МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Розрахунково-графічна робота**

з дисципліни «Архітектура та проектування програмного

забезпечення .Net »

*назва дисципліни*

на тему: «Шаблони проектування»

Виконав: студент 2 курсу групи № 621п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Конарєва В.В.

(прізвище й ініціали студента)

Прийняв:к.т.н., доц., доцент каф. 603

Лучшев П.О.

(посада, науковий ступінь, прізвище й

ініціали)

Кількість балів:

Харків – 2025

**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.

Додатково кожний шаблон проектування (***design pattern***) за варіантом

створити у вигляді ***zip***-файлу, який є шаблоном проекту або елемента ( ***Project***

***/ Item Template*** ) для середовища розробки ***Visual Studio*** [2-5].

**Постановка задачі**

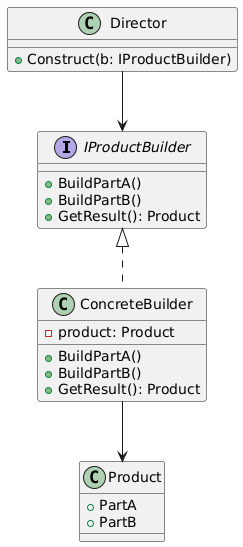
Варіант 621п.09

– «Реалізувати 4 патерни: Builder, Front Controller, Design Servant, Active Object. Побудувати UML-діаграми. Створити шаблон VS для одного патерну.»

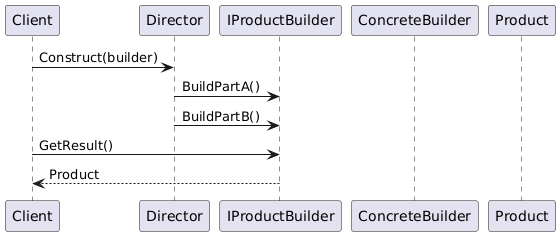
**Теоретичні відомості**  
– **Builder**: відокремлює конструювання складного об’єкта від його представлення.  
– **Front Controller**: забезпечує єдину точку входу для обробки запитів.  
– **Design Servant**: виділення служби (Servant) для виконання операцій над різними об’єктами.  
– **Active Object**: розділяє виклик операції та її виконання в іншому потоці, повертаючи майбутній результат (Future).

**UML-діаграми**   
Для кожного патерну – статична (класів) та динамічна (секвенс).

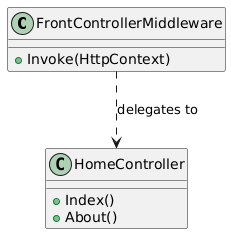
Builder – класова діаграма



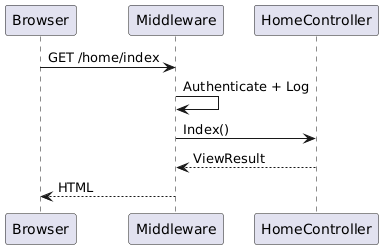
Builder – секвенс-діаграма



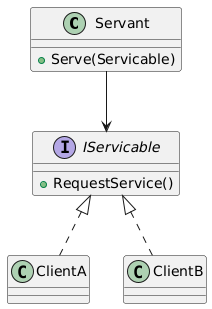
Front Controller – класова діаграма



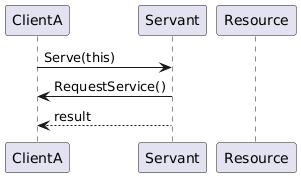
Front Controller – секвенс-діаграма



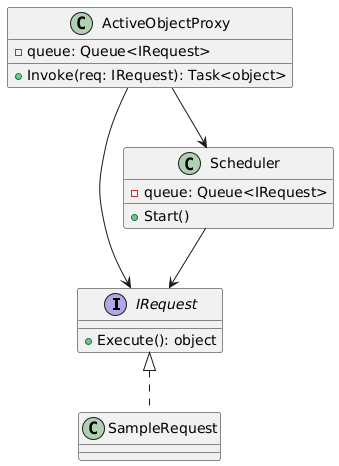
Design Servant – класова діаграма



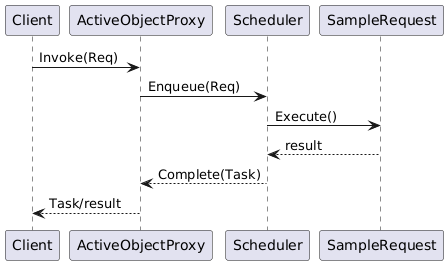
Design Servant – секвенс-діаграма



Active Object – класова діаграма



Active Object – секвенс-діаграма



**Листинги програм**

**BuilderExample/Program.cs**

using System;

namespace BuilderExample {

class Program {

static void Main() {

var director = new Director();

var builder = new ConcreteBuilder();

director.Construct(builder);

var product = builder.GetResult();

Console.WriteLine($"Parts: {product.PartA}, {product.PartB}");

}

}

}

BuilderExample/IProductBuilder.cs

namespace BuilderExample {

public interface IProductBuilder {

void BuildPartA();

void BuildPartB();

Product GetResult();

}

}

BuilderExample/Product.cs

namespace BuilderExample {

public class Product {

public string PartA { get; set; }

public string PartB { get; set; }

}

}

BuilderExample/Director.cs

namespace BuilderExample {

public class Director {

public void Construct(IProductBuilder builder) {

builder.BuildPartA();

builder.BuildPartB();

}

}

}

FrontControllerExample/Startup.cs

using Microsoft.AspNetCore.Builder;

using Microsoft.Extensions.DependencyInjection;

namespace FrontControllerExample {

public class Startup {

public void ConfigureServices(IServiceCollection services) {

services.AddControllersWithViews();

}

public void Configure(IApplicationBuilder app) {

app.UseMiddleware<Middleware.FrontControllerMiddleware>();

app.UseRouting();

app.UseEndpoints(endpoints => {

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

});

}

}

}

FrontControllerExample/Middleware/FrontControllerMiddleware.cs

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

namespace FrontControllerExample.Middleware {

public class FrontControllerMiddleware {

private readonly RequestDelegate \_next;

public FrontControllerMiddleware(RequestDelegate next) => \_next = next;

public async Task Invoke(HttpContext ctx) {

// авторизація, логування

if (!ctx.User.Identity.IsAuthenticated) {

ctx.Response.Redirect("/Home/Login");

return;

}

await \_next(ctx);

}

}

}

FrontControllerExample/Controllers/HomeController.cs

using Microsoft.AspNetCore.Mvc;

namespace FrontControllerExample.Controllers {

public class HomeController : Controller {

public IActionResult Index() => View();

public IActionResult About() => View();

public IActionResult Login() => View();

}

}

FrontControllerExample/Program.cs

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Hosting;

namespace FrontControllerExample {

public class Program {

public static void Main(string[] args) =>

CreateHostBuilder(args).Build().Run();

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder => webBuilder.UseStartup<Startup>());

}

}

ServantExample/Servant.cs

namespace ServantExample {

public class Servant {

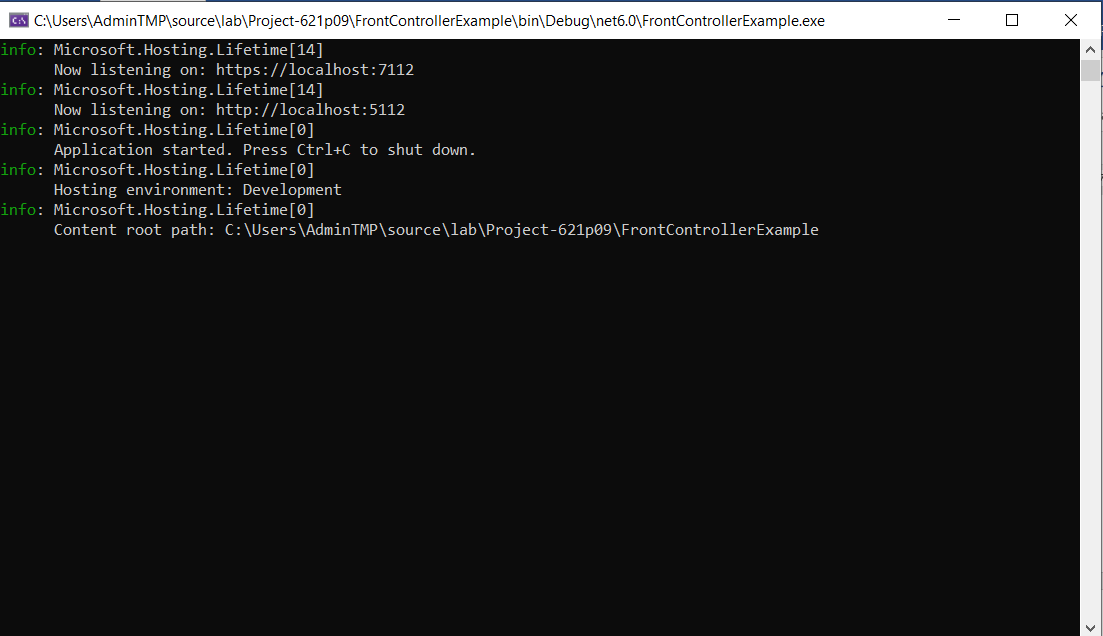
public string Serve(IServicable client) {

return client.RequestService().ToUpper();

}

}

}



ServantExample/ClientA.cs

namespace ServantExample {

public class ClientA : IServicable {

public string RequestService() => "Data from A";

}

}

ServantExample/ClientB.cs

namespace ServantExample {

public class ClientB : IServicable {

public string RequestService() => "Info from B";

}

}

ServantExample/Program.cs

using System;

namespace ServantExample {

class Program {

static void Main() {

var servant = new Servant();

foreach (var client in new IServicable[]{ new ClientA(), new ClientB() }) {

Console.WriteLine(servant.Serve(client));

}

}

}

}

ActiveObjectExample/IRequest.cs

using System.Threading.Tasks;

namespace ActiveObjectExample {

public interface IRequest {

object Execute();

}

}

ActiveObjectExample/SampleRequest.cs

namespace ActiveObjectExample {

public class SampleRequest : IRequest {

private readonly int \_x;

public SampleRequest(int x) => \_x = x;

public object Execute() {

// складна операція

return \_x \* \_x;

}

}

}

ActiveObjectExample/ActiveObjectProxy.cs

using System.Collections.Concurrent;

using System.Threading.Tasks;

namespace ActiveObjectExample {

public class ActiveObjectProxy {

private readonly BlockingCollection<IRequest> \_queue = new();

public ActiveObjectProxy() {

var scheduler = new Scheduler(\_queue);

scheduler.Start();

}

public Task<object> Invoke(IRequest req) {

var tcs = new TaskCompletionSource<object>();

\_queue.Add(new WrappedRequest(req, tcs));

return tcs.Task;

}

private record WrappedRequest(IRequest Req, TaskCompletionSource<object> Tcs) : IRequest {

public object Execute() {

var r = Req.Execute();

Tcs.SetResult(r);

return r;

}

}

} }

ActiveObjectExample/Scheduler.cs

using System.Collections.Concurrent;

using System.Threading;

namespace ActiveObjectExample {

public class Scheduler {

private readonly BlockingCollection<IRequest> \_queue;

public Scheduler(BlockingCollection<IRequest> queue) => \_queue = queue;

public void Start() {

var thread = new Thread(() => {

foreach (var req in \_queue.GetConsumingEnumerable())

req.Execute();

});

thread.IsBackground = true;

thread.Start();

}

}

}

ActiveObjectExample/Program.cs

using System;

using System.Threading.Tasks;

namespace ActiveObjectExample {

class Program {

static async Task Main() {

var proxy = new ActiveObjectProxy();

Task<object> t = proxy.Invoke(new SampleRequest(7));

Console.WriteLine("Request sent...");

var result = await t;

Console.WriteLine($"Result: {result}");

}

}

}

**Таблиця відповідності вимогам**

|  |  |  |
| --- | --- | --- |
| **№** | **Вимоги** | **Виконано** |
| 1 | Builder | ✓ |
| 2 | Front Controller | ✓ |
| 3 | Design Servant | ✓ |
| 4 | Active Object | ✓ |
| 5 | UML-діаграми | ✓ |