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 «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Виконала:
студентка групи ІА-33
Самойленко Анастасія

Перевірив:
асистент кафедри ІСТ
Мягкий Михайло Юрійович

Київ 2025

Тема: Патерни проектування.
Мета: Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.
Тема роботи:
27. Особиста бухгалтерія (state, prototype, decorator, bridge, flyweight, SOA)
Програма повинна бути наочним засобом для ведення особистих фінансів: витрат і прибутку; з можливістю встановлення періодичних витрат / прибутку (зарплата і орендна плата); введення сканованих чеків з відповідними статтями витрат; побудова статистики; експорт/імпорт в Excel, реляційні джерела даних; різні рахунки; ведення єдиного фонду на всі рахунки (всією сім'єю) – на особливі потреби (ремонт, автомобіль, відпустка); можливість введення вкладів / кредитів для контролю банківських рахунків (звірка нарахованих відсотків з необхідними і т.д.).

Вступ

Проєктування програмних систем передбачає створення гнучкої, масштабованої та легко підтримуваної архітектури. Одним із ефективних способів досягнення цих цілей є використання шаблонів проєктування, які дозволяють стандартизувати структуру коду та покращити взаємодію між компонентами.

У даній лабораторній роботі розглядаються та вивчаються чотири структурно-поведінкові шаблони: Mediator, Facade, Bridge та Template Method. Їх застосування дозволяє знизити зв’язність модулів, спростити складні взаємодії між об’єктами, відокремити абстракції від реалізацій та стандартизувати алгоритми роботи.

Метою роботи є вивчення структури зазначених шаблонів та набуття практичних навичок їх використання у розробці програмної системи, зокрема шляхом інтеграції їх у функціональні модулі реального застосунку.

Теоретичні відомості

Шаблон (патерн) проєктування — це формалізований опис типового рішення, яке багаторазово зустрічається під час розроблення інформаційних систем. Він містить узагальнений спосіб вирішення певної задачі проєктування, рекомендації щодо його застосування та має загальновживану назву.

Головна мета використання шаблонів — повторне застосування вже знайдених ефективних рішень у нових ситуаціях. Важливою умовою є коректне моделювання предметної області, що дозволяє правильно визначити задачу та обрати відповідний патерн.

Застосування шаблонів проєктування забезпечує низку переваг:

- створення структурованої та логічної моделі системи;
- підвищення наочності та спрощення вивчення архітектури;
- глибше опрацювання структури програмного продукту;
- підвищення стійкості системи до змін вимог;
- спрощення подальшого доопрацювання та розширення;
- полегшення інтеграції систем і взаєморозуміння між розробниками завдяки спільному “словнику проєктування”.

Шаблон «Mediator»

Патерн «Mediator» (Посередник) використовується для упорядкування взаємодії між об’єктами через спеціальний центральний об’єкт-посередник, замість того щоб об’єкти напряму зберігали посилання один на одного. На відміну від патерну «Команда», тут зосереджується логіка саме взаємодій між компонентами.

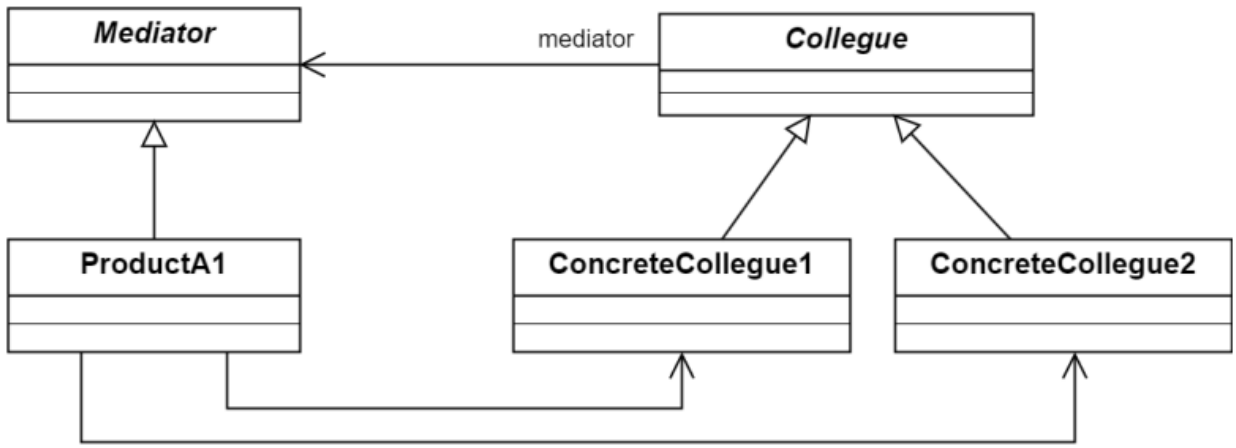
Шаблон доцільно застосовувати тоді, коли багато об’єктів мають складні та численні зв’язки між собою. У такому випадку «медіатор» бере на себе координацію дій, подібно до диригента, який керує окремими інструментами, забезпечуючи узгоджену роботу всієї системи.

Проблема. У великих інтерфейсах, де присутні десятки або сотні елементів, одна дія може впливати на багато інших компонентів. Це породжує численні зв’язки між об’єктами, ускладнює додавання нового функціоналу та робить систему важкою для супроводу.

Рішення. Патерн «Mediator» пропонує винести усю взаємодію в окремий клас-посередник. Компоненти більше не знають один про одного — тільки про медіатор, який і виконує необхідну логіку. Це різко знижує зв’язність та спрощує підтримку. За потреби посередника можна підмінити, наприклад, для тестування.
Переваги:

- зменшення залежностей між об’єктами;
- спрощення структури та легша підтримка коду;
- можливість підключати нові типи посередників без змін у компонентах.

Недоліки:
– посередник може поступово стати надто «важким» та перетворитися на «God Object».
Структура патерну Медіатор на рівні об’єктів:



Шаблон «Facade»

Патерн «Facade» (Фасад) створює єдиний спрощений інтерфейс для роботи зі складною підсистемою. Замість того щоб клієнт взаємодіяв із багатьма класами та їхніми деталями, фасад надає один уніфікований спосіб доступу, приховуючи внутрішню реалізацію. Це допомагає уникнути «спагеті-коду» та ізолює користувача від змін всередині підсистеми.
При потребі окремі компоненти підсистеми можуть залишатися доступними напряму, але головна ідея — мінімізувати зайві залежності та спростити використання системи.

Проблема. Складні компоненти, що включають багато класів (наприклад, робота з різними протоколами чи типами запитів), важко правильно налаштувати та використовувати. Зовнішні системи знають про внутрішню структуру, тому будь-які зміни в підсистемі стають трудомісткими — доводиться переробляти код у багатьох місцях.

Рішення. Патерн «Facade» пропонує створити один «вхідний» клас, який інкапсулює всю роботу підсистеми. Наприклад, клас InternetClient може містити методи для налаштування та відправлення запитів незалежно від того, як це реалізовано всередині. Користувач працює лише з простим публічним інтерфейсом, а внутрішню структуру можна змінювати без впливу на зовнішній код.

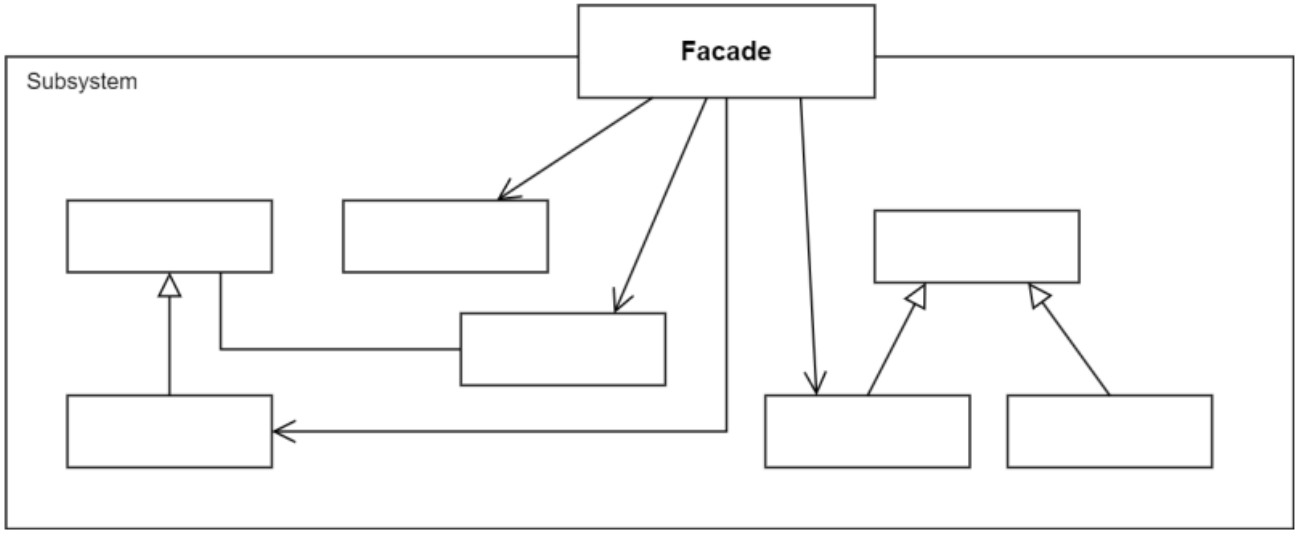
Переваги:

- приховує складність підсистеми;
- робить інтерфейс простішим для використання;
- дозволяє змінювати внутрішню реалізацію без впливу на клієнтів.

Недоліки:

– зменшує гнучкість, оскільки фасад може приховувати частину можливостей підсистеми.

Структура патерну Фасад:



Шаблон «Bridge»

Патерн «Bridge» (Міст) використовується для розділення абстракції та її реалізації, щоб вони могли змінюватися незалежно одна від одної. Це особливо корисно тоді, коли існує багато варіантів абстракцій і одночасно багато варіантів реалізацій, і поєднувати їх потрібно без створення громіздкої ієрархії класів.

Проблема

Уявімо графічний редактор, де необхідно відображати різні фігури (лінії, кола) на різних пристроях (екран, принтер, bitmap). Якщо реалізовувати це напряму, виникає вибух класів: LineDraw, LinePrint, LineBitmap, CircleDraw, CirclePrint тощо. Додавання будь-якої нової фігури або нового способу відображення множить кількість класів і ускладнює підтримку коду.

Рішення

Патерн «Bridge» пропонує розділити систему на дві незалежні ієрархії:

1. Абстракція (Shape) – фігури, які потрібно відобразити.
2. Реалізація (DrawApi) – конкретні способи відтворення.

Абстракція містить посилання на реалізацію та делегує їй виконання операцій. Таким чином, додаючи нову фігуру, ми не змінюємо драйвери відображення, а додаючи новий драйвер — не змінюємо фігури.

Переваги та недоліки

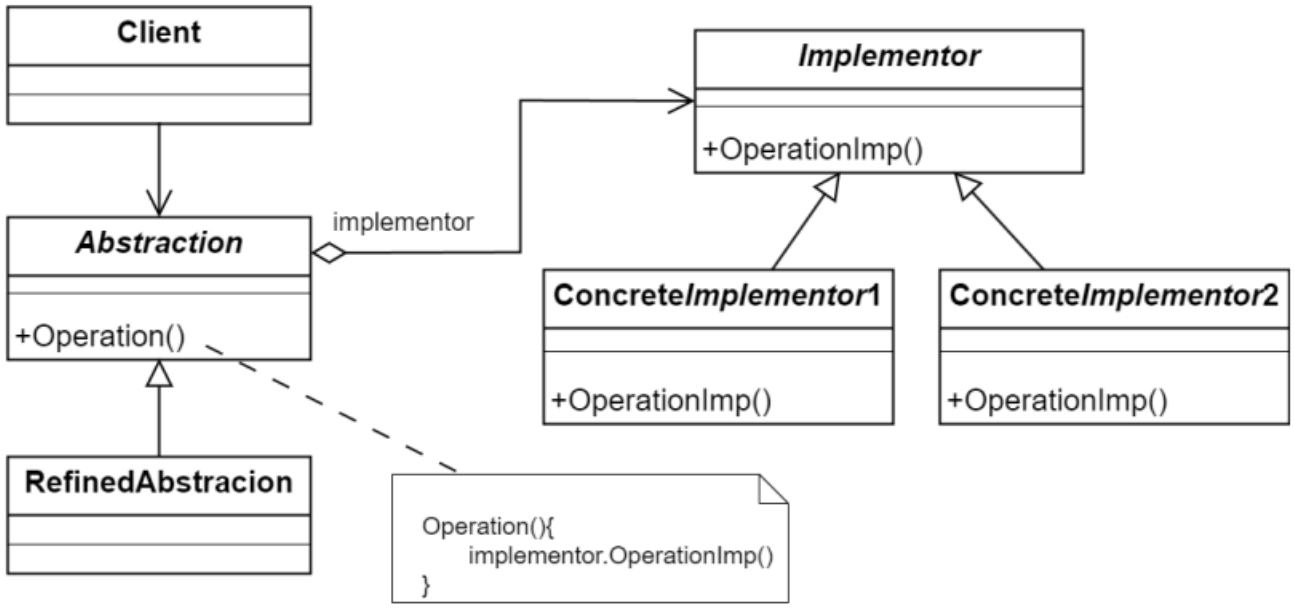
Переваги:

- Абстракція та реалізація розвиваються незалежно.
- Код стає гнучкішим, простішим у розширенні та підтримці.
- Відсутній вибух кількості класів при додаванні нових варіантів.

Недоліки:

– Вводиться додаткова складність через поділ класів і проміжні рівні.

Структура шаблону Міст:



Шаблон «Template Method»

Патерн «Template Method» дозволяє визначити кроки алгоритму в базовому класі, а деталізацію окремих кроків — залишити підкласам. Абстрактний клас задає шаблон алгоритму, а підкласи перевизначають лише ті частини, які повинні виконуватись по-різному. Це зручно, коли алгоритм загалом однаковий, але має відмінності в окремих етапах.

Проблема

Уявімо застосунок для редагування відео, який уже підтримує формат MPEG-4. При додаванні підтримки MPEG-2, MPEG-1 та H.262 розробник вносить усе більше умов у наявний клас. З кожним новим форматом кількість розгалужень зростає, код стає заплутаним і важким у підтримці. Загальний алгоритм обробки відео однаковий приблизно на 70%, але специфічні фрагменти змішані з основною логікою, що ускладнює розвиток системи.

Рішення

Патерн «Template Method» пропонує винести спільний алгоритм у базовий клас, а змінні частини оформити як окремі віртуальні методи.

Підкласи перевизначають тільки специфічні кроки — наприклад, читання, декодування кадрів чи обробку звуку. Для кожного формату створюється окремий клас, який реалізує свою версію цих кроків, тоді як основна послідовність алгоритму залишається незмінною в базовому класі.

Після цього система стає зрозумілішою: додати новий формат — це просто створити новий підклас і перевизначити потрібні методи, не торкаючись логіки інших форматів.

Переваги та недоліки

Переваги:

- Значне повторне використання загального коду.
- Упорядкування алгоритмів, які мають однакову структуру.
- Полегшення розширення системи за рахунок нових підкласів.

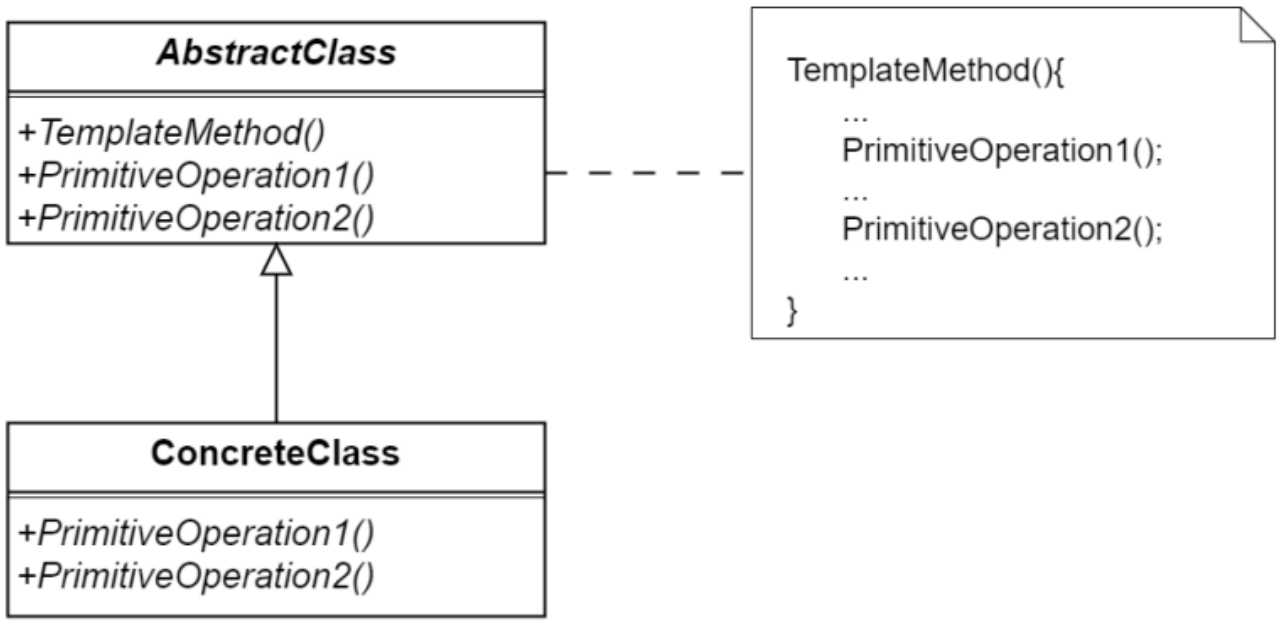
Недоліки:

– Жорстка фіксація структури алгоритму в базовому класі.

– Можливе порушення принципу підстановки Лісков, якщо підклас змінює базову поведінку.

– Великі шаблони з багатьма віртуальними методами можуть бути важкими у підтримці.

Структура патерна Шаблонний метод:



Хід роботи

1) Діаграма класів частини функціоналу робочої програми. (рис. 1)

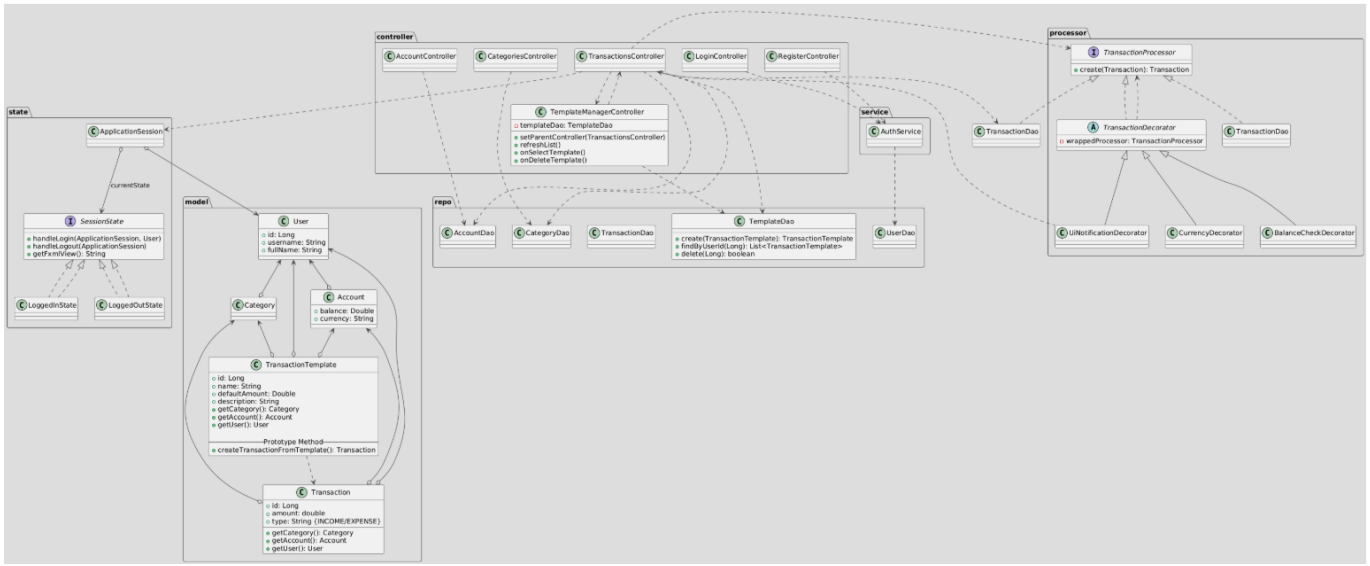
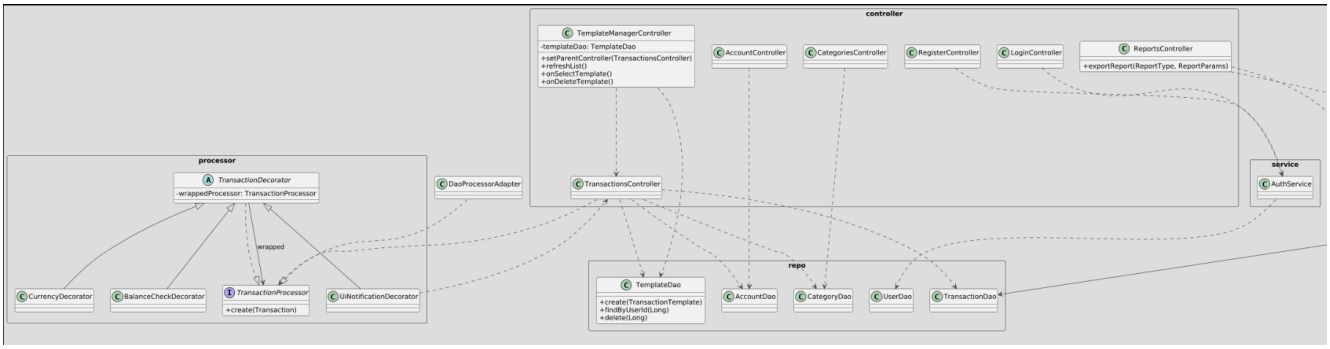


Рисунок 1 – Діаграма класів



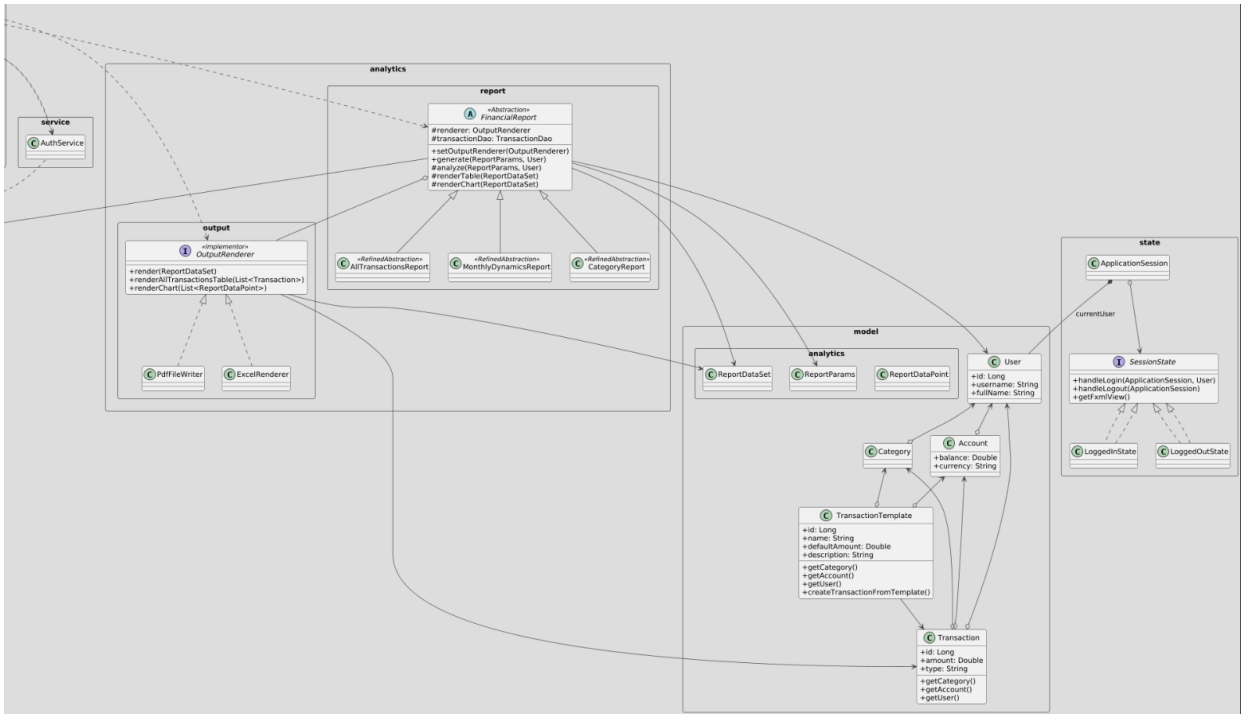


Рисунок 2 – Діаграма класів з патерном Bridge

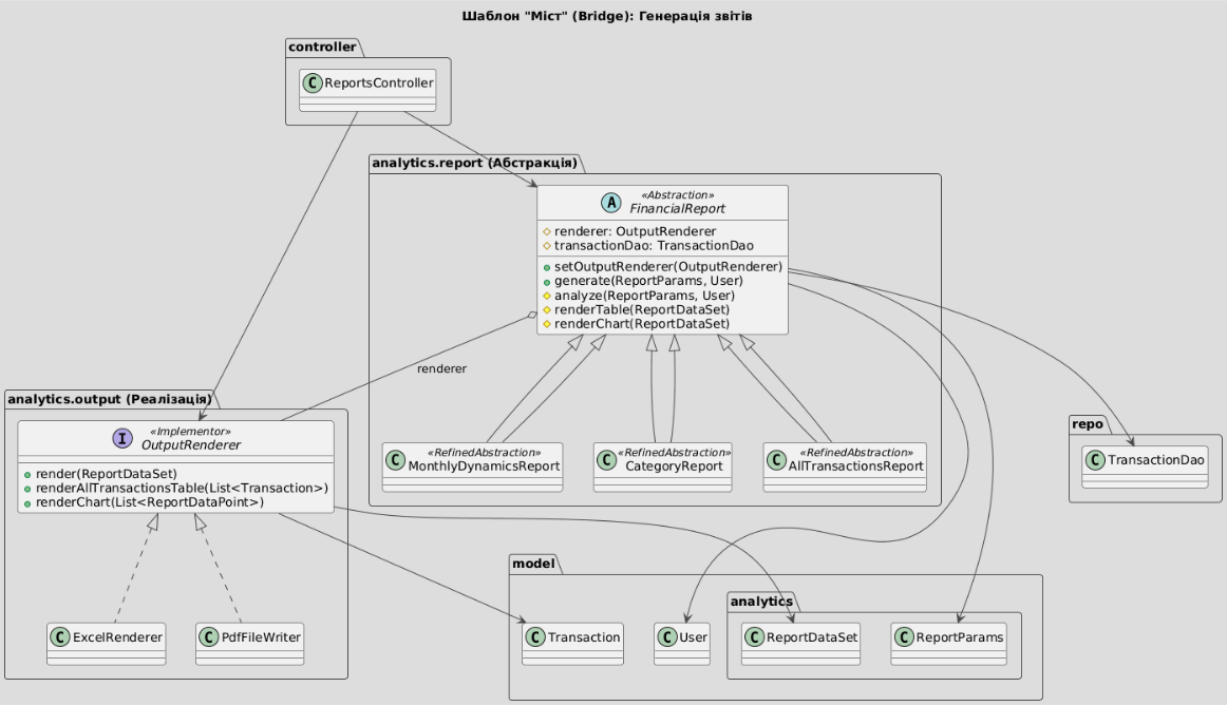


Рисунок 3 – Представлення самого шаблону на діаграмі

2) Реалізація Шаблону «Міст» (Bridge)

Патерн Міст (Bridge) належить до структурних патернів проєктування. Його основна мета — відокремити абстракцію від її реалізації, щоб обидві могли розвиватися незалежно одна від одної. Це досягається завдяки створенню двох незалежних ієрархій класів: ієрархії Абстракції (Abstraction) та ієрархії Реалізації (Implementation).

У проєкті управління особистими фінансами шаблон Міст був застосований для створення гнучкої системи генерації та експорту фінансових звітів. Він дозволяє незалежно додавати нові типи звітів (наприклад, "Звіт за категоріями", "Щомісячна динаміка") та нові формати експорту (наприклад, PDF, Excel) без зміни вже наявного коду.

1. Структура Патерну «Міст»

Реалізація патерну базується на двох ключових ієрархіях, пов'язаних інтерфейсом:

1.1. Ієрархія Реалізації (Implementation) — OutputRenderer

Це ієрархія, що відповідає за виведення даних у певний формат.

- Інтерфейс Реалізації:

OutputRenderer визначає спільний контракт для всіх форматів виведення. Він містить низькорівневі методи для рендерингу (наприклад, render(ReportDataSet dataSet), renderAllTransactionsTable(List<Transaction>), renderChart(List<ReportDataPoint>)).

- Конкретні Реалізації (Concrete Implementors):
    - ExcelRenderer: Реалізує OutputRenderer, використовуючи бібліотеку Apache POI для формування файлів у форматі XLSX.
    - PdfFileWriter: Реалізує OutputRenderer, використовуючи бібліотеку iText для формування файлів у форматі PDF.

1.2. Ієрархія Абстракції — FinancialReport

Це ієрархія, що відповідає за логіку аналізу та формування звітних даних.

- Абстракція (Abstraction):
    - FinancialReport визначає логіку високого рівня (метод generate()). Він містить посилання на об'єкт OutputRenderer (Міст).
    - Клас делегує завдання виведення даних об'єкту Реалізації через методи renderTable() та renderChart().
- Уточнена Абстракція:
    - MonthlyDynamicsReport: Визначає, як аналізувати дані для звіту про динаміку (ключ — місяць, значення — сума).
    - CategoryReport: Визначає, як агрегувати дані за категоріями.
    - AllTransactionsReport: Визначає, як представити детальний список усіх транзакцій.

2. Принцип Роботи та Переваги

Зв'язок між цими двома ієрархіями встановлюється в FinancialReport через метод setOutputRenderer(), де об'єкту звіту передається конкретна реалізація рендерера (Міст).

Як працює Міст:

1. Клієнт створює об'єкт певного звіту (наприклад, new MonthlyDynamicsReport(...)).
2. Клієнт встановлює необхідний рендерер (наприклад, report.setOutputRenderer(new ExcelRenderer(...))).

3. Коли клієнт викликає report.generate(), звіт виконує логіку аналізу (analyze()) та використовує внутрішній об'єкт renderer для виведення результату у вибраний формат.

Основні переваги використання цього шаблону:

- Незалежний розвиток абстракції та реалізації:
    - Розширення звітів: Якщо ми захочемо додати новий звіт, наприклад, AnnualBudgetReport, нам потрібно лише створити новий клас, успадкований від FinancialReport. Нам не потрібно змінювати ExcelRenderer або PdfFileWriter.
    - Розширення форматів: Якщо в майбутньому знадобиться експорт у формат JSON або CSV, ми просто створюємо новий клас (JsonRenderer або CsvRenderer), який реалізує OutputRenderer. Нам не потрібно змінювати логіку жодного звітного класу (MonthlyDynamicsReport чи CategoryReport).
- Чистий та масштабований код:
    - Код кожного звіту повністю зосереджений на аналізі даних, а не на низькорівневих деталях форматування (робота з комірками Excel, елементами PDF тощо).
    - Це дозволяє уникнути "вибуху класів", де кожен звіт мав би окремий метод для кожного формату (наприклад, MonthlyReport.generateExcel(), MonthlyReport.generatePdf()), що різко збільшило б складність і дублювання коду.

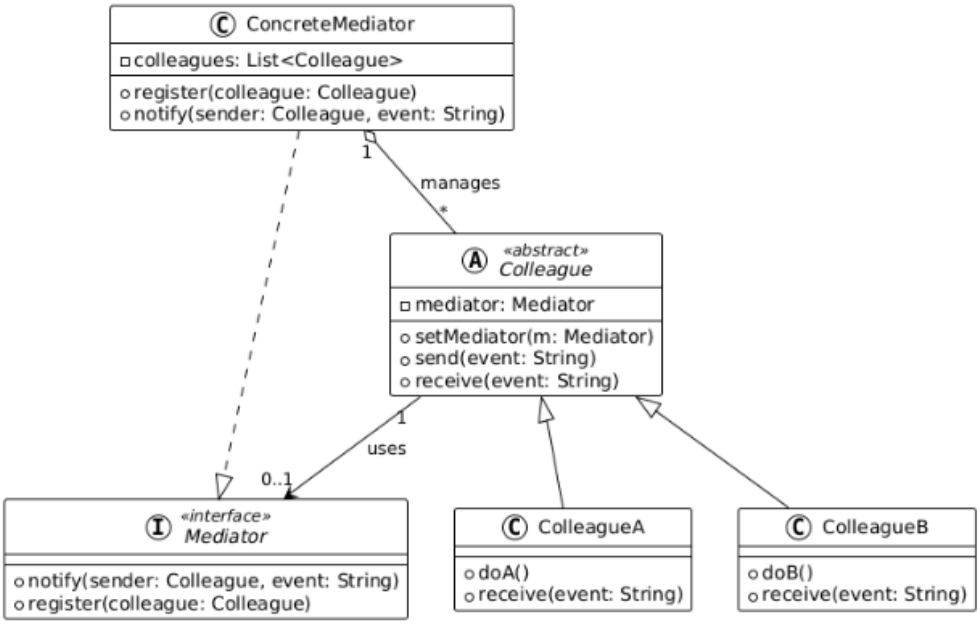
Таким чином, шаблон Міст забезпечив чітку, розширювану та гнучку архітектуру для системи звітів, дозволяючи легко додавати нові аналітичні функції або формати експорту.

Відповіді на контрольні запитання:

1. Яке призначення шаблону «Посередник»?

Посередник — це поведінковий патерн проектування, що дає змогу зменшити зв’язаність великої кількості класів між собою, завдяки переміщенню цих зв’язків до одного класу-посередника.

2. Нарисуйте структуру шаблону «Посередник».



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

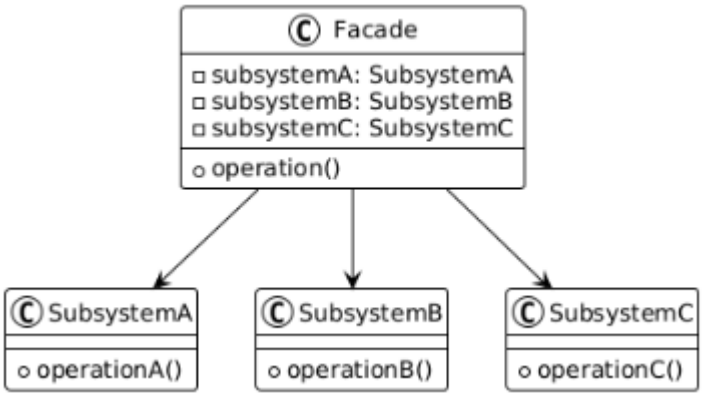
1. Mediator — інтерфейс для обміну повідомленнями між компонентами.
2. ConcreteMediator — реалізує координацію взаємодії компонентів.
3. Component (BaseComponent) — базовий клас з посиланням на медіатора.
4. ConcreteComponents — компоненти, що звертаються до посередника, а не один до одного.

Взаємодія:
Компоненти викликають методи Посередника, а той вирішує, як реагують інші об’єкти.

4. Яке призначення шаблону «Фасад»?

Фасад — це структурний патерн проектування, який надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку.

5. Нарисуйте структуру шаблону «Фасад».



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

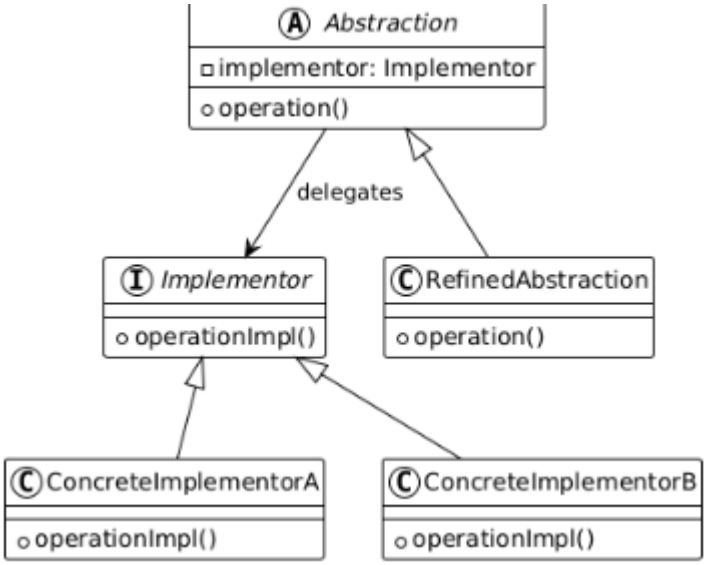
1. Facade — надає простий метод/інтерфейс.
2. Subsystem classes (A, B, C) — складні частини підсистеми з власною логікою.

Взаємодія:
Клієнт працює тільки з фасадом, а фасад делегує виклики внутрішнім підсистемам.

7. Яке призначення шаблону «Міст»?

Міст — це структурний патерн проектування, який розділяє один або кілька класів на дві окремі ієрархії — абстракцію та реалізацію, дозволяючи змінювати код в одній гілці класів, незалежно від іншої.

8. Нарисуйте структуру шаблону «Міст».



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

1. Abstraction — верхній рівень абстракції.
2. RefinedAbstraction — розширена абстракція.
3. Implementor — інтерфейс реалізації.

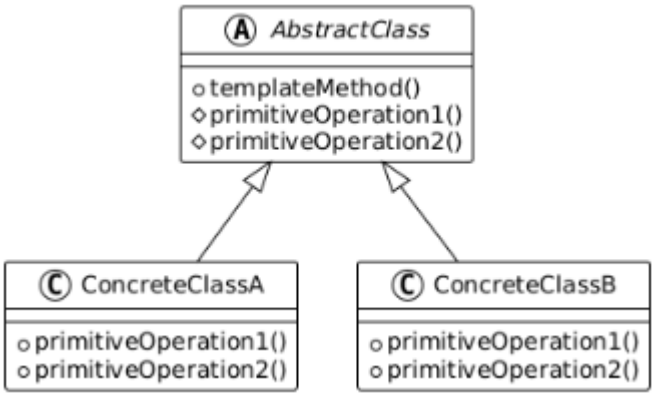
4. ConcreteImplementor — конкретні реалізації.

Взаємодія відбувається через виклик клієнтом Шаблонного методу, який, використовуючи поліморфізм, виконує послідовність кроків, частину з яких реалізовано у підкласах, зберігаючи загальну структуру алгоритму незмінною.

10.Яке призначення шаблону «Шаблоний метод»?

Шаблонний метод — це поведінковий патерн проектування, який визначає кістяк алгоритму, перекладаючи відповідальність за деякі його кроки на підкласи. Патерн дозволяє підкласам перевизначати кроки алгоритму, не змінюючи його загальної структури

11.Нарисуйте структуру шаблону «Шаблоний метод».



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

1. Абстрактний клас визначає незмінний Шаблонний метод (скелет алгоритму), який викликає послідовність Примітивних операцій (деякі з них абстрактні).
2. Конкретні класи успадковують Абстрактний клас і надають специфічні реалізації для абстрактних Примітивних операцій.

Взаємодія відбувається через виклик клієнтом Шаблонного методу, який, використовуючи поліморфізм, виконує послідовність кроків, частину з яких реалізовано у підкласах, зберігаючи загальну структуру алгоритму незмінною.

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

Шаблонний метод визначає алгоритм і дозволяє змінювати окремі його кроки.

Фабричний метод визначає спосіб створення об’єктів, делегуючи це підкласам.

Template Method керує логікою, Factory Method — створенням.

14.Яку функціональність додає шаблон «Міст»?

Дозволяє масштабувати програму в двох напрямках: розширювати абстракції та реалізації окремо.

Уникає надмірного успадкування.

Забезпечує незалежність рівнів: UI, логіка, інтерфейси пристроїв, формати даних тощо.

Висновок:

У ході лабораторної роботи було розглянуто структуру та принципи функціонування шаблонів проєктування Mediator**,** Facade**,** Bridge **та** Template Method**.** Опрацьовано їх призначення, ключові компоненти та взаємодію між класами, а також особливості застосування цих патернів у різних програмних ситуаціях. Крім того, набуті практичні навички побудови UML-діаграм та роботи зі структурними і поведінковими патернами проектування.

У практичній частині було реалізовано патерн Міст (Bridge)**,** що дозволив розділити абстракцію та реалізацію, забезпечивши незалежність їх розвитку. Використання цього патерну дало змогу гнучко поєднувати різні реалізації з різними абстракціями, підвищивши модульність, розширюваність та підтримуваність програмного забезпечення. Такий підхід спростив інтеграцію нових функціональних можливостей без зміни існуючих компонентів системи.