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 № 7

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 «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

2. Теоретичні відомості:

Mediator (Посередник): Шаблон для централізації взаємодії між об'єктами через посередника, замість прямих зв'язків. Кожен об'єкт зберігає лише посилання на медіатор. Застосовується в складних формах з великою кількістю взаємодіючих компонентів (наприклад, чекбокси, блоки, що ховаються/показуються). Переваги: Зменшення зв'язаності, простота розширення, легке тестування. Недоліки: Медіатор може стати «God Object».

Facade (Фасад): Надає спрощений уніфікований інтерфейс до складної підсистеми, приховуючи її внутрішню структуру. Використовується для роботи з різними протоколами (HTTP, TCP) або складними бібліотеками. Переваги: Інкапсуляція складності, простіший API, легке оновлення внутрішньої реалізації. Недоліки: Зменшення гнучкості.

Bridge (Міст): Розділяє абстракцію та її реалізацію, дозволяючи змінювати їх незалежно. Утворює дві ієрархії. Застосовується в графічних редакторах (фігури + драйвери: екран, принтер, bitmap). Переваги: Незалежний розвиток абстракції та реалізації, гнучкість. Недоліки: Збільшення складності.

Template Method (Шаблонний метод): Визначає скелет алгоритму в базовому класі, залишаючи окремі кроки для перевизначення в похідних. Використовується для обробки різних форматів відео (MPEG-4, MPEG-2) або компіляції веб-сторінок. Переваги: Повторне використання коду, чітка структура алгоритму. Недоліки: Жорсткий скелет, можливе порушення принципу Лісков, складність при багатьох кроках.

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 системі "PowerShell Terminal" виникла необхідність підтримки різних інтерпретаторів команд (Shell Engines) — сучасного PowerShell Core та класичного Windows CMD. Якби ми використовували звичайне успадкування, нам довелося б створювати окремі класи для кожного типу терміналу (наприклад, PowerShellTerminalWindow, CmdTerminalWindow). Якщо в майбутньому ми захочемо додати підтримку Bash або SSH, кількість класів зростатиме експоненціально, а логіка UI змішається з логікою виконання команд.

Паттерн Bridge (Міст) є доцільним, оскільки він дозволяє:

- Відокремити абстракцію від реалізації: TerminalSystem (абстракція, що керує сесією) не залежить від того, як саме виконується команда. Вона лише делегує роботу інтерфейсу IShellEngine (реалізація).
- Змінювати реалізацію під час виконання: Ми можемо перемикати двигун з PowerShell на CMD натисканням однієї кнопки, не перестворюючи вікно терміналу.
- Розширювати систему незалежно: Ми можемо додавати нові двигуни (наприклад, BashEngine), не змінюючи код графічного інтерфейсу.

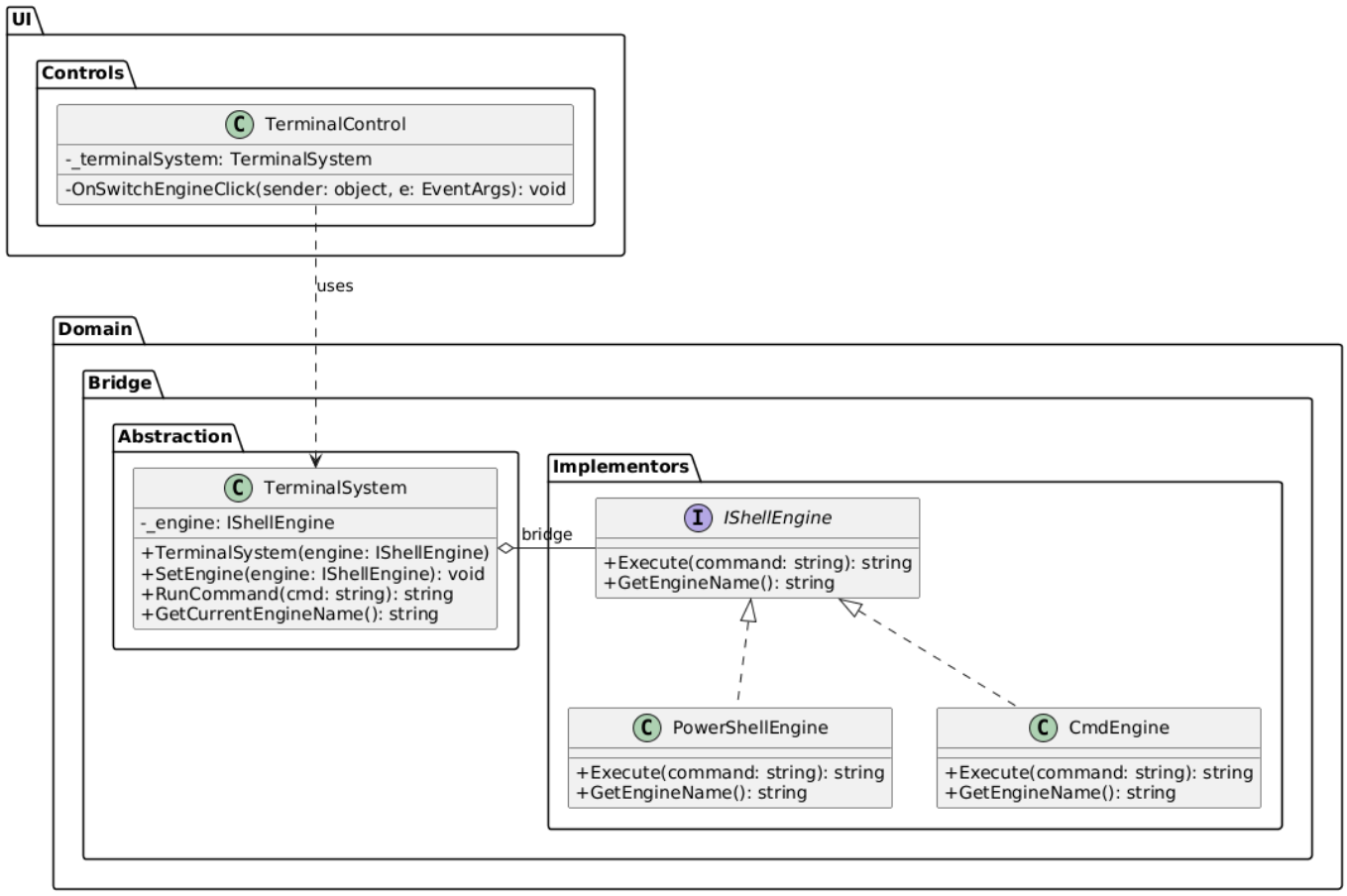


Рисунок 1 - Структура патерну bridge

Діаграма (зображена на рисунку 1) демонструє структуру патерну Bridge, який розділяє систему на дві незалежні ієрархії:

- Abstraction (TerminalSystem): Це високорівневий клас, з яким працює клієнтський код (UI). Він зберігає посилання на поточний двигун (_engine) і делегує йому виконання команд через метод RunCommand. Також він має метод SetEngine, що дозволяє динамічно замінити реалізацію.
- Implementor (IShellEngine): Спільний інтерфейс для всіх низькорівневих виконавців команд. Він оголошує метод Execute.
- Concrete Implementors (PowerShellEngine, CmdEngine): Конкретні класи, що реалізують логіку запуску процесів ОС (powershell.exe або cmd.exe). Вони приховують складні деталі (кодування, аргументи процесу) від Абстракції.
- Client (TerminalControl): Використовує TerminalSystem для виконання команд. При натисканні кнопки перемикання він просто викликає SetEngine, замінюючи об'єкт всередині моста.

Програмна реалізація:

a) Конкретна реалізація PowerShell (PowerShellEngine.cs):

```
public class PowerShellEngine : IShellEngine
    {
```

```
        public string GetEngineName() => "PowerShell Core";

        public string Execute(string command)
        {
            return RunProcess("powershell.exe", command);
        }

        private string RunProcess(string fileName, string commandText)
        {
            try
            {
                ProcessStartInfo psi = new ProcessStartInfo();
                psi.FileName = fileName;

                string psCommand = $"[Console]::OutputEncoding =
[System.Text.Encoding]::UTF8; {commandText}";

                psi.Arguments = $"-NoProfile -ExecutionPolicy Bypass -Command
\"{psCommand}\"";

                psi.RedirectStandardOutput = true;
                psi.RedirectStandardError = true;
                psi.UseShellExecute = false;
                psi.CreateNoWindow = true;

                psi.StandardOutputEncoding = Encoding.UTF8;
                psi.StandardErrorEncoding = Encoding.UTF8;

                using (Process process = Process.Start(psi))
                {
                    string output = process.StandardOutput.ReadToEnd();
                    string error = process.StandardError.ReadToEnd();
                    process.WaitForExit();

                    if (!string.IsNullOrEmpty(error)) return $"[PS Error] {error}";
                    return output;
                }
            }
            catch (Exception ex) { return $"CRITICAL: {ex.Message}"; }
        }
    }
```

б) Конкретна реалізація CMD (CmdEngine.cs):

```
public class CmdEngine : IShellEngine
    {
        public string GetEngineName() => "Windows CMD";

        public string Execute(string command)
```

```
        {
            return RunProcess("cmd.exe", command);
        }

        private string RunProcess(string fileName, string commandText)
        {
            try
            {
                ProcessStartInfo psi = new ProcessStartInfo();
                psi.FileName = fileName;

                psi.Arguments = $"/C {commandText}";

                psi.RedirectStandardOutput = true;
                psi.RedirectStandardError = true;
                psi.UseShellExecute = false;
                psi.CreateNoWindow = true;

                psi.StandardOutputEncoding = Encoding.GetEncoding(866);
                psi.StandardErrorEncoding = Encoding.GetEncoding(866);

                using (Process process = Process.Start(psi))
                {
                    string output = process.StandardOutput.ReadToEnd();
                    string error = process.StandardError.ReadToEnd();
                    process.WaitForExit();

                    if (!string.IsNullOrEmpty(error)) return $"[CMD Error] {error}";

                    return output;
                }
            }
            catch (Exception ex) { return $"CRITICAL: {ex.Message}"; }
        }
    }
```

в) Абстракція (TerminalSystem.cs): (Саме цей клас виступає мостом)

```
public class TerminalSystem
    {
        protected IShellEngine _engine;

        public TerminalSystem(IShellEngine engine)
        {
            _engine = engine;
        }

        public void SetEngine(IShellEngine engine)
        {
            _engine = engine;
```

```
        }

        public string RunCommand(string cmd)
        {
            if (string.IsNullOrWhiteSpace(cmd)) return "";

            return _engine.Execute(cmd);
        }

        public string GetCurrentEngineName()
        {
            return _engine.GetEngineName();
        }
    }
```

г) Використання у Клієнті

```
private void OnSwitchEngineClick(object? sender, EventArgs e)
        {
            string current = _terminalSystem.GetCurrentEngineName();
            if (current.Contains("PowerShell"))
                _terminalSystem.SetEngine(new CmdEngine());
            else
                _terminalSystem.SetEngine(new PowerShellEngine());

            _btnSwitchEngine.Text = $"Engine: {_terminalSystem.GetCurrentEngineName()}";
            _txtInput.Focus();
        }
```

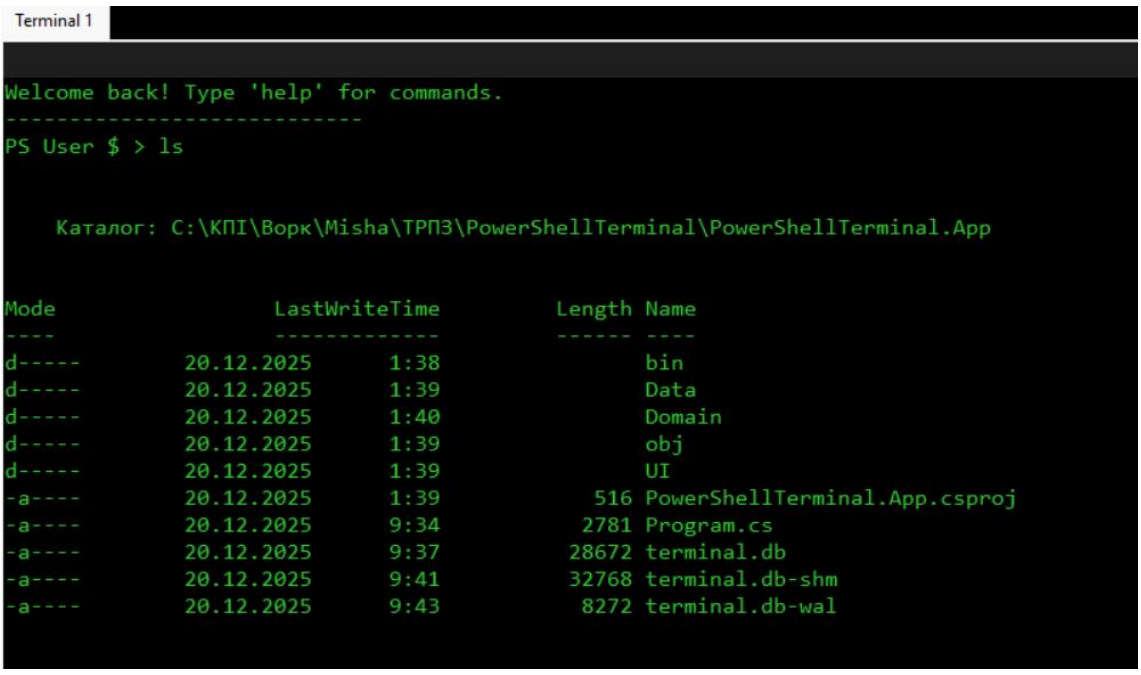


Рисунок 2 – Engine: PowerShell Core

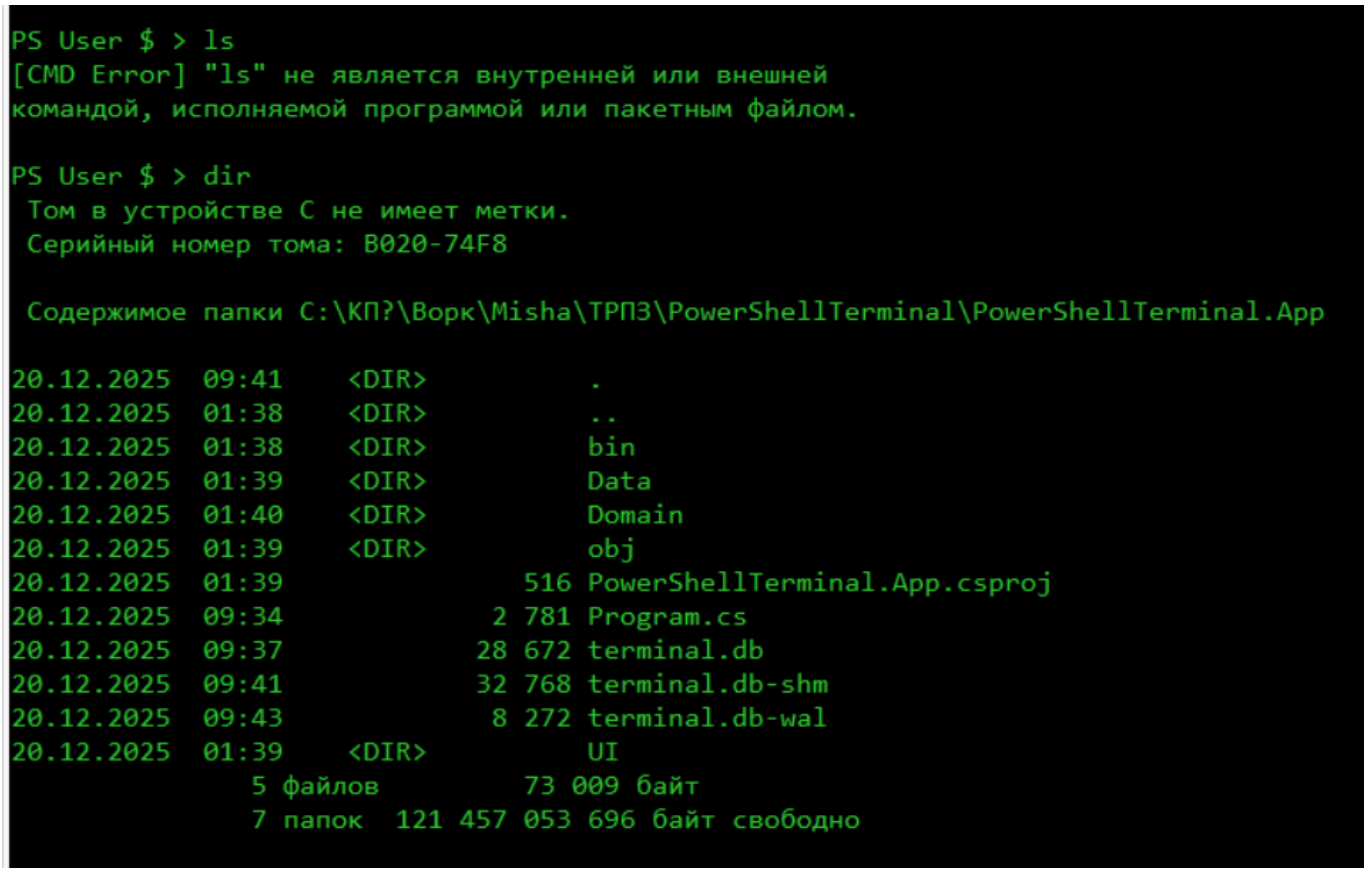


Рисунок 3 – Engine: Windows CMD

5. Висновок

У ході виконання лабораторної роботи було досліджено та реалізовано структурний патерн проєктування «Міст» (Bridge). Цей патерн дозволив відокремити абстракцію термінальної сесії (TerminalSystem) від її технічної реалізації (IShellEngine). Завдяки цьому архітектурному рішенню було досягнуто можливості динамічного перемикання між різними командними оболонками (PowerShell та Windows CMD) під час виконання програми, не змінюючи код графічного інтерфейсу та не створюючи надлишкової ієрархії класів. Реалізація патерну Bridge значно підвищила гнучкість системи, дозволяючи в майбутньому легко інтегрувати нові типи інтерпретаторів (наприклад, Bash через WSL) шляхом простого додавання нових класів-реалізацій.

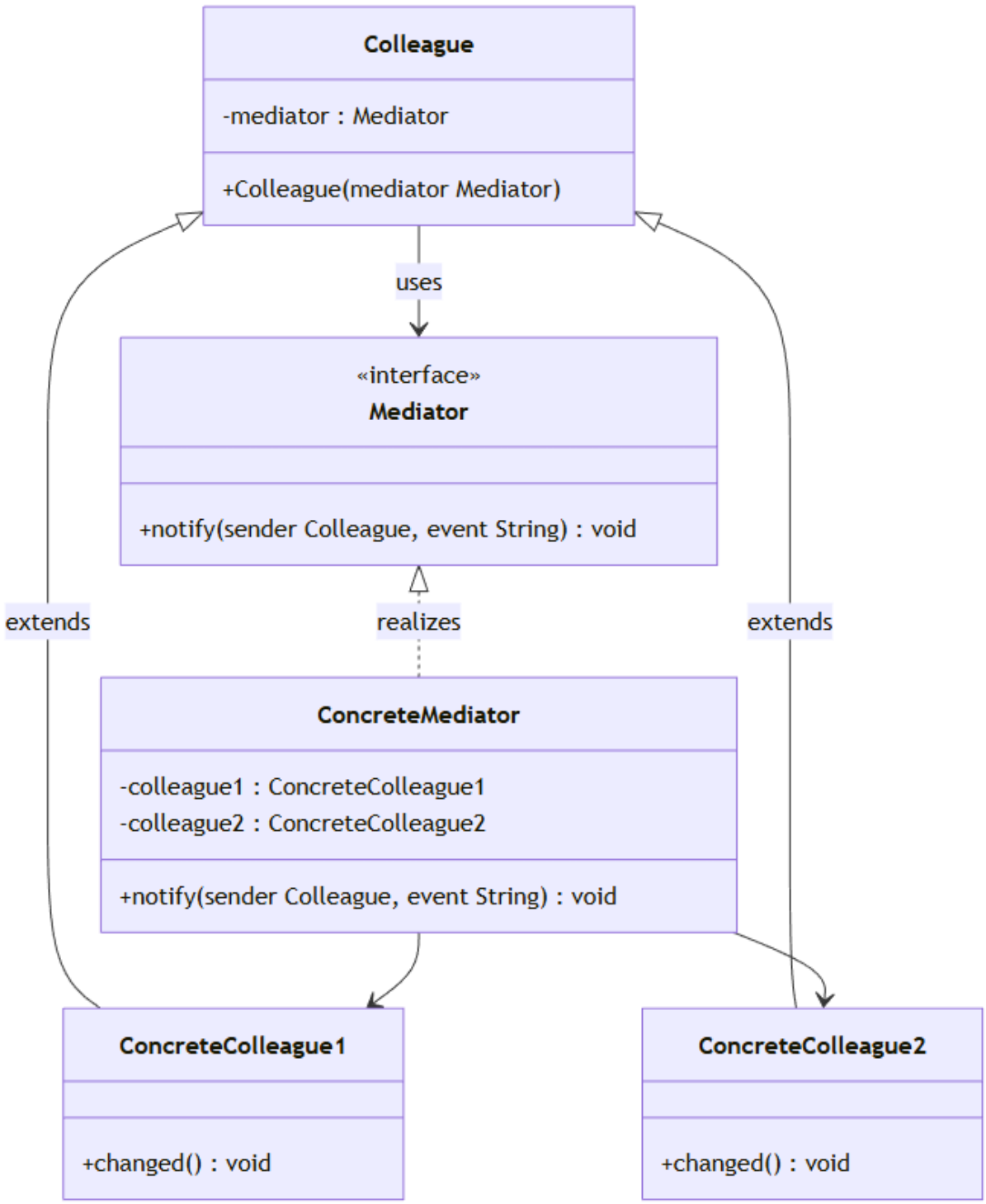
6. Контрольні питання:

1. Яке призначення шаблону «Посередник»?

Централізує взаємодію між об’єктами через єдиний об’єкт-посередник, усуваючи прямі зв’язки між ними. Зменшує зв’язність, спрощує логіку взаємодії.

2. Нарисуйте структуру шаблону «Посередник».

м



3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

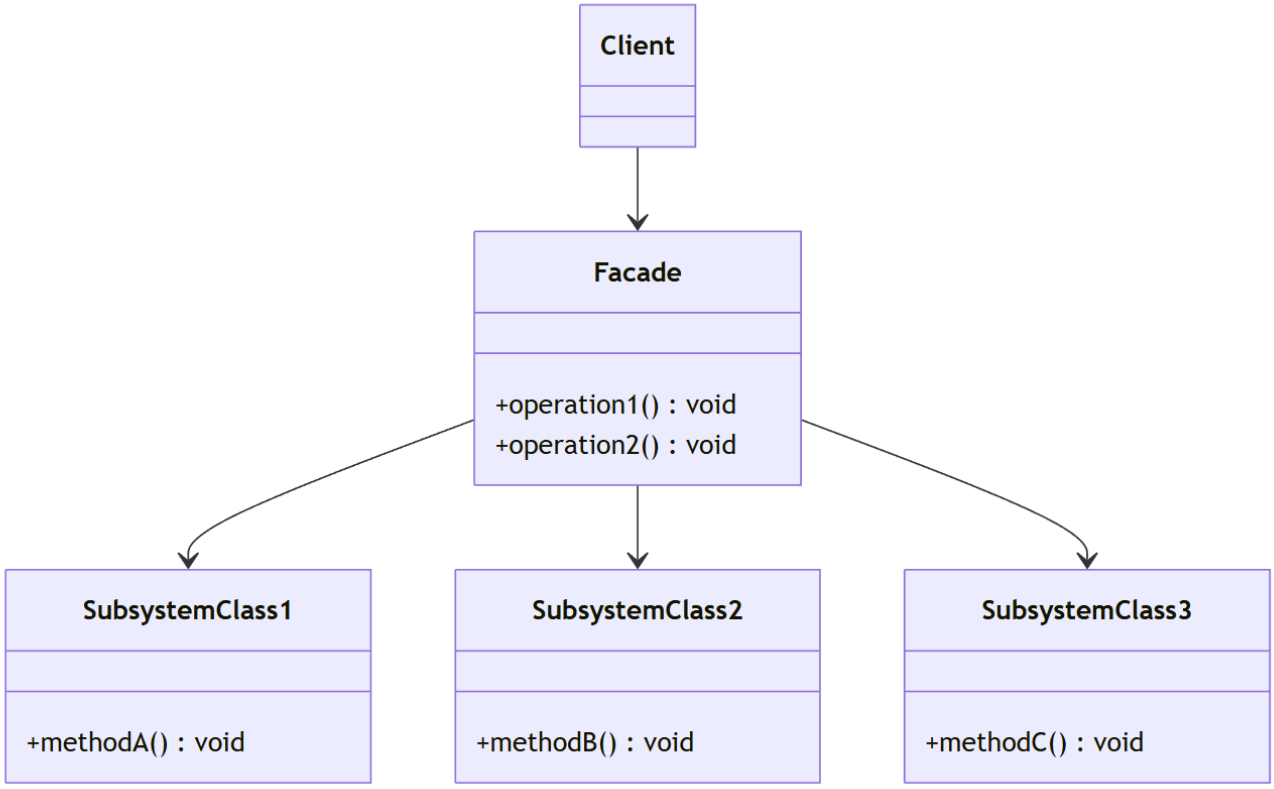
- Mediator — інтерфейс посередника
- ConcreteMediator — реалізація, координує взаємодію

- Colleague — базовий клас компонентів
- ConcreteColleague1, ConcreteColleague2 — конкретні компоненти

4. Яке призначення шаблону «Фасад»?

Надає спрощений уніфікований інтерфейс до складної підсистеми, приховуючи її внутрішню складність.

5. Нарисуйте структуру шаблону «Фасад».



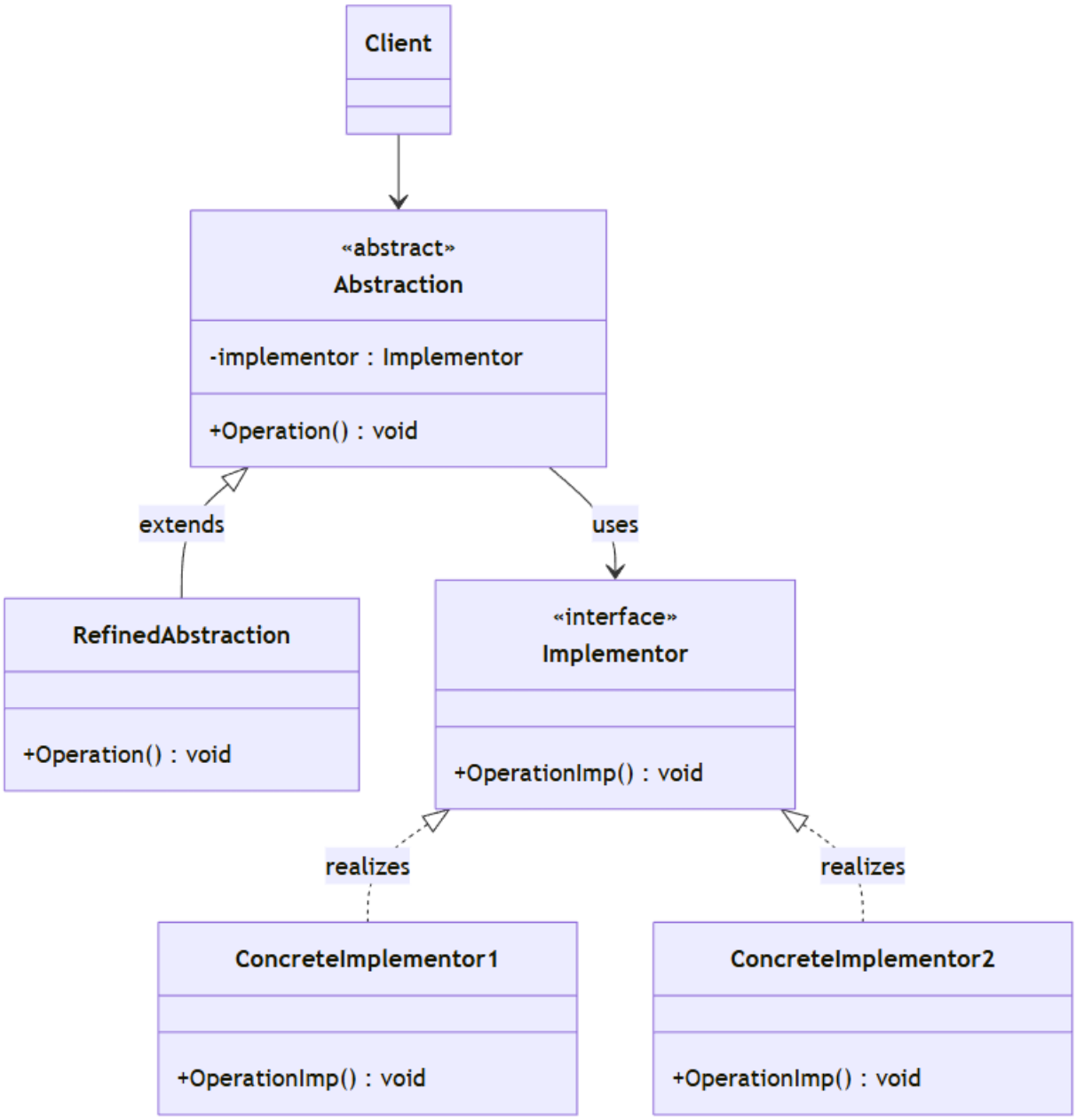
6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

- Facade — єдиний інтерфейс
- SubsystemA, SubsystemB, SubsystemC — класи підсистеми

7. Яке призначення шаблону «Міст»?

Розділяє абстракцію та її реалізацію, дозволяючи змінювати їх незалежно (дві ієрархії).

8. Нарисуйте структуру шаблону «Міст».



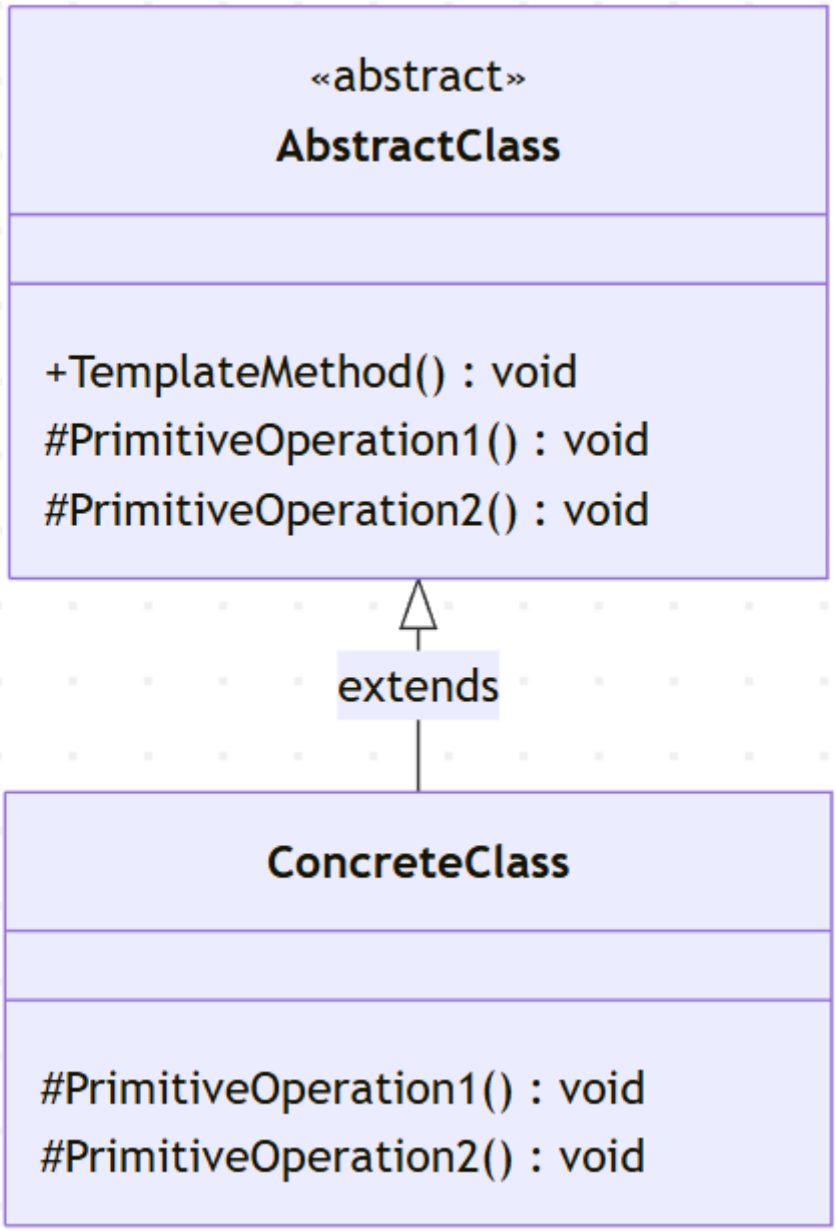
9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

- Abstraction — абстракція
- RefinedAbstraction — розширена абстракція
- Implementor — інтерфейс реалізації
- ConcreteImplementorA/B — конкретні реалізації

10. Яке призначення шаблону «Шаблонний метод»?

Визначає скелет алгоритму в базовому класі, залишаючи окремі кроки для перевизначення в похідних.

11. Нарисуйте структуру шаблону «Шаблонний метод».



12. Які класи входять в шаблон «Шаблонний метод», та яка між ними взаємодія?

- AbstractClass — містить TemplateMethod() і абстрактні методи
- ConcreteClass — реалізує абстрактні методи

13. Чим відрізняється шаблон «Шаблонний метод» від «Фабричного методу»?

Шаблонний метод» визначає скелет алгоритму, дозволяючи підкласам перевизначати окремі кроки. Фабричний метод» визначає інтерфейс створення об’єкта, залишаючи підкласам вибір конкретного класу.

14. Яку функціональність додає шаблон «Міст»?

Дозволяє незалежно розвивати абстракцію та реалізацію.