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 №8

із дисципліни *«Технології розроблення програмного забезпечення»*

Тема: «ШАБЛОНИ «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR»»

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 «**Composite**» використовується для роботи з деревоподібними структурами даних, коли об’єкти можуть бути представлені у вигляді дерев із вузлів і листків. Він дозволяє клієнтам однаково працювати як з окремими об'єктами, так і з їх групами. Цей шаблон забезпечує можливість створювати ієрархії об’єктів, де вузли і листя мають однаковий інтерфейс, полегшує додавання нових типів компонентів і знижує складність коду клієнта, адже той працює з об'єктами через єдиний інтерфейс. Наприклад, його можна використовувати у системах роботи з файлами, де файл і папка мають однакові операції, як-от відображення розміру або видалення.

Шаблон «**Flyweight**» оптимізує використання пам’яті за рахунок збереження спільного стану для великої кількості об’єктів. Він дозволяє використовувати один спільний об’єкт для декількох екземплярів з однаковими даними. Цей підхід розділяє стан об'єктів на внутрішній (загальний) і зовнішній (індивідуальний), що дозволяє зменшити витрати на збереження повторюваних даних і ефективно працювати з великою кількістю дрібних об’єктів. Наприклад, у текстових редакторах однакові символи можуть використовувати спільний набір атрибутів, таких як шрифт і розмір.

Шаблон «**Interpreter**» призначений для створення мови або розробки інтерпретаторів для конкретної предметної області. Він дозволяє описувати граматику мови та реалізовувати механізми її обробки. Цей шаблон представляє граматику у вигляді класів, які відповідають за окремі правила, і застосовується тоді, коли потрібен розбір і виконання структурованих виразів. Його часто використовують для обмежених мов або форматів даних,

наприклад, для розбору математичних виразів чи створення SQL-подібних мов запитів.

Шаблон «**Visitor**» дає змогу додавати нові операції до класів без їх модифікації, відокремлюючи алгоритми від структури даних і забезпечуючи їх незалежний розвиток. Він реалізує подвійний диспетчеринг, що дозволяє виконувати дії залежно від типу об’єкта. Цей підхід полегшує додавання нових операцій без зміни існуючих класів і зазвичай використовується у випадках, коли структура даних стабільна, але функціональність може змінюватися. Наприклад, система може виконувати різні операції над елементами, як-от серіалізація об'єктів чи їх валідація.

Крок 2. **Реалізація шаблону “ Flyweight ”**

Для реалізації створення моделі вкладів/кредитів було вирішено використати шаблон проєктування "Flyweight". У системі "Особиста бухгалтерія" для моделі LoanAndDeposit (вклади та кредити) деякі дані, такі як тип кредиту/вкладу**,** початкова сума (principalAmount) та процентна ставка (interestRate), часто повторюються серед багатьох об'єктів. Наприклад, декілька користувачів можуть мати вклади однакового типу з тією ж процентною ставкою та сумою. Шаблон Flyweight дозволяє уникнути дублювання цих даних, зберігаючи їх у централізованому місці.

```java
@Data
@RequiredArgsConstructor
public class LoanAndDepositFlyweight {
    private final Type type;
    private final BigDecimal principalAmount;
    private final BigDecimal interestRate;
}
```

Рисунок 1 – LoanAndDepositFlyweight

Клас LoanAndDepositFlyweight представляє внутрішній (спільний) стан об'єктів, який не змінюється між різними об'єктами LoanAndDeposit. У цьому випадку це type (тип кредиту/вкладу), principalAmount (початкова сума) та interestRate (процентна ставка).

```java
public class LoanAndDepositFlyweightFactory {
    3 usages
    private static final Map<String, LoanAndDepositFlyweight> flyweights = new HashMap<>();

    3 usages  new *
    public static LoanAndDepositFlyweight getFlyweight(Type type, BigDecimal principalAmount, BigDecimal interestRate) {
        String key = type + "-" + principalAmount + "-" + interestRate;

        if (!flyweights.containsKey(key)) {
            LoanAndDepositFlyweight flyweight = new LoanAndDepositFlyweight(type, principalAmount, interestRate);
            flyweights.put(key, flyweight);
        }

        return flyweights.get(key);
    }
}
```

Рисунок 2 – LoanAndDepositFlyweight

Фабрика LoanAndDepositFlyweightFactory відповідає за створення та управління об'єктами LoanAndDepositFlyweight. Зберігає унікальні екземпляри LoanAndDepositFlyweight у карті (HashMap), щоб забезпечити повторне використання.

```java
@Entity(name = "loans_and_deposits")
@Data
public class LoanAndDeposit {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @Column(nullable = false, unique = true)
    private String title;

    @Transient
    private LoanAndDepositFlyweight sharedData;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private Type type;

    @Column(nullable = false, precision = 15, scale = 2, columnDefinition = "DECIMAL(15,2)")
    private BigDecimal principalAmount;

    @Column(nullable = false, precision = 15, scale = 2, columnDefinition = "DECIMAL(5,2)")
    private BigDecimal interestRate;

    @Column(nullable = false)
    private LocalDate startDate;

    @Column(nullable = false)
    private LocalDate endDate;

    new *
    @PostLoad
    public void loadSharedData() {
        this.sharedData = LoanAndDepositFlyweightFactory.getFlyweight(type, principalAmount, interestRate);
    }
}
```

Рисунок 3 – LoanAndDeposit

Клас LoanAndDeposit представляє зовнішній стан об'єктів, який є унікальним для кожного об'єкта (наприклад, id, user, title, startDate, endDate). Включає поле sharedData, яке посилається на об'єкт LoanAndDepositFlyweight для доступу до спільного стану. Метод

@PostLoad використовується для завантаження спільного стану (LoanAndDepositFlyweight) після ініціалізації об'єкта LoanAndDeposit з бази даних.

Висновок

Результатом виконання лабораторної роботи стало вивчення та практичне засвоєння шаблонів проєктування «Composite», «Flyweight», «Interpreter» та «Visitor». У процесі роботи було розглянуто особливості кожного з шаблонів, їхню роль у розробці програмного забезпечення, а також їхній вплив на оптимізацію архітектури систем. Особливу увагу було приділено реалізації шаблону «Flyweight», який був використаний для моделювання вкладів і кредитів у системі «Особиста бухгалтерія». Застосування цього шаблону дало змогу значно зменшити дублювання даних та оптимізувати використання пам’яті шляхом розділення стану об’єктів на спільний і унікальний. Практична реалізація також допомогла краще зрозуміти принципи створення ефективного коду з використанням фабрики для управління об’єктами зі спільним станом. Виконана робота дозволила закріпити теоретичні знання про шаблони проєктування та набути практичних навичок їх застосування у розробці складних програмних систем.