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-31
Кизим Є. К.

Київ 2025

Зміст

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.

Моя тема: 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.).

Теоретичні відомості

1. Детальний опис досліджуваних шаблонів

- **Mediator (Посередник)**
    - Призначення: Використовується для визначення взаємодії об'єктів за допомогою окремого об'єкта-посередника, замість того щоб об'єкти посилалися напряму один на одного.

- Принцип роботи: Колеги (взаємодіючі об'єкти) не знають один про одного, а спілкуються лише з Посередником. Це усуває сильну зв'язність («павутиння» залежностей) і дозволяє змінювати логіку взаємодії незалежно від компонентів.
- Аналогія: Нагадує диригента в оркестрі, який керує вступом кожного інструменту.

- **Facade (Фасад)**
  - Призначення: Надає уніфікований інтерфейс для доступу до набору інтерфейсів у підсистемі. Фасад визначає інтерфейс вищого рівня, який спрощує використання підсистеми.
  - Принцип роботи: Фасад приховує складність системи (безліч класів, залежностей, налаштувань) за простим інтерфейсом. Клієнт взаємодіє тільки з Фасадом, а Фасад перенаправляє виклики відповідним об'єктам підсистеми.
  - Переваги: Зменшує кількість залежностей між клієнтом і складною системою, запобігаючи появі «спагеті-коду».
- **Bridge (Міст)**
  - Призначення: Використовується для відокремлення абстракції від її реалізації, щоб вони могли змінюватися незалежно одна від одної.
  - Застосування: Ефективний, коли існує кілька різних абстракцій (наприклад, типи фігур), над якими можна виконувати дії різними способами (наприклад, малювати різними графічними API). Дозволяє уникнути комбінаторного вибуху кількості класів .
- **Template Method (Шаблонний метод)**
  - Призначення: Визначає "скелет" алгоритму в базовому класі, але делегує реалізацію окремих кроків підкласам.
  - Відмінність: На відміну від «Strategy», який замінює весь алгоритм, «Template Method» змінює лише окремі його частини, зберігаючи загальну структуру виконання.

2. Обґрунтування вибору шаблону для реалізації Для оптимізації архітектури системи «Особиста бухгалтерія» було обрано структурний шаблон **Facade (Фасад)**.

Поточна реалізація клієнтської частини була перевантажена прямою взаємодією з численними сервісами, репозиторіями та фабриками (TransactionService, TransactionRepository, ReportContext, TransactionCreator).

Це призводило до сильної зв'язності коду та дублювання логіки ініціалізації.

Впровадження класу AccountingFacade дозволило:

1. Створити єдину точку входу для всіх фінансових операцій (додавання, дублювання, експорт).
2. Приховати складність вибору конкретних фабрик (IncomeCreator vs ExpenseCreator) та стратегій звітування від UI.
3. Зробити код візуальних форм чистішим та незалежним від змін у бізнес-логіці.

Хід роботи

Діаграма класів реалізації патерну Facade

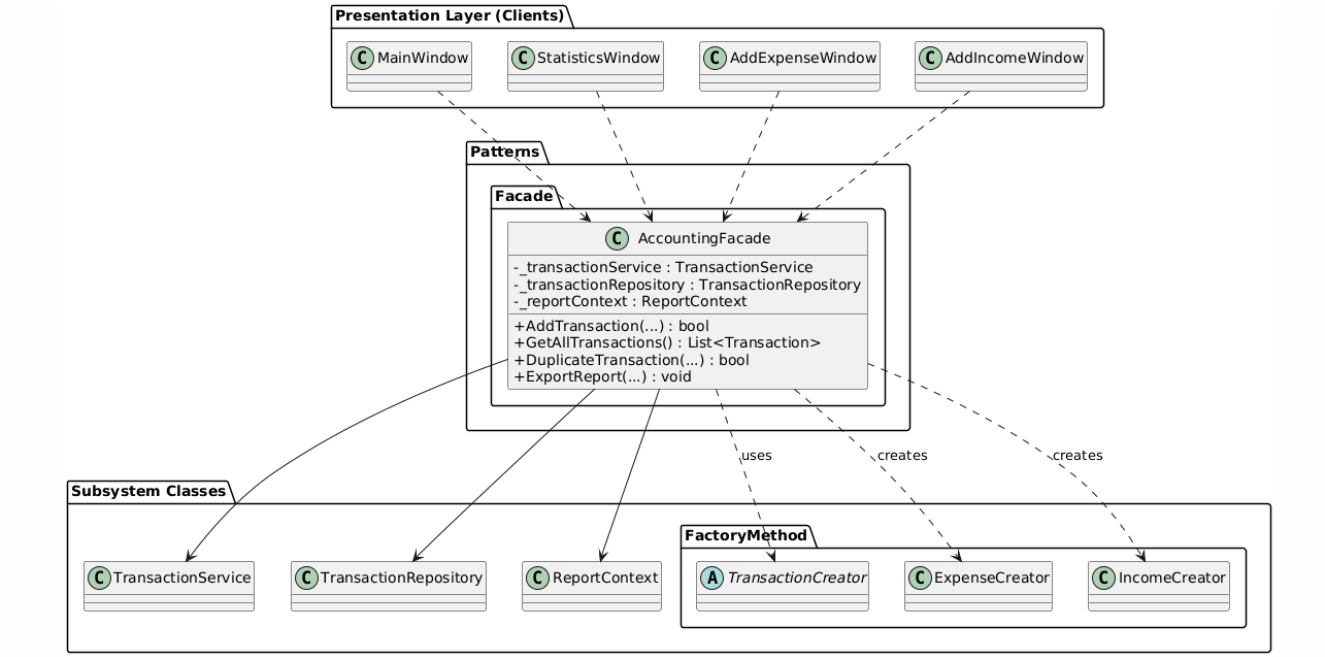


Рисунок 1 – Діаграма класів реалізації патерну Facade

Опис реалізації патерну «Facade»:

Для спрощення взаємодії між користувацьким інтерфейсом та бізнес-логікою програми було впроваджено структурний шаблон **Facade (Фасад)**.

Було створено клас AccountingFacade, який інкапсулює ініціалізацію та використання компонентів підсистеми (TransactionService, TransactionRepository, ReportContext) та керує вибором відповідних патернів (Factory Method для створення транзакцій, Strategy для експорту звітів). Клієнтські класи (вікна програми) тепер звертаються виключно до методів фасаду, що значно знизило зв'язність коду.

Фрагменти програмного коду:

Клас Фасаду (Patterns/Facade/AccountingFacade.cs):

```
using System;
using System.Collections.Generic;
using personalAccounting.Models;
using personalAccounting.Patterns.FactoryMethod;
using personalAccounting.Patterns.Strategy;
using personalAccounting.Repositories;
using personalAccounting.Services;

namespace personalAccounting.Patterns.Facade
{
   public class AccountingFacade
   {
      private readonly TransactionService _transactionService;
      private readonly TransactionRepository _transactionRepository;
      private readonly ReportContext _reportContext;

      public AccountingFacade()
      {
         _transactionService = new TransactionService();
         _transactionRepository = new TransactionRepository();
         _reportContext = new ReportContext();
      }

      public bool AddTransaction(decimal amount, string category, string description,
Transaction.TransactionType type)
      {
         TransactionCreator creator;

         if (type == Transaction.TransactionType.Income)
         {
```

```
            creator = new IncomeCreator();
        }
        else
        {
            creator = new ExpenseCreator();
        }

        Transaction transaction = creator.Create(amount, category, description,
DateTime.Now);

        if (type == Transaction.TransactionType.Income)
        {
            return _transactionService.AddIncome(transaction, 1);
        }
        else
        {
            return _transactionService.AddExpense(transaction, 1);
        }
    }

    public List<Transaction> GetAllTransactions()
    {
        return _transactionRepository.GetAll();
    }

    public bool DuplicateTransaction(Transaction original)
    {
        Transaction clone = original.Clone();

        if (clone.Type == Transaction.TransactionType.Income)
```

```
            return _transactionService.AddIncome(clone, 1);
        else
            return _transactionService.AddExpense(clone, 1);
    }

    public void ExportReport(List<Transaction> data, string filePath, string format)
    {
        if (format == "xlsx")
        {
            _reportContext.SetStrategy(new XLSXReportStrategy());
        }
        else
        {
            _reportContext.SetStrategy(new TxtReportStrategy());
        }

        _reportContext.CreateReport(data, filePath);
    }
  }
}
```

Використання Фасаду у вікні витрат (AddExpenseWindow.xaml.cs):

```
using System.Windows;
using System.Windows.Controls;
using personalAccounting.Models;
using personalAccounting.Patterns.Facade;

namespace personalAccounting
{
```

```
public partial class AddExpenseWindow : Window
{
    private readonly AccountingFacade _facade;

    public AddExpenseWindow()
    {
        InitializeComponent();
        _facade = new AccountingFacade();
    }

    private void SaveBtn_Click(object sender, RoutedEventArgs e)
    {
        if (!decimal.TryParse(AmountBox.Text, out decimal amount) || amount <= 0)
        {
            MessageBox.Show("Будь ласка, введіть коректну суму.", "Помилка");
            return;
        }

        string category = (CategoryBox.SelectedItem as
ComboBoxItem)?.Content.ToString();
        string description = DescriptionBox.Text;

        bool success = _facade.AddTransaction(amount, category, description,
Transaction.TransactionType.Expense);

        if (success)
        {
            MessageBox.Show("Витрату успішно додано!", "Успіх");
            this.Close();
        }
```

```
            else
            {
                MessageBox.Show("Помилка при збереженні.", "Помилка");
            }
        }

        private void CancelBtn_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }
    }
}
```

Використання Фасаду у вікні доходів (AddIncomeWindow.xaml.cs):

```
using System.Windows;
using System.Windows.Controls;
using personalAccounting.Models;
using personalAccounting.Patterns.Facade;

namespace personalAccounting
{
    public partial class AddIncomeWindow : Window
    {
        private readonly AccountingFacade _facade;

        public AddIncomeWindow()
        {
            InitializeComponent();
            _facade = new AccountingFacade();
```

```
    }

    private void SaveBtn_Click(object sender, RoutedEventArgs e)
    {
        if (!decimal.TryParse(AmountBox.Text, out decimal amount) || amount <= 0)
        {
            MessageBox.Show("Введіть коректну суму.", "Помилка");
            return;
        }

        string category = (CategoryBox.SelectedItem as
ComboBoxItem)?.Content.ToString();
        string description = DescriptionBox.Text;

        bool success = _facade.AddTransaction(amount, category, description,
Transaction.TransactionType.Income);

        if (success)
        {
            MessageBox.Show("Дохід успішно додано!", "Успіх");
            this.Close();
        }
        else
        {
            MessageBox.Show("Помилка при збереженні.", "Помилка");
        }
    }

    private void CancelBtn_Click(object sender, RoutedEventArgs e)
    {
```

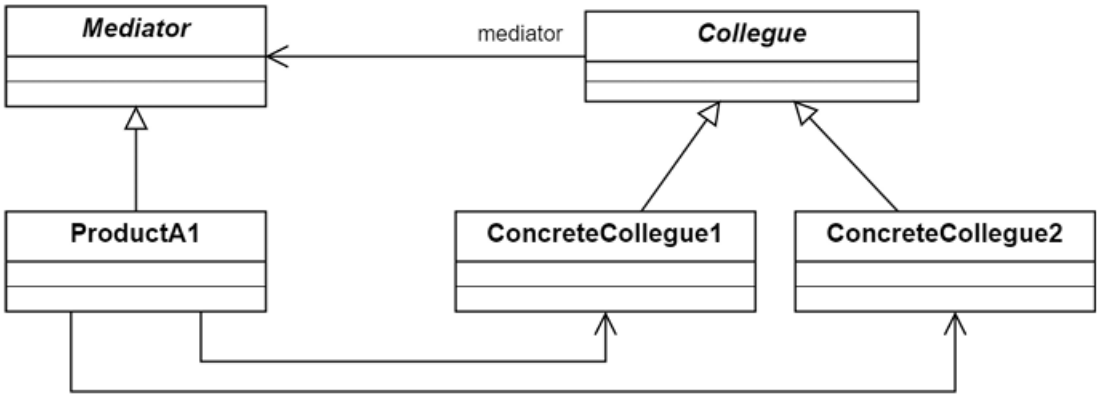
```
            this.Close();
        }
    }
}
```

Контрольні питання

1. Яке призначення шаблону «Посередник»?

Він створює центральний об'єкт-координатор для групи об'єктів. Це усуває хаотичні прямі зв'язки («павутиння») між компонентами, дозволяючи їм спілкуватися лише через посередника.

2. Нарисуйте структуру шаблону «Посередник».



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

Mediator: інтерфейс взаємодії.

ConcreteMediator: реалізує логіку координації.

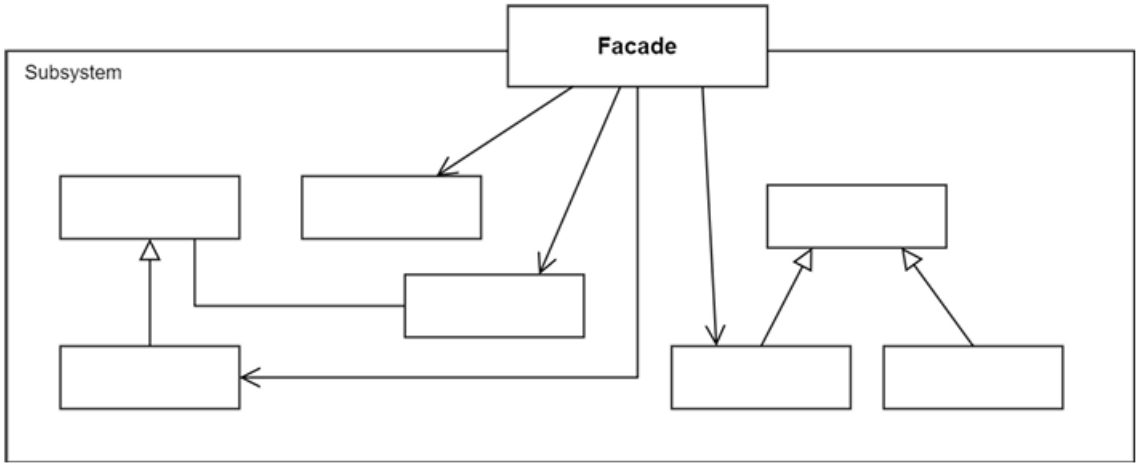
Colleague: класи учасників.

Взаємодія: Колеги не знають один про одного. Вони надсилають події Посереднику, а той вирішує, кому перенаправити запит або яку дію виконати.

4. Яке призначення шаблону «Фасад»?

Надає простий, «фасадний» інтерфейс для роботи зі складною бібліотекою або фреймворком, приховуючи від клієнта заплутану внутрішню структуру системи.

5. Нарисуйте структуру шаблону «Фасад».



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

Facade: спрощений вхід у систему.

Subsystem Classes: класи зі складною бізнес-логікою.

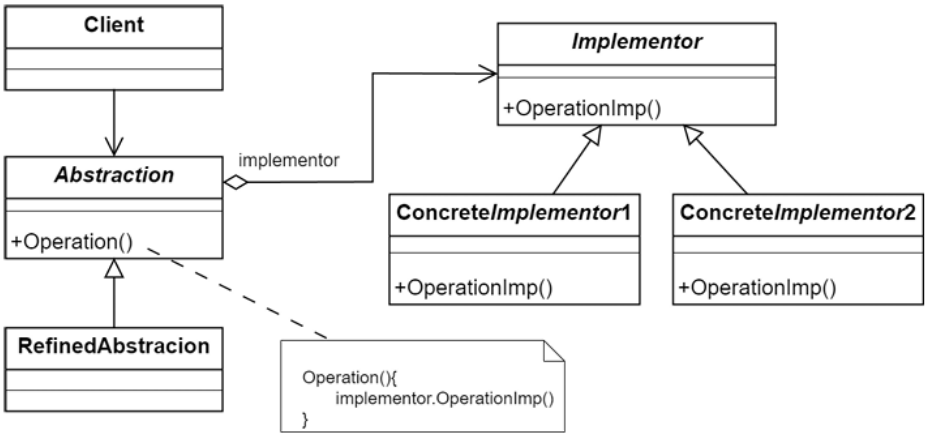
Client: користувач фасаду.

Взаємодія: Клієнт викликає метод Фасаду, а Фасад розподіляє завдання між об'єктами підсистеми. Клієнт не має прямих зв'язків із підсистемою.

7. Яке призначення шаблону «Міст»?

Розділяє один великий клас на дві окремі ієрархії: абстракцію (логіку) та реалізацію (платформу). Це дозволяє змінювати та розширювати їх незалежно одна від одної.

8. Нарисуйте структуру шаблону «Міст».



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

Abstraction: високорівнева логіка керування.

RefinedAbstraction: розширена логіка.

Implementor: спільний інтерфейс для реалізацій.

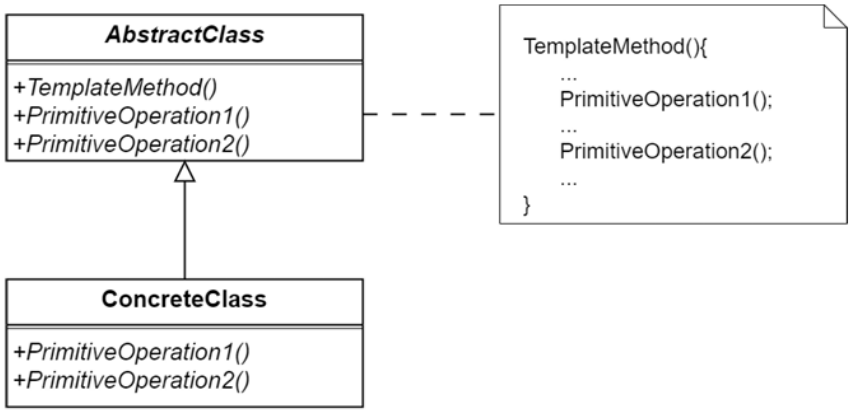
ConcreteImplementor: конкретний код під певну платформу.

Взаємодія: Абстракція містить посилання на об'єкт Implementor і делегує йому виконання низькорівневої роботи.

10. Яке призначення шаблону «Шаблоний метод»?

Задає «скелет» алгоритму в базовому класі, але деталі реалізації окремих кроків залишає на розсуд підкласів, не змінюючи загальної структури виконання.

11. Нарисуйте структуру шаблону «Шаблоний метод».



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

AbstractClass: містить шаблонний метод і абстрактні кроки.

ConcreteClass: реалізує ці кроки.

Взаємодія: Клієнт викликає шаблонний метод, який послідовно запускає кроки алгоритму. Конкретна логіка кроків виконується з підкласу.

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

«Шаблонний метод» — це *поведінковий* патерн, що керує потоком виконання алгоритму. «Фабричний метод» — *породжувальний* патерн, що відповідає лише за створення об'єктів (і часто є просто одним із кроків усередині Шаблонного методу).

14. Яку функціональність додає шаблон «Міст»?

Він дозволяє динамічно (під час роботи програми) змінювати реалізацію об'єкта, а також дає змогу розробляти абстракцію (інтерфейс) та реалізацію (код платформи) паралельно і незалежно.

Висновки: Під час виконання лабораторної роботи я ознайомилась із принципами роботи та структурою структурних та поведінкових шаблонів проєктування: «Mediator» (Посередник), «Facade» (Фасад), «Bridge» (Міст) та «Template Method» (Шаблонний метод).

Практична частина роботи полягала в реалізації структурного шаблону Facade (Фасад) у системі «Особиста бухгалтерія». Для цього було розроблено клас AccountingFacade, який інкапсулював у собі складну логіку взаємодії з сервісами (TransactionService), репозиторіями (TransactionRepository) та іншими патернами (Factory Method для створення транзакцій, Strategy для звітів). Впровадження фасаду дозволило створити єдину спрощену точку доступу до бізнес-логіки програми, значно зменшити зв'язність між клієнтським кодом (візуальними

формами) та підсистемами, а також зробити код додатку більш чистим, структурованим та легким для супроводу.

Репозиторій: посилання