

invoker = AsyncInvoker()

invoker.invoke(some\_method, callback)

|  |  |  |
| --- | --- | --- |
| Таблиця з позначкою вимог до роботи (табл. 1):  Таблиця 1 | | |
| Вимоги до роботи | % | Оцінка |
| Текстовий опис шаблону, його основних складових частин, їх призначення (*з посиланням на першоджерело!* ) | 5 |  |
| ***UML*** модель шаблону у вигляді ***Mermaid*** діаграми | 5 |  |
| Програмний проект на *С#* , що ­демонструє ­специфіку шаблону проектування (кожен шаблон окремо) | 5 |  |
| Реалізація шаблону проектування додатковою (будь-якою) мовою програмування | 5 |  |
| Створення шаблону проектування (***design******pattern***) у вигляді шаблону проекту або елемента (***project*** */* ***item******template***) для ***Visual******Studio*** (***zip*** -файл) | 5 |  |

# **ВИСНОВОК**

Під час виконання розрахункової роботи було вивчено стандартні ситуації у процесі розробки складних програмних проектів та застосування шаблонів проектування (Design patterns) для їх вирішення. У нашому випадку були розглянуті такі паттерни як Multiton, Adapter, Protocol Stack та Asynchronous Method Invocation. Був проведений аналіз заданих паттернів у вигляді створення UML-діаграм, програмних прикладів, шаблонів для Visual Studio та реалізацій на декількох мовах програмування. Оформлено звіт та розміщено результати у репозиторії GitHub для подальшого використання.