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

Лабораторна робота №7
Технології розроблення програмного забезпечення
«Патерни проектування»

Тема: Особиста бухгалтерія

Виконала:
студентка групи ІА-32
Коляда М. С.

Перевірила:
Мягкий М.Ю.

Київ 2025

Зміст

Тема: Патерни проектування.

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

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.

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.

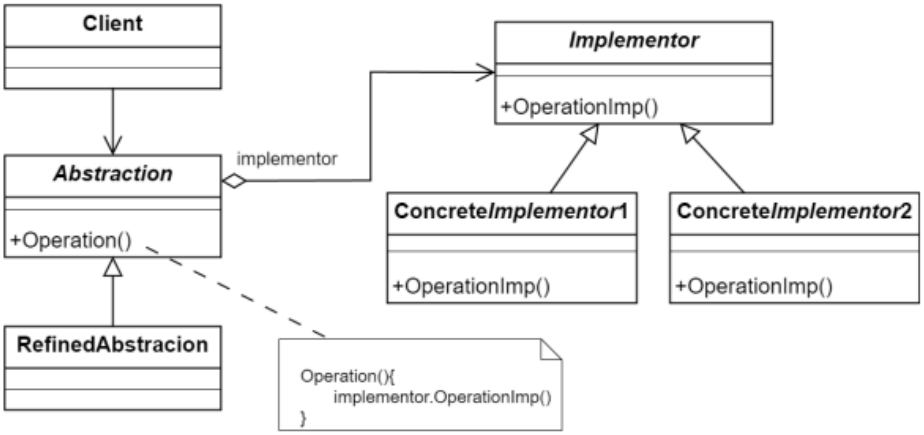


Рисунок 7.3. Структура патерну «Міст»

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.

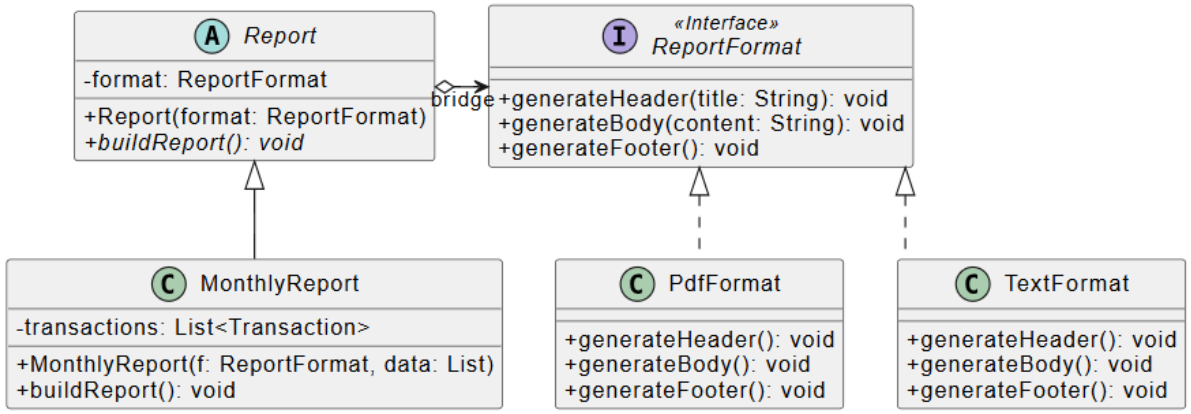
Переваги та недоліки:

+ 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

Діаграма класів:



Опис реалізації патерну «Bridge»

Реалізовано структурний патерн Bridge (Міст) для розділення абстракції (типи фінансових звітів) та її реалізації (формати виводу даних). Метою впровадження патерну була необхідність уникнути комбінаторного зростання кількості класів при розширенні системи новими типами звітів (наприклад, річний звіт) або новими форматами (наприклад, Excel).

Система дозволяє генерувати звіти незалежно від формату виводу:

1. Абстракція звітності: Визначає логіку збору даних (наприклад, звіт за конкретний місяць).
2. Реалізація форматування: Визначає спосіб відображення даних (простий текст у консолі або форматований PDF-вивід).

Механізм роботи: Абстрактний клас Report містить посилання на об'єкт інтерфейсу ReportFormat (поле format). Клас звіту (MonthlyReport)

відповідає за бізнес-логіку — які саме транзакції включити у звіт. Проте, коли справа доходить до відображення (друк заголовка, тіла, підвалу), звіт не робить це самостійно, а делегує задачу об'єкту format. Це дозволяє змінювати формат звіту "на льоту", просто передаючи інший об'єкт реалізатора.

Учасники патерну:

- ❖ Report (Abstraction / Абстракція) Базовий абстрактний клас для всіх видів звітів. Він зберігає посилання на об'єкт реалізації (ReportFormat) і визначає інтерфейс для побудови звіту (buildReport). Цей клас виступає "мостом", що з'єднує високорівневу логіку з низькорівневою реалізацією.
- ❖ MonthlyReport (Refined Abstraction / Уточнена абстракція) Конкретний клас, що розширює абстракцію. Він реалізує специфічну логіку для місячних звітів: фільтрує транзакції за датою, формує структуру документу, але для безпосереднього виводу даних використовує методи успадкованого об'єкта format.
- ❖ ReportFormat (Implementor / Реалізатор) Спільний інтерфейс для всіх форматів виводу. Він оголошує набір примітивних операцій (generateHeader, generateBody, generateFooter), які повинні бути реалізовані конкретними форматами. Інтерфейс не залежить від того, який саме звіт будується.
- ❖ PdfFormat / TextFormat (Concrete Implementors / Конкретні реалізатори) Класи, що реалізують інтерфейс ReportFormat.
    - TextFormat: Виводить звіт у вигляді простого тексту, використовуючи стандартний потік виводу.
    - PdfFormat: Імітує генерацію PDF-документу, додаючи відповідні метадані та форматування.

Зв'язки: Абстракція (Report) агрегує (містить посилання на) Реалізатора (ReportFormat). Цей зв'язок є ключовим елементом патерну ("міст").

Уточнена абстракція (MonthlyReport) використовує методи реалізатора для виконання своїх функцій.

Обґрунтування вибору шаблону «Bridge»

Для вирішення задачі генерації звітів було обрано саме шаблон Bridge, а не інші, з наступних причин:

1. Якби ми використовували звичайне успадкування, нам довелося б створити класи для кожної комбінації: MonthlyTextReport, MonthlyPdfReport, YearlyTextReport, YearlyPdfReport тощо. При додаванні нового формату (наприклад, HTML) довелося б змінювати всі класи звітів. Bridge дозволяє додавати нові формати, просто створивши новий клас, що реалізує ReportFormat, не чіпаючи ієрархію звітів.
2. Клієнтський код працює з високорівневою абстракцією (Report), не заглиблюючись у деталі того, як саме формується PDF або текст. Це знижує зв'язність коду.

Фрагменти програмного коду:

ReportFormat.java

```
package ia32.koliada.finance.bridge;

public interface ReportFormat {
   void generateHeader(String title);
   void generateBody(String content);
   void generateFooter();
}
```

PdfFormat.java

```
package ia32.koliada.finance.bridge;

public class PdfFormat implements ReportFormat {
   @Override
```

```
    public void generateHeader(String title) {
        System.out.println("[PDF] --- " + title + " ---");
    }

    @Override
    public void generateBody(String content) {
        System.out.println("[PDF] Content: " + content);
    }

    @Override
    public void generateFooter() {
        System.out.println("[PDF] --- End of Document ---");
    }
}
```

TextFormat.java

```
package ia32.koliada.finance.bridge;

public class TextFormat implements ReportFormat {
    @Override
    public void generateHeader(String title) {
        System.out.println("########## " + title + " ##########");
    }

    @Override
    public void generateBody(String content) {
        System.out.println(content);
    }

    @Override
    public void generateFooter() {
        System.out.println("####################################");
    }
}
```

Report.java

```
package ia32.koliada.finance.bridge;
```

```
public abstract class Report {
    protected ReportFormat format;

    public Report(ReportFormat format) {
        this.format = format;
    }

    public abstract void buildReport();
}
```

MonthlyReport.java

```
package ia32.koliada.finance.bridge;

import ia32.koliada.finance.entity.Transaction;
import java.util.List;

public class MonthlyReport extends Report {
    private List<Transaction> transactions;

    public MonthlyReport(ReportFormat format, List<Transaction> transactions)
{
        super(format);
        this.transactions = transactions;
    }

    @Override
    public void buildReport() {
        format.generateHeader("Місячний звіт");

        StringBuilder body = new StringBuilder();
        for (Transaction t : transactions) {
            body.append("\n - ").append(t.getDescription())
                .append(": ").append(t.getAmount()).append(" грн");
        }

        format.generateBody(body.toString());
```

```
        format.generateFooter();
    }
}
```

Вихідний код:

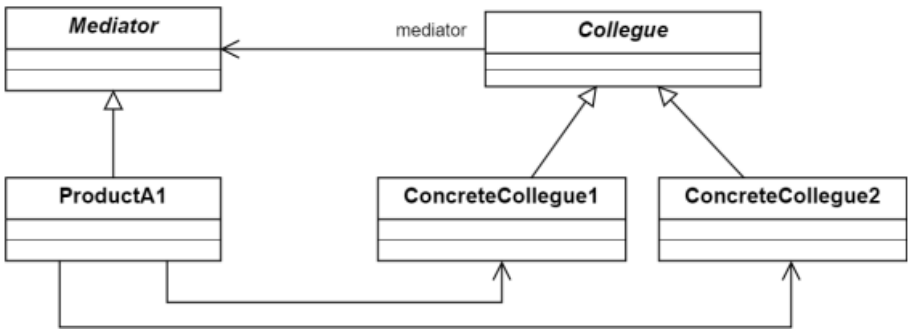
https://github.com/mariiakoliada/TRPZ/tree/master/src/ia32/koliada/finance/bridge

Питання до лабораторної роботи

1. Яке призначення шаблону «Посередник»?

Шаблон визначає об'єкт, який інкапсулює спосіб взаємодії множини об'єктів, зменшуючи зв'язність між ними та дозволяючи змінювати їхню взаємодію незалежно.

2. Нарисуйте структуру шаблону «Посередник».



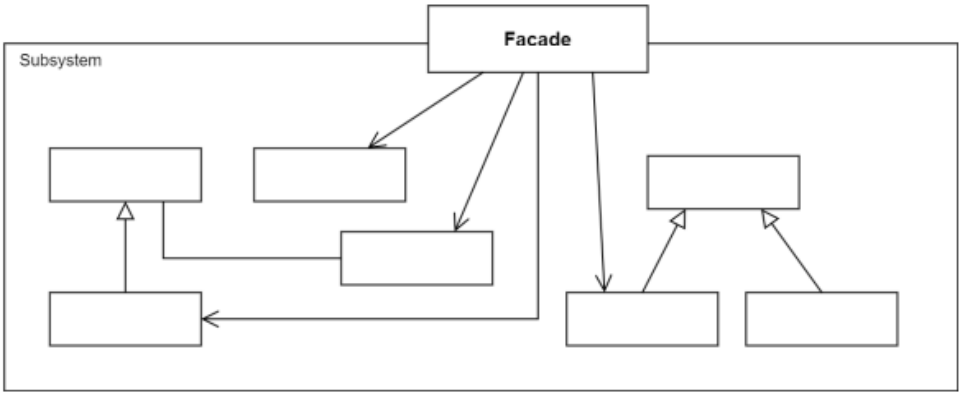
3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

До складу входять: Mediator (інтерфейс), ConcreteMediator (координатор), Colleague (класи учасників). Взаємодія: Колеги не спілкуються напряму, а відправляють сповіщення Посереднику, який приймає рішення та перенаправляє команди іншим Колегам.

4. Яке призначення шаблону «Фасад»?

Шаблон надає простий уніфікований інтерфейс до складної системи класів, бібліотеки або фреймворку, приховуючи складність системи від клієнта.

5. Нарисуйте структуру шаблону «Фасад».



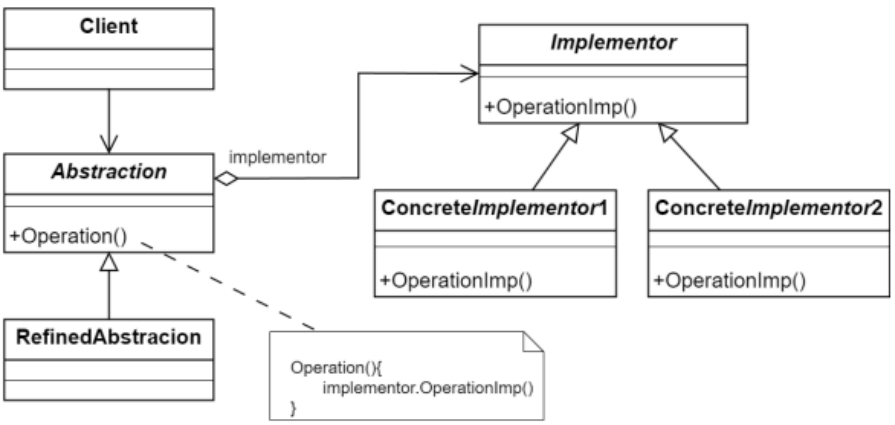
6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

До складу входять: Facade (простий інтерфейс), Subsystem Classes (складна логіка), Client. Взаємодія: Клієнт викликає методи Фасаду, а Фасад делегує виконання відповідним об'єктам підсистеми, не даючи клієнту працювати з ними напряму.

7. Яке призначення шаблону «Міст»?

Шаблон розділяє абстракцію та реалізацію так, щоб вони могли змінюватися незалежно одна від одної, запобігаючи комбінаторному зростанню кількості класів.

8. Нарисуйте структуру шаблону «Міст».



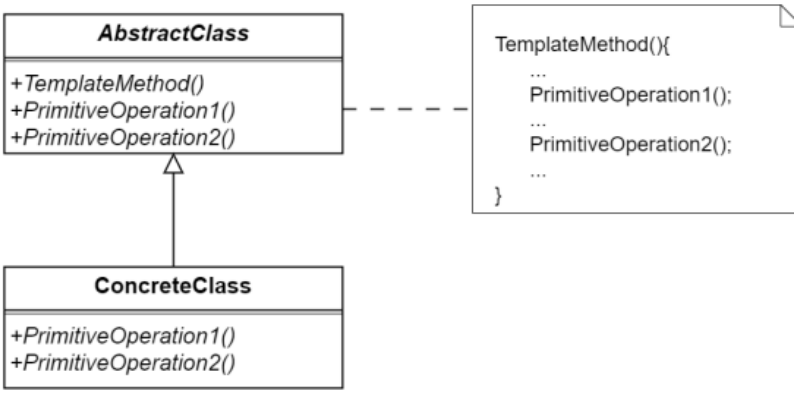
9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

До складу входять: Abstraction (інтерфейс управління), RefinedAbstraction (розширена абстракція), Implementor (інтерфейс реалізації), ConcreteImplementor (конкретна реалізація). Взаємодія: Абстракція зберігає посилання на об'єкт Реалізації і делегує йому виконання низькорівневих операцій.

10.Яке призначення шаблону «Шаблоний метод»?

Шаблон визначає скелет алгоритму в суперкласі, але дозволяє підкласам перевизначати певні кроки цього алгоритму без зміни його загальної структури.

11.Нарисуйте структуру шаблону «Шаблоний метод».



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

До складу входять: AbstractClass (оголошує кроки та шаблонний метод), ConcreteClass (реалізує абстрактні кроки). Взаємодія: Клієнт викликає публічний шаблонний метод, який послідовно виконує кроки, реалізація яких береться з конкретного підкласу.

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

Шаблонний метод – це поведінковий патерн, що визначає *алгоритм* дій на рівні методів. Фабричний метод – це породжувальний патерн, що

спеціалізується виключно на створенні об'єктів (і часто виступає одним із кроків усередині Шаблонного методу).

14.Яку функціональність додає шаблон «Міст»?

Він додає можливість змінювати реалізацію об'єкта під час виконання програми (динамічно), а також дозволяє розробляти та розширювати ієрархії класів абстракції та реалізації незалежно одна від одної.

Висновок: під час виконання лабораторної роботи, було вивчено призначення та структуру шаблонів проєктування «Mediator», «Facade», «Bridge» та «Template method». Набуто практичних навичок їх застосування для побудови гнучкої архітектури програмного забезпечення.На прикладі системи «Особиста бухгалтерія» було успішно реалізовано структурний шаблон «Bridge». Це дозволило відокремити абстракцію фінансових звітів (логіку їх формування) від реалізації їх відображення (форматів виводу, таких як текст або PDF). Такий архітектурний підхід забезпечив незалежність цих двох складових, дозволяючи додавати нові типи звітів або нові формати файлів без необхідності модифікації існуючого коду та без експоненціального зростання кількості класів.