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: «Вступ до паттернів проектування»

Виконав
Студент групи ІА-31:
Олея М. С.

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. **Завдання:**

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.

4. **Хід роботи:**

Варіант - 8

Powershell terminal (strategy, command, abstract factory, bridge, interpreter, client-server)

Термінал для powershell повинен нагадувати типовий термінал з можливістю налаштування кольорів синтаксичних конструкцій, розміру вікна, фону вікна, а

також виконання команд powershell і виконуваних файлів, а також працювати в декількох вікнах терміналу (у вкладках або одночасно шляхом розділення вікна)

Призначення та доцільність:

У додатку "PowerShell Terminal" основною функцією є виконання текстових команд користувача (наприклад, dir, cd, echo). Без використання патерну Command (Команда) логіка обробки вводу, історія команд та безпосереднє виконання (виклик процесу ОС) були б змішані в одному класі форми. Це призвело б до жорсткої зв'язності та неможливості гнучкого керування запитами.

Паттерн Command є доцільним, оскільки він дозволяє:

- Інкапсулювати запит як об'єкт: Кожна введена команда перетворюється на об'єкт RunScriptCommand, що містить всю необхідну інформацію для її виконання.
- Відокремити ініціатора від виконавця: UI (форма) лише створює команду і передає її інвокеру, не знаючи, як саме вона буде виконана (це знає TerminalSystem).
- Зберігати історію: Завдяки тому, що команди є об'єктами, їх легко зберігати у списку (історії) для подальшого аналізу або повторного виконання.
- Підтримати розширюваність: Додавання нових типів команд (наприклад, макросів) не вимагає зміни коду інтерфейсу.

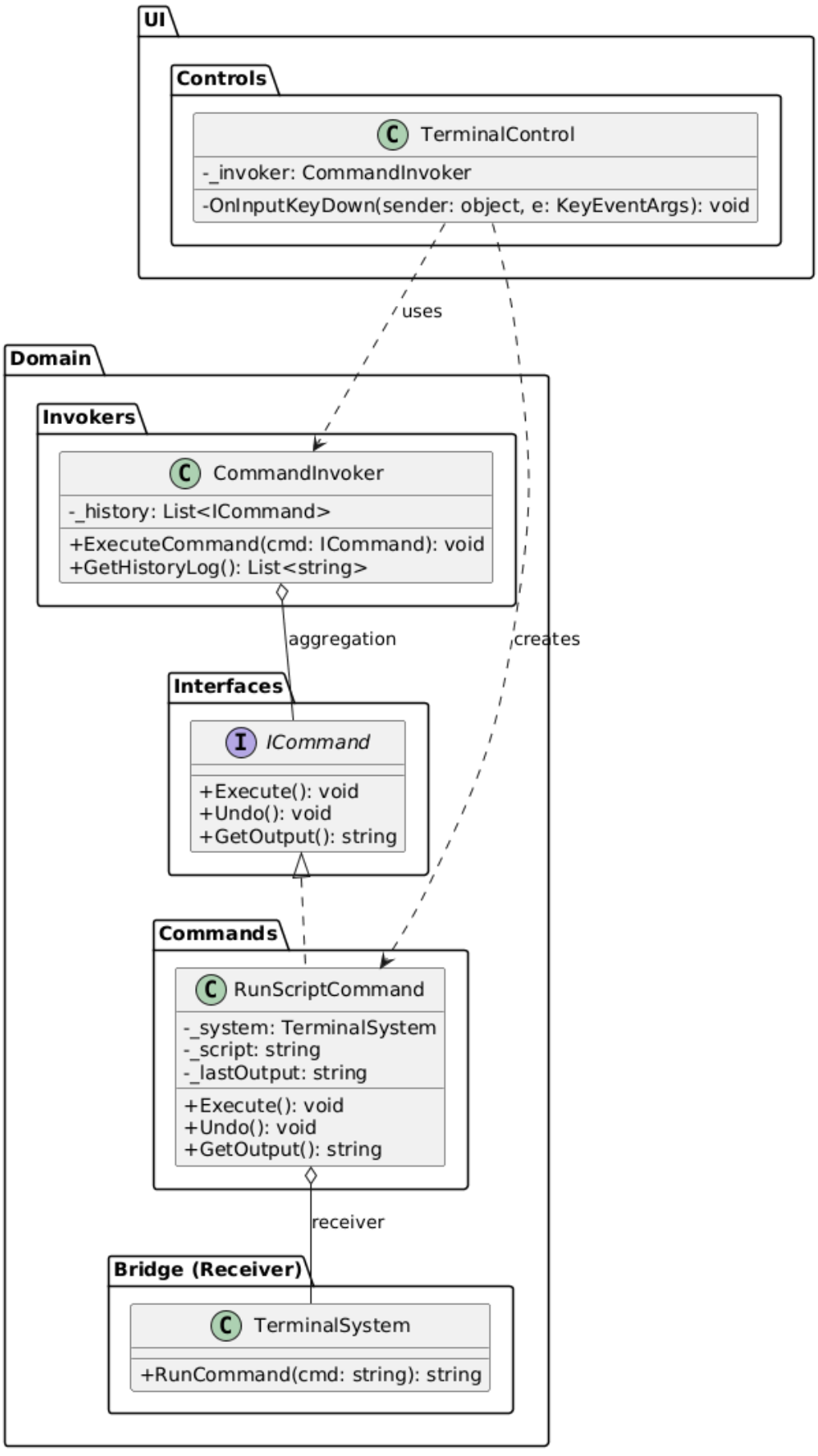


Рисунок 1 - Структура патерну Command

Діаграма (рисунок 1) відображає взаємодію компонентів патерну Command:

- ICommand (Command Interface): Оголошує інтерфейс для виконання операцій (Execute, Undo). Це контракт, якому повинні відповідати всі команди.
- RunScriptCommand (Concrete Command): Реалізує інтерфейс ICommand. Цей клас зв'язує дію (рядок скрипта _script) з одержувачем (_system). Метод Execute() викликає відповідний метод одержувача.
- TerminalSystem (Receiver): Клас, який знає, як здійснювати операції (в даному випадку — запускати процес PowerShell або CMD). Команда делегує йому виконання.
- CommandInvoker (Invoker): Ініціатор, який отримує команду від клієнта і запускає її виконання. Він також зберігає список виконаних команд (_history), що дозволяє реалізувати журнал дій.
- TerminalControl (Client): Створює конкретну команду на основі вводу користувача і передає її інвокеру для виконання.

Програмна реалізація:

а) Інтерфейс Команди (ICommand.cs)

```
namespace PowerShellTerminal.App.Domain.Interfaces
{
    public interface ICommand
    {
        void Execute();
        void Undo();
        string GetOutput();
        string GetCommandText();
    }
}
```

б) Конкретна Команда (RunScriptCommand.cs)

```
using PowerShellTerminal.App.Domain.Interfaces;
using PowerShellTerminal.App.Domain.Bridge;
using PowerShellTerminal.App.Domain.Interpreter;

namespace PowerShellTerminal.App.Domain.Commands
{
    public class RunScriptCommand : ICommand
    {
        private readonly TerminalSystem _system;
        private readonly string _script;
        private string _lastOutput;

        public RunScriptCommand(TerminalSystem system, string script)
        {
```

```
            _system = system;
            _script = script;
            _lastOutput = string.Empty;
        }

        public void Execute()
        {
            var parser = new ExpressionParser(_system);
            IExpression expressionTree = parser.Parse(_script);
            var context = new InterpreterContext(string.Empty);
            expressionTree.Interpret(context);
            _lastOutput = context.Output;
        }

        public void Undo()
        {
            _lastOutput = "Undo not supported.";
        }

        public string GetOutput()
        {
            return _lastOutput;
        }

        public string GetCommandText()
        {
            return _script;
        }
    }
}
```

в) Інвокер (CommandInvoker.cs):

```
public class CommandInvoker
    {
        private readonly List<ICommand> _history = new List<ICommand>();

        public void ExecuteCommand(ICommand command)
        {
            command.Execute();

            _history.Add(command);
        }

        public List<string> GetHistoryLog()
        {
            List<string> log = new List<string>();
            foreach (var cmd in _history)
            {
                log.Add(cmd.GetCommandText());
```

```
            }
            return log;
        }
    }
```

г) Використання у Клієнті (TerminalControl.cs):

```
private void OnInputKeyDown(object? sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Enter)
            {
                string text = _txtInput.Text;
                if (string.IsNullOrWhiteSpace(text)) return;

                var parser = new ExpressionParser(_terminalSystem);

                ICommand cmd = new RunScriptCommand(_terminalSystem, text);

                AppendText($"{_promptStr}{text}", _txtInput.ForeColor);
                _invoker.ExecuteCommand(cmd);

                string result = cmd.GetOutput();
                AppendText(result, _rtbOutput.ForeColor);

                _txtInput.Clear();
                e.SuppressKeyPress = true;
            }
        }
```

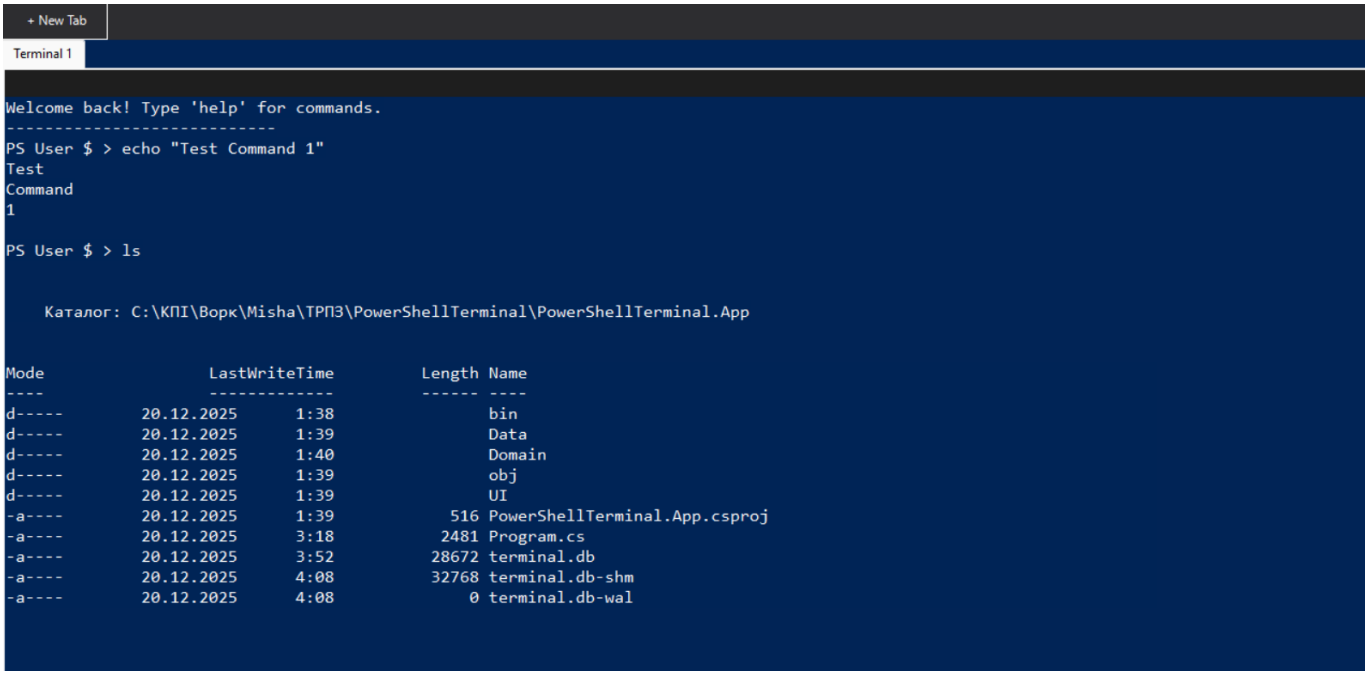


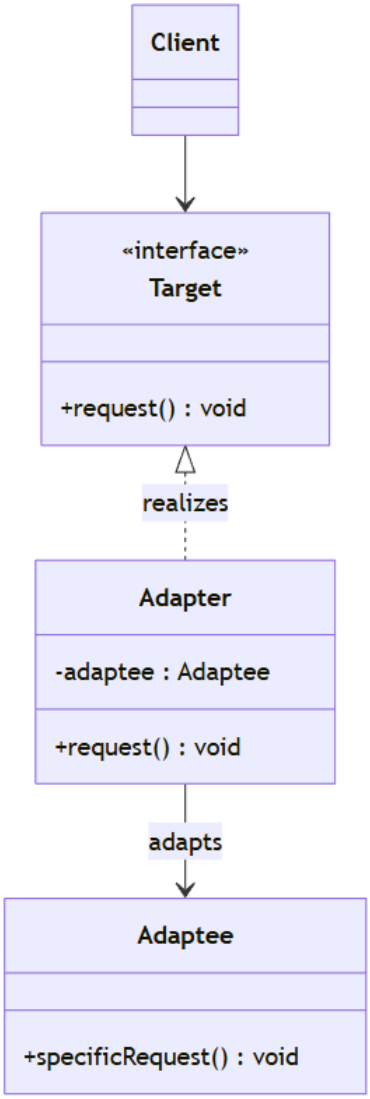
Рисунок 2 – Демонстрація роботи патерну Command

5. Висновок

У ході виконання лабораторної роботи було досліджено та імплементовано поведінковий патерн проєктування «Команда» (Command). Цей патерн дозволив інкапсулювати запити користувача до терміналу у вигляді окремих об'єктів (RunScriptCommand), що відокремило логіку ініціації запиту (інтерфейс користувача) від логіки його виконання (системний процес). Завдяки використанню класу CommandInvoker, було забезпечено централізоване керування виконанням команд та створено основу для ведення історії операцій. Реалізація патерну підвищила модульність системи, спростила додавання нових типів команд у майбутньому та забезпечила відповідність принципам SOLID.

6. Контрольні питання:

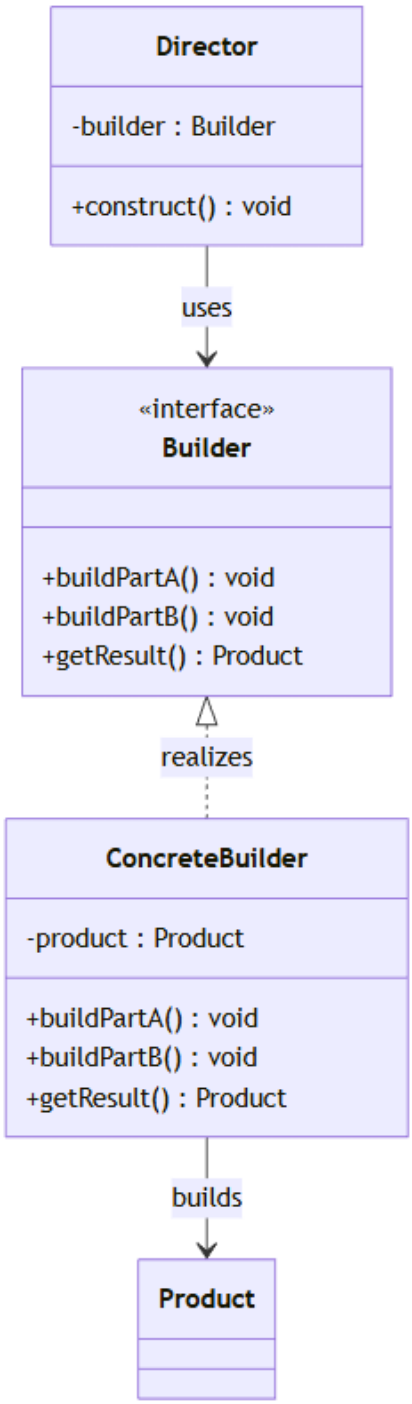
1) Яке призначення шаблону «Адаптер»? Адаптує інтерфейс одного об’єкта до іншого, дозволяючи несумісним компонентам працювати разом (наприклад, уніфікація бібліотек принтерів).
2) Нарисуйте структуру шаблону «Адаптер».



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.

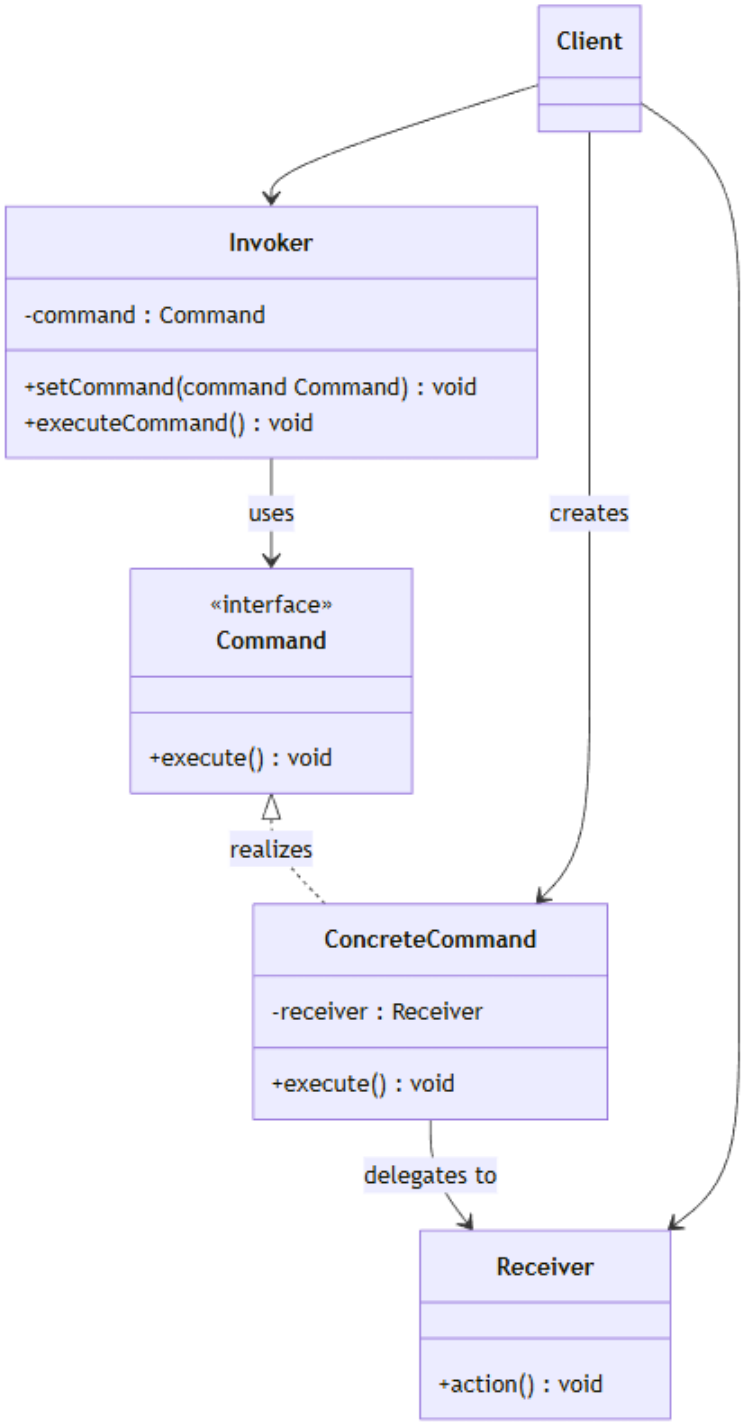
4) Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів?
   Об’єктний адаптер використовує композицію, класовим — успадкування.
5) Яке призначення шаблону «Будівельник»? Відокремлює процес створення об’єкта від його представлення, придатний для складних або багатоформних конструкцій.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



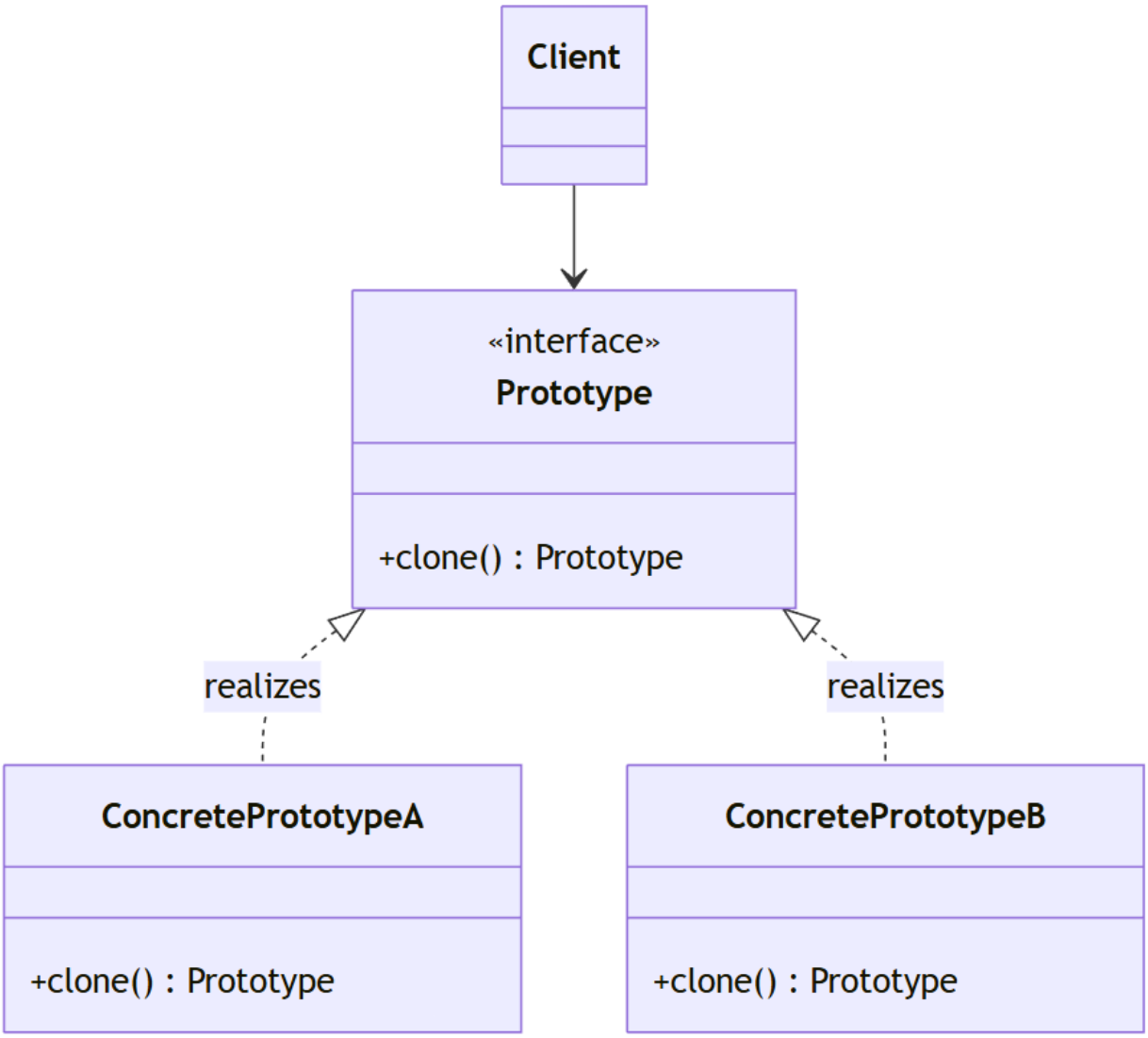
11) Які класи входять в шаблон «Команда», та яка між ними взаємодія?

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.