Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 7

з дисципліни «Теорія розробки прогрманого забезпечення»

Тема: «Патерни проектування.»

Варіант «Powershell terminal»

Виконав:

студент групи ІА-32
Бечке Олексій Ігорович

Перевірив:

Мягкий Михайло Юрійович

Київ 2025

Зміст

Вступ

Тема: Патерни проектування.

Мета: Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

Тема проєкту: 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).

Теоретичні відомості

Шаблон «Bridge»

Призначення патерну: Шаблон «Bridge» (міст) використовується для поділу інтерфейсу і його реалізації. Це необхідно у випадках, коли може існувати кілька різних абстракцій, над якими можна проводити дії різними способами.

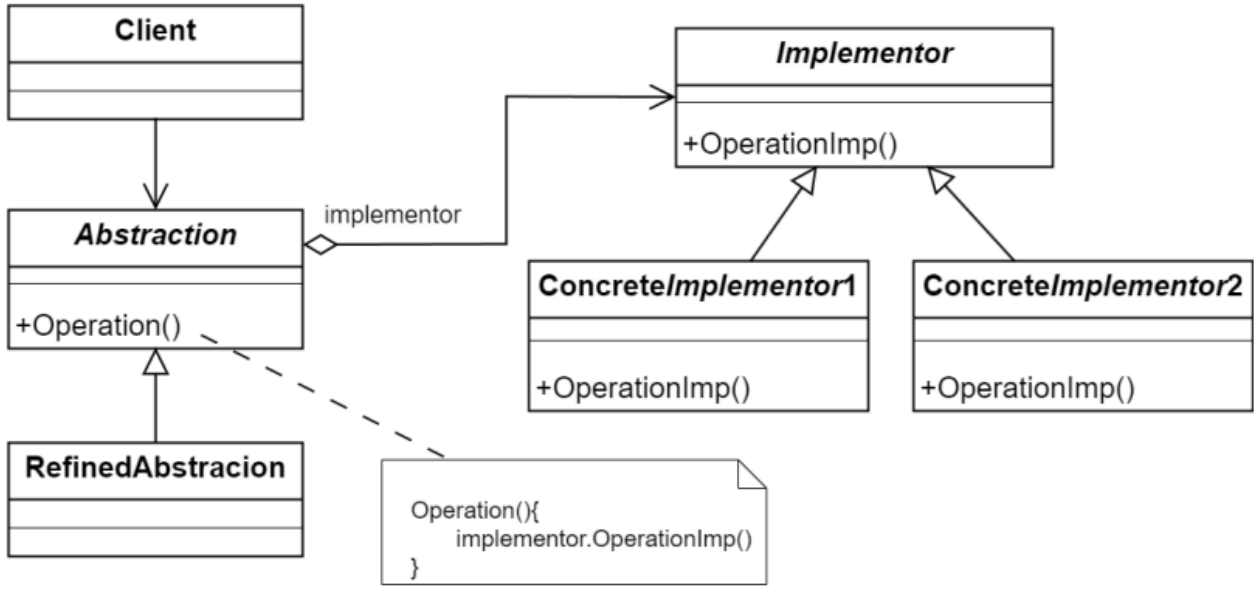


Рисунок 7.3. Структура патерну «Міст»

Проблема: Ви реалізовуєте графічний векторний редактор, який дозволяє рисувати круги, прямокутники, прямі та довільні лінії.

Ви маєте реалізувати функціонал відображення отриманого рисунку на екрані та друкувати на принтер

Спростимо ситуацію: ваш редактор дозволяє рисувати лише лінії та круги.

При такому підході, нам потрібно будувати ієрархію фігур з різною реалізацією дочірніх класів: LinePrint, LineDraw, CirclePrint, CircleDraw.

якщо добавити прямі, то добавиться ще два підкласи, і т.д. А як бути, коли нам потрібно буде ще і реалізувати збереження в bitmap форматі? добавляємо ще LineBinery, CircleBinery? При такому підході ми отримуємо дуже складну ієрархію класів.

Рішення: В даному випадку ми можемо використати патерн «Міст» (Bridge): робимо дві ієрархії – фігур (Shape) та рисування (DrawApi).

При такому підході DrawApi – це інтерфейс імплементації відображення (графічного драйвера), а Shape – інтерфейс абстракції фігур, яка має агрегацію з об'єктом DrawApi. При такому підході фігури будуть делегувати рисування об'єкту DrawApi.

Лінія, коло, та інші будуть дочірніми класами до Shape, а WindowDrawApi та PrinterDrawApi – дочірні класи до DrawApi, які представляють графічні драйвери для відображення на екрані та принтері відповідно. Якщо нам потрібно буде добавити ще і збереження в bitmap форматі, то ми добавимо ще один підклас реалізації графічного драйвера BitmapDrawApi. Таким чином ми маємо дві різні ієрархії об’єктів і вони в нас не перетинаються і не збільшуються в геометричній прогресії при додаванні нових драйверів або фігур.

Також слід відмітити, що DrawApi нічого не знає про фігури (абстракцію), а дочірні класи абстракції не залежать від реалізації графічного драйвера.

Переваги та недоліки:

+ Дозволяє змінювати ієрархії абстракції та реалізації незалежно одна від одної.

+ Розділивши абстракцію від реалізації отримуємо більшу гнучкість та простіший супровід такого коду.

- Підвищена гнучкість при використанні патерну отримується за рахунок більшої складності, введення додаткових проміжних рівнів.

Хід роботи

Завдання

- Ознайомитись з короткими теоретичними відомостями.

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

Реалізація шаблону проєктування для майбутньої системи

Ми використовуємо патерн Bridge для відокремлення високорівневої абстракції роботи терміналу від низькорівневої реалізації механізму виконання команд. Замість того, щоб жорстко прив'язувати бізнес-логіку сервісу до конкретного способу запуску процесів (наприклад, виклику powershell.exe через ProcessBuilder), ми делегуємо виконання команд окремій ієрархії реалізаторів (ExecutionEngine). Абстракція (TerminalSystem) керує процесом через цей інтерфейс, а конкретна реалізація (WindowsPowerShellEngine або TestMockEngine) може бути змінена динамічно під час роботи програми. Це дозволяє розробляти логіку терміналу незалежно від операційної системи, легко перемикатися на тестові заглушки (Mock) для безпечної перевірки, а також у майбутньому додавати підтримку інших оболонок (наприклад, Bash або SSH) без необхідності змінювати основний код системи.

```
/**
 * Implementor: Визначає інтерфейс для реалізації виконання команд.
 * Це "низькорівнева" частина моста.
 */
public interface ExecutionEngine {  8 usages  2 implementations  new *
    String execute(String command);  1 usage  2 implementations  new *
}
```

Рис. 1 – Код інтерфейсу ExecutionEngine

```java
/**
 * ConcreteImplementor A: Реальне виконання в середовищі Windows.
 */
public class WindowsPowerShellEngine implements ExecutionEngine {  3 usages  new *

    @Override  1 usage  new *
    public String execute(String commandText) {
        StringBuilder output = new StringBuilder();
        try {
            ProcessBuilder builder = new ProcessBuilder( ...command: "powershell.exe", "/c", commandText);
            builder.redirectErrorStream(true);
            Process process = builder.start();

            BufferedReader reader = new BufferedReader(
                    new InputStreamReader(process.getInputStream(), Charset.forName( charsetName: "CP866"))
            );

            String line;
            while ((line = reader.readLine()) != null) {
                output.append(line).append("\n");
            }

            process.waitFor();
        } catch (Exception e) {
            return "Error executing command in Windows Engine: " + e.getMessage();
        }
        return output.toString();
    }
}
```

Рис. 2 – Код класу WindowsPowerShellEngine

```java
/**
 * ConcreteImplementor B: Тестова реалізація або імітація Linux середовища.
 * Дозволяє перевірити роботу системи без реального виклику PowerShell.
 */
public class TestMockEngine implements ExecutionEngine {  2 usages  new *

    @Override  1 usage  new *
    public String execute(String command) {
        return "[MOCK ENGINE] Command '" + command + "' executed successfully.\n" +
                "System status: OK\n" +
                "Time: " + java.time.LocalTime.now();
    }
}
```

Рис. 3 – Код класу TestMockEngine

```java
/**
 * Abstraction: Визначає інтерфейс "високого рівня" і зберігає посилання на Implementor.
 */
public abstract class TerminalSystem {  3 usages  1 inheritor  new *

    protected ExecutionEngine engine;  3 usages

    public TerminalSystem(ExecutionEngine engine) {  1 usage  new *
        this.engine = engine;
    }

    public void setEngine(ExecutionEngine engine) {  2 usages  new *
        this.engine = engine;
    }

    public abstract String run(String command);  1 usage  1 implementation  new *
}
```

Рис. 4 – Код класу TerminalSystem

```java
public class DarkButton implements WebButton {  1 usage
    @Override  1 usage  Oleksiy Bechke *
    public String renderHtml() {
        return "<button type='submit' " +
                "style='background-color: #333; " +
                "color: white; " +
                "border: 1px solid #555; " +
                "padding: 5px 10px; " +
                "cursor: pointer;'>Execute</button>";
    }
}
```

Рис. 5 – Код класу WebTerminalSystem

```java
/**
 * RefinedAbstraction: Розширює інтерфейс абстракції.
 * Може додавати додаткову логіку (логування, перевірки) перед викликом двигуна.
 */
public class WebTerminalSystem extends TerminalSystem {  2 usages  new *

    public WebTerminalSystem(ExecutionEngine engine) {  1 usage  new *
        super(engine);
    }

    @Override  1 usage  new *
    public String run(String command) {
        // Тут може бути специфічна логіка для Веб-терміналу (наприклад, валідація)
        if (command == null || command.trim().isEmpty()) {
            return "";
        }

        // Делегування виконання конкретному двигуну через міст
        return engine.execute(command);
    }
}
```

Зображення структури шаблону

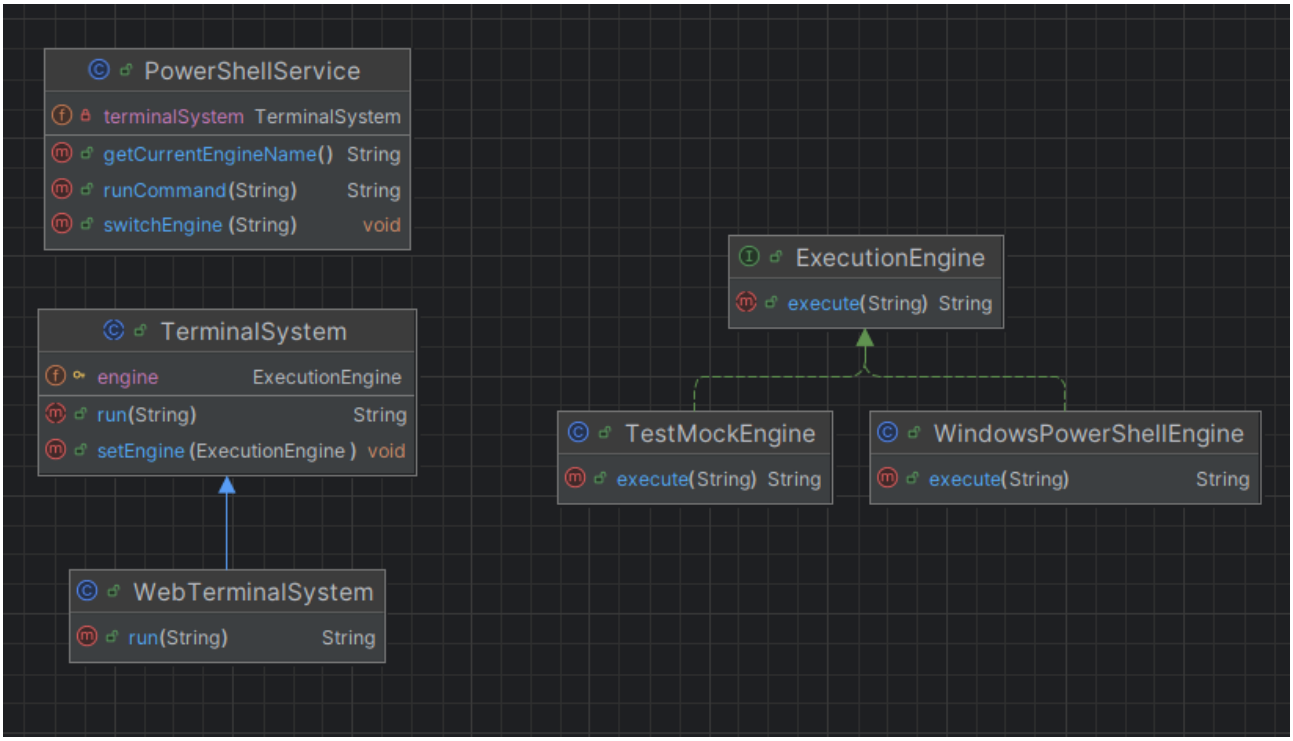


Рис. 10 – Структура шаблону Bridge

Структура реалізації:

- ExecutionEngine (Implementor) – інтерфейс, що оголошує методи для низькорівневого виконання команд (метод execute()). Він слугує спільним контрактом для всіх можливих механізмів запуску (Windows CMD, PowerShell, Linux Bash, SSH тощо) без прив'язки до конкретної логіки терміналу.
- WindowsPowerShellEngine / TestMockEngine (Concrete Implementors) – конкретні реалізації виконавця, які містять специфічний код для взаємодії з операційною системою. WindowsPowerShellEngine використовує ProcessBuilder для реального запуску процесів у середовищі Windows, тоді як TestMockEngine імітує виконання для безпечного тестування логіки без побічних ефектів.
- TerminalSystem (Abstraction) – абстрактний клас, який визначає інтерфейс високого рівня для керування терміналом та зберігає посилання на об'єкт-виконавець (ExecutionEngine). Він делегує фактичне виконання команди реалізатору, але може містити загальну логіку керування сесією.
- WebTerminalSystem (Refined Abstraction) – уточнена абстракція, що розширює базовий клас TerminalSystem. Вона адаптує роботу терміналу під специфіку веб-середовища (наприклад, може додавати валідацію вводу або спеціальну обробку помилок) перед тим, як передати команду виконавцю.
- PowerShellService (Client) – клієнтський клас, який використовує абстракцію TerminalSystem для виконання бізнес-задач. Він відповідає за конфігурацію моста (вибір конкретного ExecutionEngine), але взаємодіє з системою виключно через інтерфейс абстракції, не вдаючись у подробиці запуску процесів.

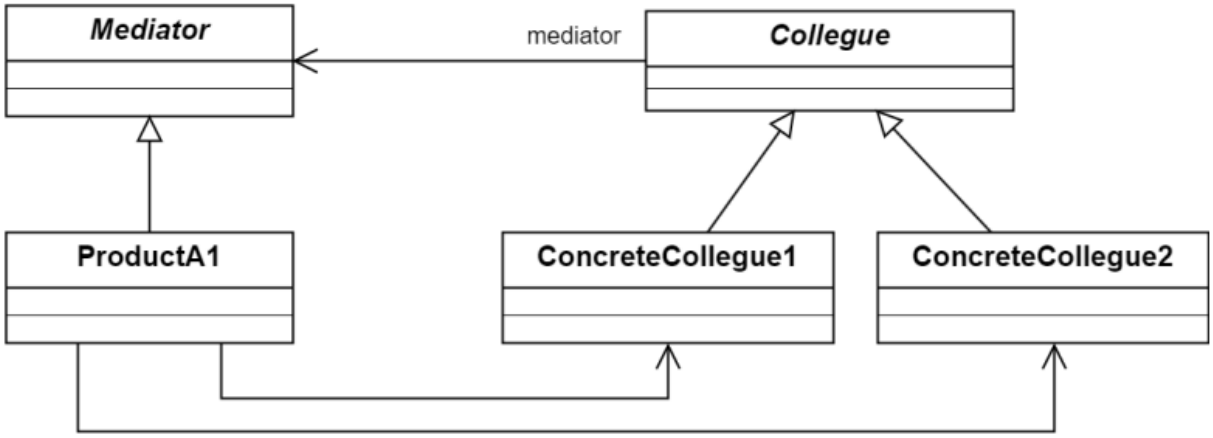
Такий підхід дозволяє відокремити високорівневу логіку роботи терміналу (абстракцію) від низькорівневої реалізації механізму виконання команд (імплементації). Це забезпечує незалежний розвиток обох частин системи: можна легко додавати підтримку нових операційних систем або протоколів (нові Realizations) без зміни коду самого терміналу, а також модифікувати логіку терміналу (нова Abstraction), не чіпаючи перевірений код запуску процесів.

Питання до лабораторної роботи

1. Яке призначення шаблону «Посередник»?

Шаблон «Mediator» (посередник) використовується для визначення взаємодії об'єктів за допомогою іншого об'єкта (замість зберігання посилань один на одного). Даний шаблон схожий на шаблон «команда», проте в даному випадку замість зберігання даних про конкретну дію, зберігаються дані про взаємодії між компонентами.

2. Нарисуйте структуру шаблону «Посередник».



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

Mediator (Посередник): Інтерфейс або абстрактний клас, що визначає методи для взаємодії з об'єктами-колегами.

ConcreteMediator (Конкретний посередник): Реалізує інтерфейс посередника та координує взаємодію між об'єктами-колегами. Він знає про всі конкретні колеги та керує їхньою співпрацею.

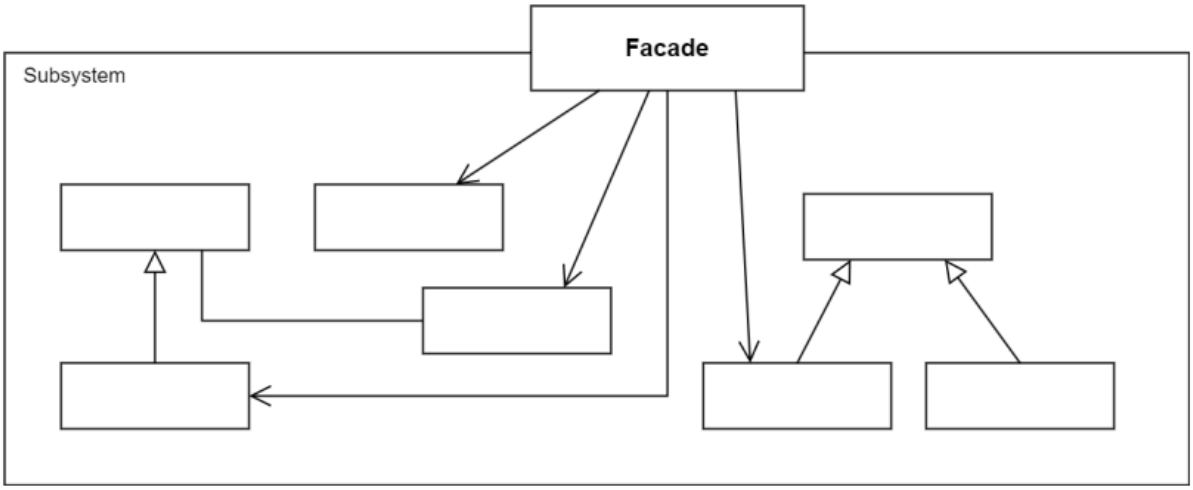
Colleague (Колега): Абстрактний клас або інтерфейс для об'єктів, що взаємодіють. Зазвичай, кожен об'єкт-колега зберігає посилання на об'єкт «медіатор».

ConcreteColleague (Конкретний колега): Реалізації класів колег (наприклад, ProductA1, ConcreteColleague1, ConcreteColleague2). Вони спілкуються з іншими колегами виключно через посередника.

4. Яке призначення шаблону «Фасад»?

Шаблон «Facade» (фасад) передбачає створення єдиного уніфікованого способу доступу до підсистеми без розкриття внутрішніх деталей підсистеми. Оскільки підсистема може складатися з безлічі класів, а кількість її функцій – не більше десяти, то щоб уникнути створення «спагетікоду» (коли все тісно пов'язано між собою) виділяють один загальний інтерфейс доступу, здатний правильним чином звертатися до внутрішніх деталей.

5. Нарисуйте структуру шаблону «Фасад».



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

Facade (Фасад): Цей клас надає спрощений та уніфікований інтерфейс для доступу до функціональності складної підсистеми. Він знає, яким класам підсистеми потрібно переадресувати запит клієнта, і делегує їм виконання роботи.

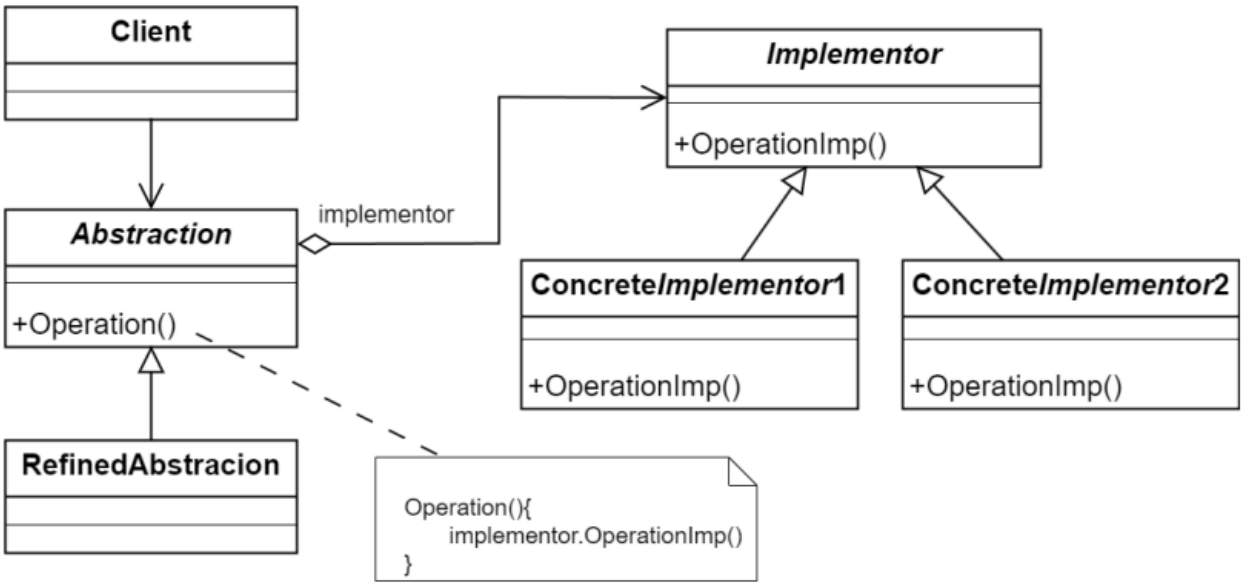
Subsystem Classes (Класи підсистеми): Це набір класів, які реалізують складну функціональність системи. Вони виконують конкретну роботу, призначену фасадом, але нічого не знають про існування фасаду і не зберігають посилань на нього.

Client (Клієнт): Взаємодіє з підсистемою виключно через інтерфейс, який надає Фасад, не звертаючись до класів підсистеми напряму.

7. Яке призначення шаблону «Міст»?

Шаблон «Bridge» (міст) використовується для поділу інтерфейсу і його реалізації. Це необхідно у випадках, коли може існувати кілька різних абстракцій, над якими можна проводити дії різними способами

8. Нарисуйте структуру шаблону «Міст».



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

Abstraction (Абстракція): Визначає інтерфейс абстракції та зберігає посилання на об'єкт типу Implementor. Вона делегує виконання роботи цьому об'єкту.

RefinedAbstraction (Уточнена абстракція): Розширює інтерфейс, визначений класом Abstraction. Це конкретний клас, який використовує можливості реалізатора.

Implementor (Реалізатор): Визначає інтерфейс для класів реалізації. Цей інтерфейс не зобов'язаний збігатися з інтерфейсом Abstraction (зазвичай Implementor надає лише примітивні операції, а Abstraction визначає операції вищого рівня на їх основі).

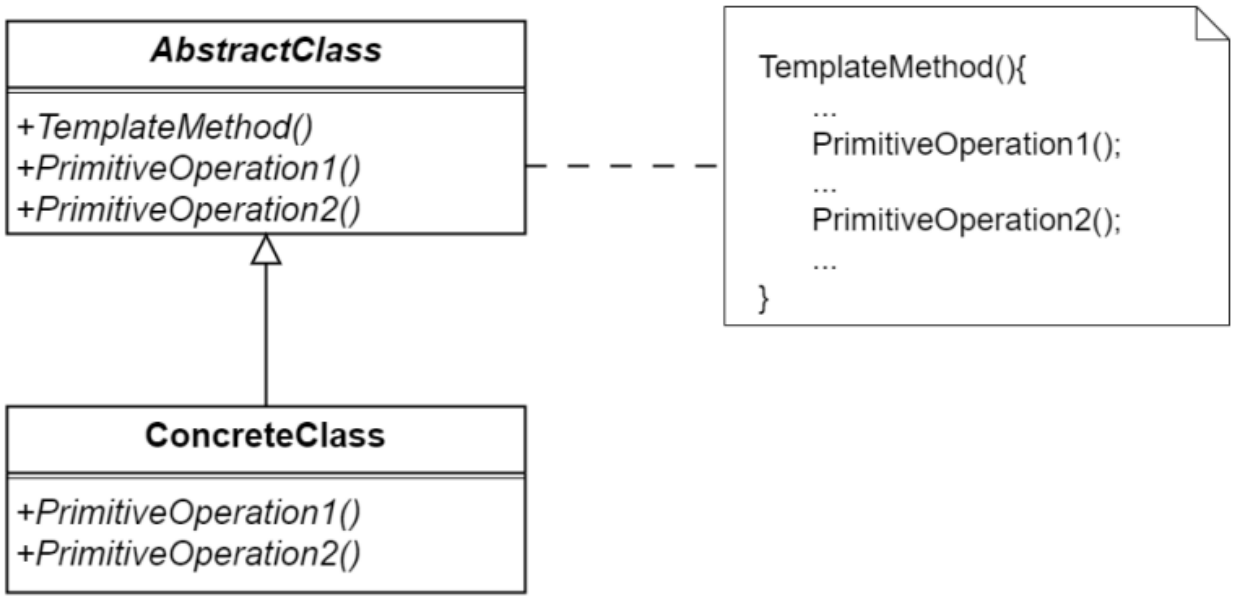
ConcreteImplementor (Конкретний реалізатор): Конкретна реалізація інтерфейсу Implementor.

Client (Клієнт): Використовує об'єкти Abstraction, не знаючи деталей реалізації.

10. Яке призначення шаблону «Шаблоний метод»?

Шаблон «Template Method» (шаблонний метод) дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації підкласам. Можна привести в приклад формування вебсторінки: необхідно додати заголовки, вміст сторінки, файли, що додаються, і нижню частину сторінки. Код для додавання вмісту сторінки може бути абстрактним і реалізовуватися в різних класах – AspNetCompiler, HtmlCompiler, PhpCompiler і т.п. Додавання всіх інших елементів виконується за допомогою вихідного абстрактного класу з алгоритмом

11. Нарисуйте структуру шаблону «Шаблоний метод».



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

AbstractClass (Абстрактний клас): Цей клас визначає абстрактні примітивні операції, які заміщуються в конкретних підкласах для реалізації кроків алгоритму. Також він реалізує сам шаблонний метод (TemplateMethod), що визначає скелет алгоритму. Шаблонний метод викликає примітивні операції, а також може містити операції, визначені в самому класі AbstractClass або в інших об'єктах .

ConcreteClass (Конкретний клас): Цей клас реалізує примітивні операції (PrimitiveOperation1, PrimitiveOperation2), виконуючи кроки алгоритму у специфічний для себе спосіб .

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

Шаблонний метод (Template Method) – це поведінковий шаблон, який використовується для покрокового визначення скелета алгоритму. Він дозволяє підкласам перевизначати певні кроки алгоритму без зміни його загальної структури. Цей шаблон не обов'язково створює нові об'єкти, а фокусується на визначенні послідовності дій.

Фабричний метод (Factory Method) – це породжувальний шаблон, який визначає інтерфейс для створення об'єктів, але дозволяє підкласам змінювати тип створюваних об'єктів.

14.Яку функціональність додає шаблон «Міст»?

Розділення інтерфейсу та реалізації: Шаблон використовується для того, щоб розділити абстракцію (інтерфейс) та її реалізацію так, щоб вони могли змінюватися незалежно одна від одної.

Підвищення гнучкості: Завдяки цьому поділу система стає більш гнучкою, а код – простішим для супроводу.

Уникнення комбінаторного вибуху класів: Шаблон дозволяє уникнути створення складної ієрархії класів, яка б виникала при спробі реалізувати всі можливі комбінації абстракцій та їх реалізацій (наприклад, різні фігури на різних графічних драйверах).

Приховування деталей: Клієнтський код працює з абстракцією і не залежить від деталей реалізації.

Висновок

Під час виконання лабораторної роботи я вивчив структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи. Реалізував шаблон «Bridge» у проєкті Power Shell термінал.