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

із дисципліни *«Технології розроблення програмного забезпечення»*

Тема: «ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»»

Виконав:

студент групи ІА-23
Пожар Д. Ю.

Перевірив:

Мягкий М.Ю.

Київ 2024

Тема лабораторних робіт:

Варіант 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.).

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми.

Зміст

Хід роботи

Крок 1. Короткі теоретичні відомості

Шаблон **Mediator** призначений для організації взаємодії між об'єктами, зменшуючи прямі залежності між ними. Він вводить центральний об'єкт-посередник, який координує комунікацію між різними компонентами системи. Завдяки цьому шаблон забезпечує слабку зв'язаність і спрощує підтримку та розширення системи, адже об'єкти взаємодіють лише з посередником, а не між собою.

Шаблон **Facade** надає уніфікований інтерфейс до набору інтерфейсів у підсистемі, спрощуючи її використання. Він служить своєрідним "входом" до складної системи, приховуючи її внутрішню складність і деталізацію реалізації. Facade дозволяє клієнтам працювати із системою через простий інтерфейс, полегшуючи інтеграцію та знижуючи рівень залежності між клієнтом і внутрішніми компонентами.

Шаблон **Bridge** розділяє абстракцію та її реалізацію, дозволяючи їм змінюватися незалежно одна від одної. Він особливо корисний у системах, де є потреба в поєднанні кількох варіантів реалізацій і абстракцій. Завдяки цьому шаблону можна створити багатовимірну ієрархію, уникаючи експоненційного зростання кількості класів і забезпечуючи більшу гнучкість у проектуванні.

Шаблон **Template Method** визначає кістяк алгоритму у базовому класі, дозволяючи підкласам змінювати окремі кроки цього алгоритму без зміни його структури. Завдяки цьому досягається повторне використання коду і стандартизація послідовності дій, при цьому забезпечується гнучкість для

реалізації специфічної поведінки в підкласах. Шаблон сприяє реалізації принципу "задати загальне, деталізувати конкретне".

Крок 2. Реалізація шаблону “Bridge”

Для реалізації отримання статистики було вирішено використати шаблон проєктування "Bridge". Цей шаблон дозволяє легко змінювати способи обчислення статистики транзакцій без зміни логіки вищого рівня.

У системі "Особиста бухгалтерія" можуть виникати різні потреби в обчисленні статистики транзакцій:

- Різні джерела даних: Наприклад, статистика може обчислюватися на основі бази даних або з використанням кешованих даних.
- Різні типи статистики: Наприклад, щомісячна, річна або спеціалізована статистика за категоріями.
- Легкість розширення: Необхідно забезпечити можливість додавання нових реалізацій статистики (наприклад, інтеграція з API сторонніх сервісів) без модифікації існуючого коду.

Шаблон "Bridge" дозволяє розділити ці аспекти: абстракція визначає методи для отримання статистики, а реалізація відповідає за обчислення даних.

```
public abstract class TransactionStatistics {
    3 usages
    protected TransactionStatisticsImplementation implementation;

    1 usage  new *
    public TransactionStatistics(TransactionStatisticsImplementation implementation) {
        this.implementation = implementation;
    }

    1 usage  1 implementation  new *
    public abstract BigDecimal calculateIncome(LocalDate startDate, LocalDate endDate, Long accountId);

    1 usage  1 implementation  new *
    public abstract BigDecimal calculateExpenses(LocalDate startDate, LocalDate endDate, Long accountId)
}
```

Рисунок 1 – Абстрактний клас TransactionStatistics

Клас TransactionStatistics визначає загальні методи для отримання статистики (calculateIncome і calculateExpenses). Цей клас не знає про конкретну реалізацію обчислень. Інтерфейс TransactionStatisticsImplementation визначає метод для виконання конкретних обчислень. Наприклад, клас DatabaseTransactionStatisticsImplementation реалізує цей інтерфейс для обчислення статистики на основі даних із бази.

```
public class MonthlyTransactionStatistics extends TransactionStatistics {

    1 usage  new *
    public MonthlyTransactionStatistics(TransactionStatisticsImplementation implementation) { super(implementation); }

    1 usage  new *
    @Override
    public BigDecimal calculateIncome(LocalDate startDate, LocalDate endDate, Long accountId) {
        return implementation.calculateTotalByTypeAndAccount(TransactionType.DEPOSIT, startDate, endDate, accountId);
    }

    1 usage  new *
    @Override
    public BigDecimal calculateExpenses(LocalDate startDate, LocalDate endDate, Long accountId) {
        return implementation.calculateTotalByTypeAndAccount(TransactionType.WITHDRAW, startDate, endDate, accountId);
    }
}
```

Рисунок 2 – MonthlyTransactionStatistics

Клас MonthlyTransactionStatistics реалізує специфічну логіку: щомісячну статистику, використовуючи базову абстракцію.

```
@Service
public class DatabaseTransactionStatisticsImplementation implements TransactionStatisticsImplementation {
    2 usages
    private final TransactionRepository transactionRepository;
    new *
    public DatabaseTransactionStatisticsImplementation(TransactionRepository transactionRepository) {
        this.transactionRepository = transactionRepository;
    }
    2 usages  new *
    @Override
    public BigDecimal calculateTotalByTypeAndAccount(TransactionType type, LocalDate startDate, LocalDate endDate, Long accountId) {
        return transactionRepository.calculateTotalAmountByTypeAndDateRangeAndAccount(type, startDate, endDate, accountId);
    }
}
```

Рисунок 3 – DatabaseTransactionStatisticsImplementation

DatabaseTransactionStatisticsImplementation взаємодіє з базою даних через репозиторій TransactionRepository для виконання обчислень, таких як загальна сума транзакцій певного типу за обраний період і для вказаного рахунку.

Посилання на репозиторій проєкту:
https://github.com/Maphin/personal_accounting_app

Висновок

Результатом виконання лабораторної роботи стало практичне засвоєння принципів і реалізацій шаблону проєктування "Bridge". Було розроблено частину системи "Особиста бухгалтерія", яка дозволяє виконувати розрахунок статистики фінансових транзакцій з використанням гнучкої архітектури. Шаблон "Bridge" забезпечив можливість розділення абстракції та реалізації, що спрощує додавання нових способів обчислення без модифікації існуючого коду. Такий підхід дозволяє ефективно працювати з різними джерелами даних і типами статистики, зберігаючи структурованість та масштабованість проєкту. Виконання роботи дало змогу закріпити теоретичні знання про шаблони проєктування та набути практичного досвіду їх використання в реальних завданнях розроблення програмного забезпечення.