Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

Лабораторна робота № 5

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Виконав
Студент групи ІА-31:
Підковка Д. О.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

1. **Мета:**

Вивчити структуру шаблонів «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype» та навчитися застосовувати їх в реалізації програмної системи.

2. **Теоретичні відомості:**

Adapter: Адаптує інтерфейс одного об’єкта до іншого, дозволяючи несумісним компонентам працювати разом (наприклад, уніфікація інтерфейсів бібліотек принтерів). Спрощує інтеграцію без змін у коді.
Builder: Відокремлює процес створення об’єкта від його представлення, доречний для складних або багатоформних конструкцій (наприклад, відповіді веб-сервера). Забезпечує гнучкий контроль.
Command: Перетворює виклик методу в об’єкт, дозволяючи гнучкі системи команд із відміною, логуванням чи плануванням (наприклад, дії в інтерфейсі). Покращує модульність.
Chain of Responsibility: Передає запити по ланцюжку обробників, поки один не виконає (наприклад, контекстні меню). Зменшує зв’язки та спрощує зміни.
Prototype: Створює об’єкти клонуванням прототипу, корисний для складних об’єктів або динамічних змін (наприклад, редактор рівнів гри). Зменшує ієрархію спадкування.

3. **Хід роботи:**

Тема :

6. **Web-browser** (proxy, chain of responsibility, factory method, template method, visitor, p2p)

Веб-браузер повинен мати можливість зробити наступне: мати адресний рядок для введення адреси сайту, переміщатися і відображати структуру html документа, переглядати підключений javascript та css файли, перегляд всіх підключених ресурсів (зображень), коректна обробка відповідей з сервера (коди відповідей HTTP) – переходи при перенаправленнях, відображення сторінок 404 і 502/503.

1) Ознайомитись з короткими теоретичними відомостями.
2) Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3) Реалізувати один з розглянутих шаблонів за обраною темою.
4) Реалізувати не менше 3-х класів відповідно до обраної теми.

5) Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

Патерн: Chain of Responsibility – каскадна обробка кодів стану HTTP та помилок навігації.

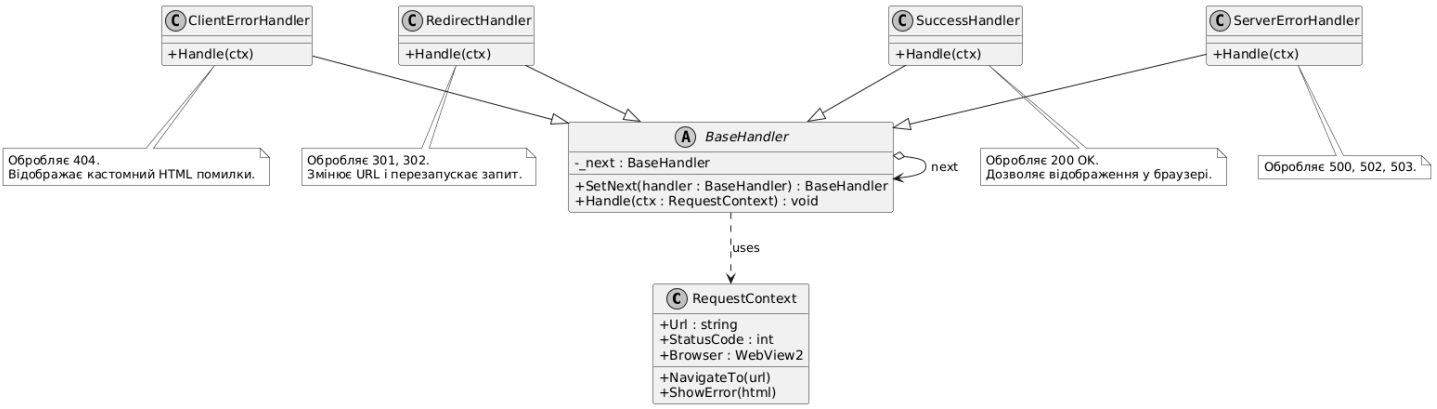


Рисунок 1 - Структура патерну Adapter

У даному застосунку реалізовано поведінковий шаблон проєктування Chain of Responsibility (Ланцюжок відповідальності), який дозволяє уникнути жорсткої прив'язки відправника запиту (браузера) до конкретних класів обробки помилок, передаючи запит вздовж ланцюжка потенційних обробників.

Абстрактний клас BaseHandler виступає контрактом для всіх обробників — він визначає метод Handle(RequestContext ctx), який приймає контекст запиту (URL та HTTP-код), а також механізм передачі управління наступному елементу (_nextHandler).

Конкретні класи-обробники (RedirectHandler, ClientErrorHandler, ServerErrorHandler, SuccessHandler) реалізують специфічну логіку реакції на статус-коди. Кожен обробник самостійно вирішує: якщо код відповіді входить у його зону відповідальності (наприклад, 404 для ClientErrorHandler), він обробляє запит (генерує сторінку помилки) і перериває ланцюжок. Якщо ні — передає контекст далі.

Клас MainWindow виступає клієнтом, який лише формує ланцюжок і запускає перевірку URL, не заглиблюючись у деталі того, як саме буде оброблено перенаправлення або помилка сервера. Такий підхід забезпечує послаблення зв'язності (loose coupling) і дозволяє легко додавати нові типи обробників (наприклад, для кодів аутентифікації 401/403) без модифікації основного коду навігації.

```
namespace WebBrowser.Coursework.Patterns
{
    7 references
    public abstract class BaseHandler
    {
        protected BaseHandler _nextHandler;

        3 references
        public BaseHandler SetNext(BaseHandler handler)
        {
            _nextHandler = handler;
            return handler;
        }

        11 references
        public virtual void Handle(RequestContext ctx)
        {

            if (_nextHandler != null)
            {
                _nextHandler.Handle(ctx);
            }
        }
    }
}
```

Рисунок 2 – BaseHandler.cs

```
using System.Windows;

namespace WebBrowser.Coursework.Patterns
{

    1 reference
    public class RedirectHandler : BaseHandler
    {
        8 references
        public override void Handle(RequestContext ctx)
        {
            if (ctx.StatusCode >= 300 && ctx.StatusCode < 400)
            {
                MessageBox.Show($"Виявлено перенаправлення (Code {ctx.StatusCode}).", "Redirect Handler");

                base.Handle(ctx);
            }
            else
            {
                base.Handle(ctx);
            }
        }
    }


    1 reference
    public class ClientErrorHandler : BaseHandler
    {
        7 references
        public override void Handle(RequestContext ctx)
        {
            if (ctx.StatusCode >= 400 && ctx.StatusCode < 500)
            {
                if (ctx.StatusCode == 404)
                {
                    ctx.RenderHtml("404 Not Found", "Вибачте, але сторінку, яку ви шукаєте, не знайдено.", "orange");
                }
                else
                {
                    ctx.RenderHtml($"Error {ctx.StatusCode}", "Виникла помилка на стороні клієнта.", "orange");
                }

            }
            else
            {
                base.Handle(ctx);
            }
        }
    }
```

```
    1 reference
    public class ServerErrorHandler : BaseHandler
    {
        7 references
        public override void Handle(RequestContext ctx)
        {
            if (ctx.StatusCode >= 500)
            {
                string msg = ctx.StatusCode == 503 ? "Сервіс тимчасово недоступний." : "Внутрішня помилка сервера.";
                ctx.RenderHtml($"Server Error {ctx.StatusCode}", msg, "red");
            }
            else
            {
                base.Handle(ctx);
            }
        }
    }

    // 4. Успішне завантаження (200 OK)
    1 reference
    public class SuccessHandler : BaseHandler
    {
        7 references
        public override void Handle(RequestContext ctx)
        {
            if (ctx.StatusCode >= 200 && ctx.StatusCode < 300)
            {
                if (ctx.Browser.Source?.ToString() != ctx.Url)
                {
                    ctx.Browser.Source = new System.Uri(ctx.Url);
                }
            }
            else
            {
                base.Handle(ctx);
            }
        }
    }
}
```

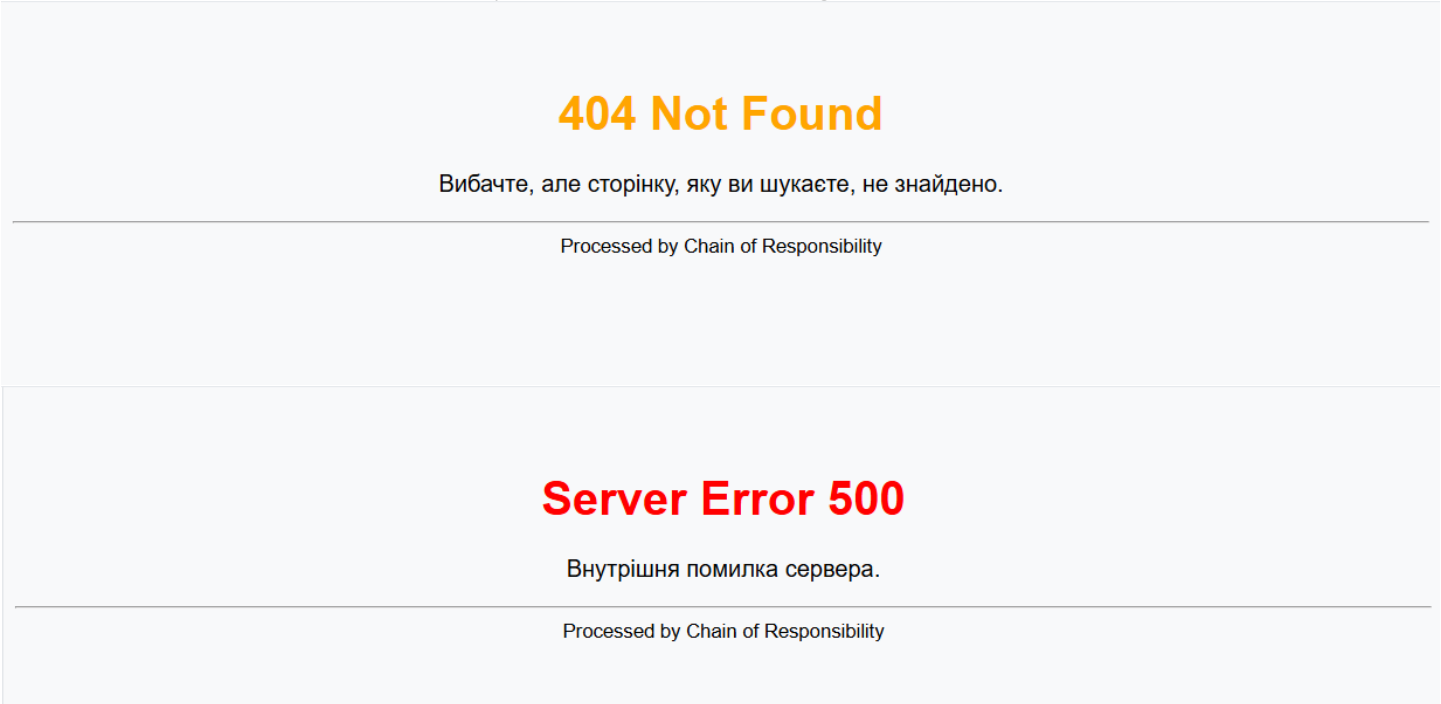
Рисунок 3 та 4 – Handlers.cs

```
using Microsoft.Web.WebView2.Wpf;

namespace WebBrowser.Coursework.Patterns
{
    6 references
    public class RequestContext
    {
        3 references
        public string Url { get; set; }
        13 references
        public int StatusCode { get; set; }
        4 references
        public WebView2 Browser { get; set; } // Посилання на компонент браузера для керування ним

        // Допоміжний метод для відображення HTML (для сторінок помилок)
        3 references
        public void RenderHtml(string title, string message, string color)
        {
            string html = $@"
                <html>
                <body style='font-family: sans-serif; text-align: center; padding-top: 50px; background-color: #f8f9fa;'>
                    <h1 style='color: {color};'>{title}</h1>
                    <p>{message}</p>
                    <hr>
                    <small>Processed by Chain of Responsibility</small>
                </body>
                </html>";

            Browser.NavigateToString(html);
        }
    }
}
```

Рисунок 4 – RequestContext.cs

```
private async void NavigateToUrl()
{
    string url = AddressBar.Text;
    if (string.IsNullOrWhiteSpace(url)) return;

    if (!url.StartsWith("http")) url = "https://" + url;

    var hRedirect = new RedirectHandler();
    var hClientError = new ClientErrorHandler();
    var hServerError = new ServerErrorHandler();
    var hSuccess = new SuccessHandler();

    hRedirect.SetNext(hClientError).SetNext(hServerError).SetNext(hSuccess);

    int statusCode = 0;
    try
    {
        using (var client = new HttpClient())
        {
            var request = new HttpRequestMessage(HttpMethod.Head, url);
            var response = await client.SendAsync(request);
            statusCode = (int)response.StatusCode;
        }
    }
    catch (HttpRequestException)
    {
        MessageBox.Show("Не вдалося з'єднатися з сервером.", "Мережева помилка");
        return;
    }

    var context = new RequestContext
    {
        Url = url,
        StatusCode = statusCode,
        Browser = BrowserView
    };

    hRedirect.Handle(context);
}
```

Рисунок 5 – Метод NavigateToUrl



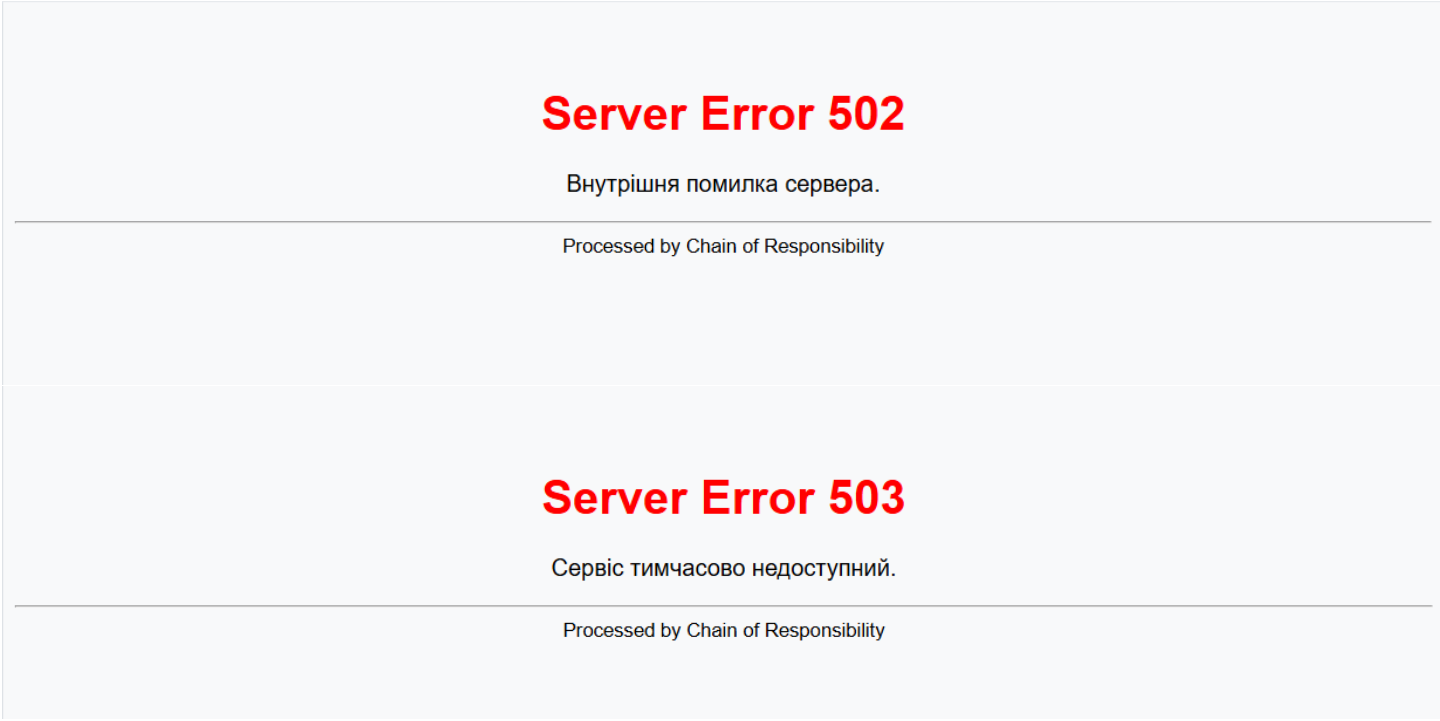


Рисунок 5,6,7,8 – Результати виконання

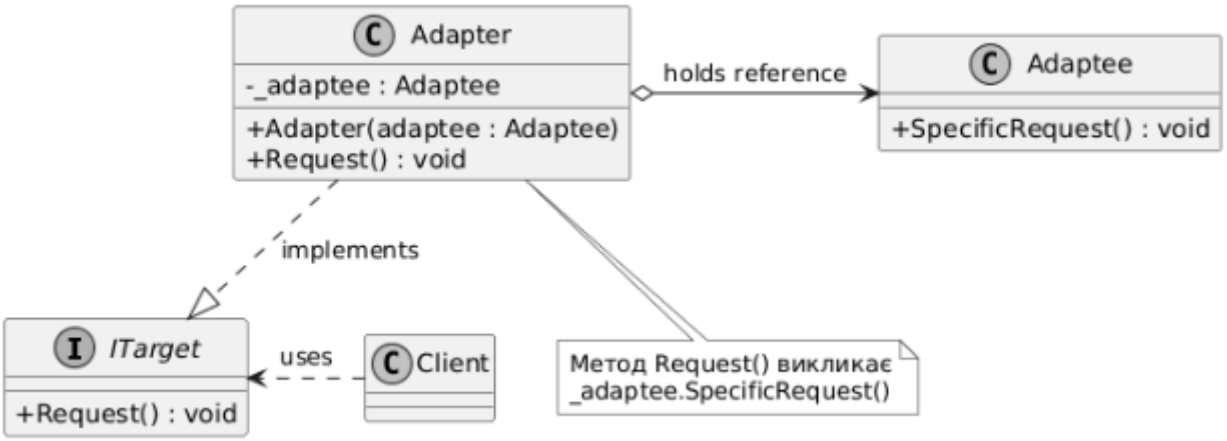
4. Висновок

У ході виконання лабораторної роботи було розглянуто структурні та поведінкові шаблони проєктування, зокрема Adapter, Builder, Command та Chain of Responsibility. Отримані знання дали можливість зрозуміти роль патернів у побудові гнучких програмних систем зі слабкою зв'язністю компонентів.

На прикладі настільного застосунку «Web-browser» було реалізовано шаблон Chain of Responsibility (Ланцюжок відповідальності) для обробки HTTP-відповідей сервера. Було створено абстракцію обробника та конкретні класи (RedirectHandler, ClientErrorHandler, ServerErrorHandler), кожен з яких відповідає за свій тип статус-кодів. Це дало змогу відокремити клієнта (вікно браузера) від складної логіки прийняття рішень щодо помилок завантаження. Такий підхід робить систему гнучкою, оскільки дозволяє легко додавати нові проміжні перевірки або змінювати порядок обробки запитів без втручання в основний код навігації.

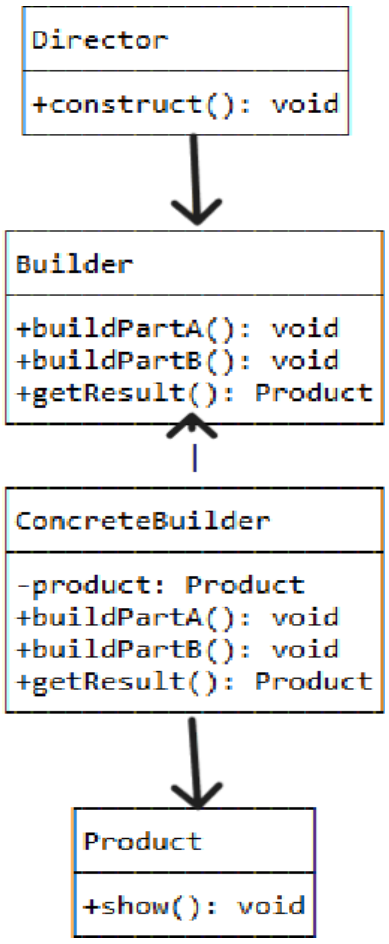
5. Контрольні питання:

1) Яке призначення шаблону «Адаптер»? Адаптує інтерфейс одного об’єкта до іншого, дозволяючи несумісним компонентам працювати разом (наприклад, уніфікація бібліотек принтерів).
2) Нарисуйте структуру шаблону «Адаптер».



3) Які класи входять в шаблон «Адаптер», та яка між ними взаємодія?

Client, Target, Adapter, Adaptee. Client працює з Target через адаптований інтерфейс, Adapter перенаправляє виклики до Adaptee.

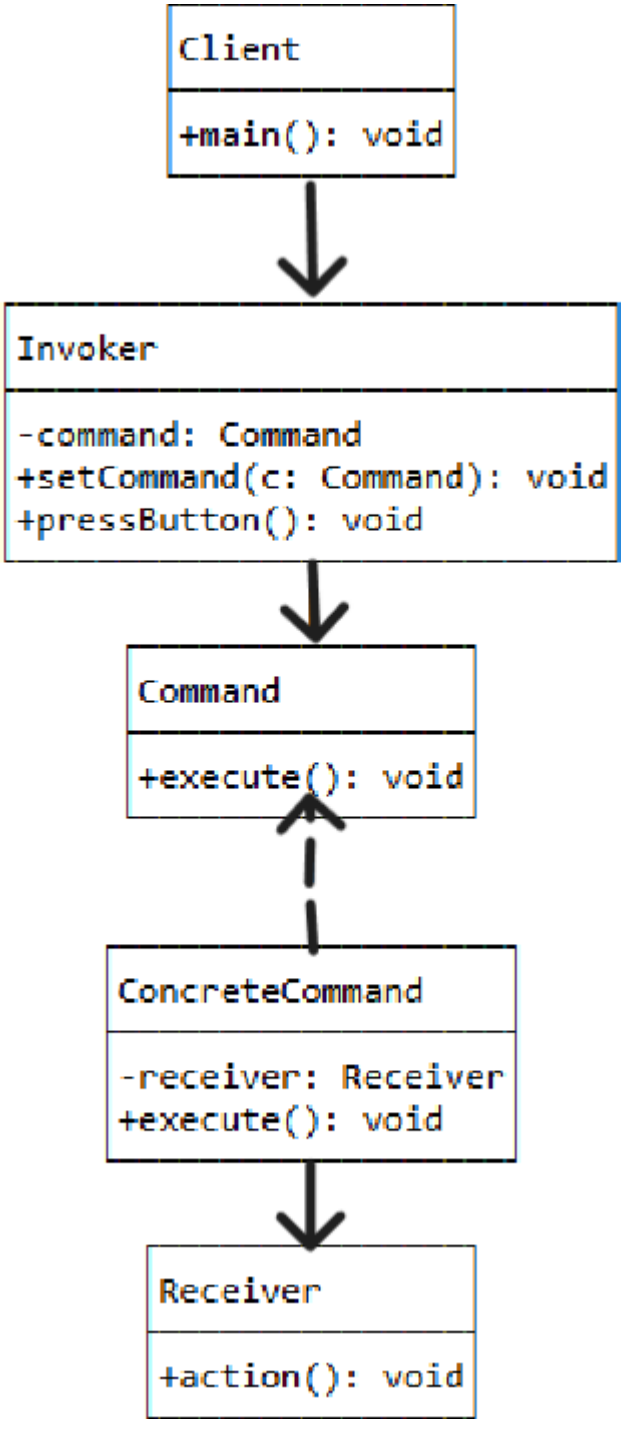


4) Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів? Об’єктний адаптер використовує композицію, класовим — успадкування.
5) Яке призначення шаблону «Будівельник»? Відокремлює процес створення об’єкта від його представлення, придатний для складних або багатоформних конструкцій.
6) Нарисуйте структуру шаблону «Будівельник».

7) Які класи входять в шаблон «Будівельник», та яка між ними взаємодія?

Director, Builder, ConcreteBuilder, Product. Director керує будівництвом, Builder визначає інтерфейс, ConcreteBuilder реалізує його, Product — результат.

8) У яких випадках варто застосовувати шаблон «Будівельник»? При складному процесі створення або необхідності різних форм об’єкта.
9) Яке призначення шаблону «Команда»? Перетворює виклик методу в об’єкт для гнучкої системи команд із відміною, логуванням чи плануванням.
10) Нарисуйте структуру шаблону «Команда».



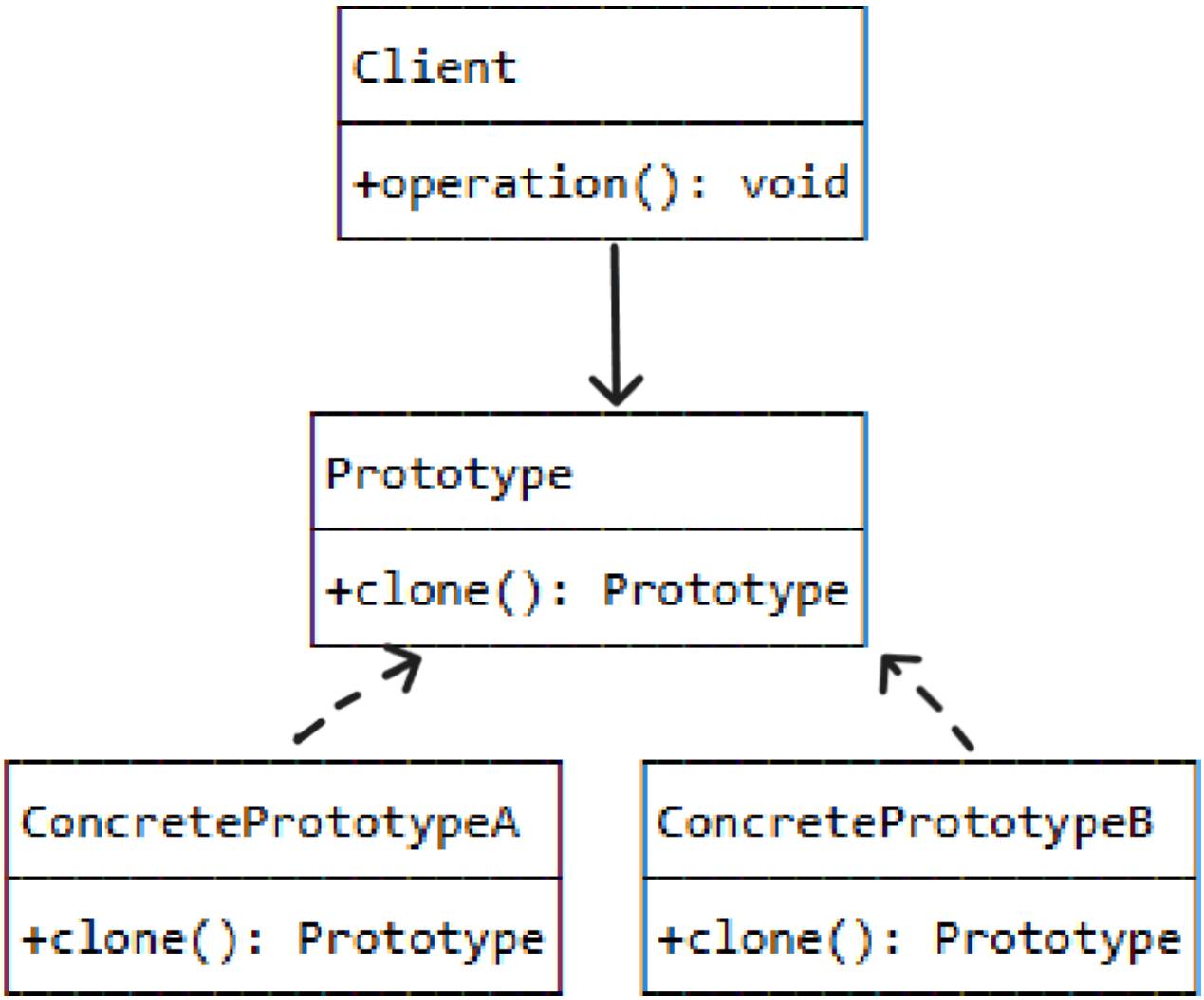
11) Які класи входять в шаблон «Команда», та яка між ними взаємодія?

Client, Invoker, Command, ConcreteCommand, Receiver. Client створює команду, Invoker виконує її, Receiver реалізує дію.

12) Розкажіть як працює шаблон «Команда». Команда інкапсулює запит як об’єкт, який передається Invoker до Receiver для виконання, підтримуючи додаткові функції (скасування, логування).

13) Яке призначення шаблону «Прототип»? Створює об’єкти клонуванням прототипу, зменшуючи ієрархію спадкування.

14) Нарисуйте структуру шаблону «Прототип».



15) Які класи входять в шаблон «Прототип», та яка між ними взаємодія?

Client, Prototype. Client клонувати об’єкт через метод Prototype.

16) Які можна привести приклади використання шаблону «Ланцюжок відповідальності»? Формування контекстного меню в UI, обробка запитів у ієрархії документів.