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 №4

із дисципліни *«Технології розроблення програмного забезпечення»*

Тема: «ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»»

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.
4. Скласти звіт про виконану роботу.

Зміст

Короткі теоретичні відомості 4

Реалізація шаблону “State” 5

Висновок 9

Хід роботи

Крок 1. Короткі теоретичні відомості

Singleton забезпечує створення лише одного екземпляра класу та надає глобальну точку доступу до нього. Він використовується для керування ресурсами, такими як конфігурація, кеш чи підключення до бази даних. Для реалізації використовується приватний конструктор, щоб запобігти створенню нових екземплярів, та статичний метод, який повертає той самий об'єкт.

Iterator дозволяє послідовно отримувати доступ до елементів колекції без розкриття її внутрішньої структури. Цей шаблон виділяє логіку перебору елементів у спеціальний об’єкт-ітератор, який зазвичай має методи next() для отримання наступного елемента та hasNext() для перевірки, чи є ще елементи. Колекція повертає відповідний ітератор, що спрощує навігацію.

Proxy створює об’єкт-заступник, який контролює доступ до іншого об’єкта. Цей шаблон використовується для додавання оптимізації, безпеки чи виконання додаткової логіки перед зверненням до основного об’єкта. Наприклад, проксі може бути кешувальним, захисним або віддаленим. Клас-проксі реалізує той самий інтерфейс, що й основний об’єкт, і викликає його методи з додатковою логікою.

State дозволяє об'єкту змінювати свою поведінку залежно від внутрішнього стану. Цей шаблон замінює складні умовні оператори (if, switch) окремими класами, кожен із яких відповідає за конкретний стан. Контекстний клас делегує виконання методів поточному стану, що спрощує підтримку та масштабування коду.

Strategy дозволяє вибирати різні алгоритми для виконання завдання під час роботи програми. Алгоритми виносяться в окремі класи, які реалізують спільний інтерфейс. Контекстний клас отримує об’єкт-стратегію та викликає його методи, що забезпечує гнучкість у зміні поведінки програми без переписування коду.

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

Реалізація шаблону "State" у системі роботи з банківськими рахунками передбачає створення окремих класів для кожного стану рахунку та використання фабрики для управління цими станами.

Ключові компоненти:

1. Account — модель банківського рахунку.
2. AccountState — інтерфейс для визначення поведінки для кожного стану.
3. Реалізації станів: ActiveState, SuspendedState, ClosedState
4. AccountStateFactory — фабрика, яка надає об'єкти станів залежно від статусу рахунку.
5. AccountService — бізнес-логіка для управління рахунками (зняття, поповнення, закриття).

Інтерфейс AccountState

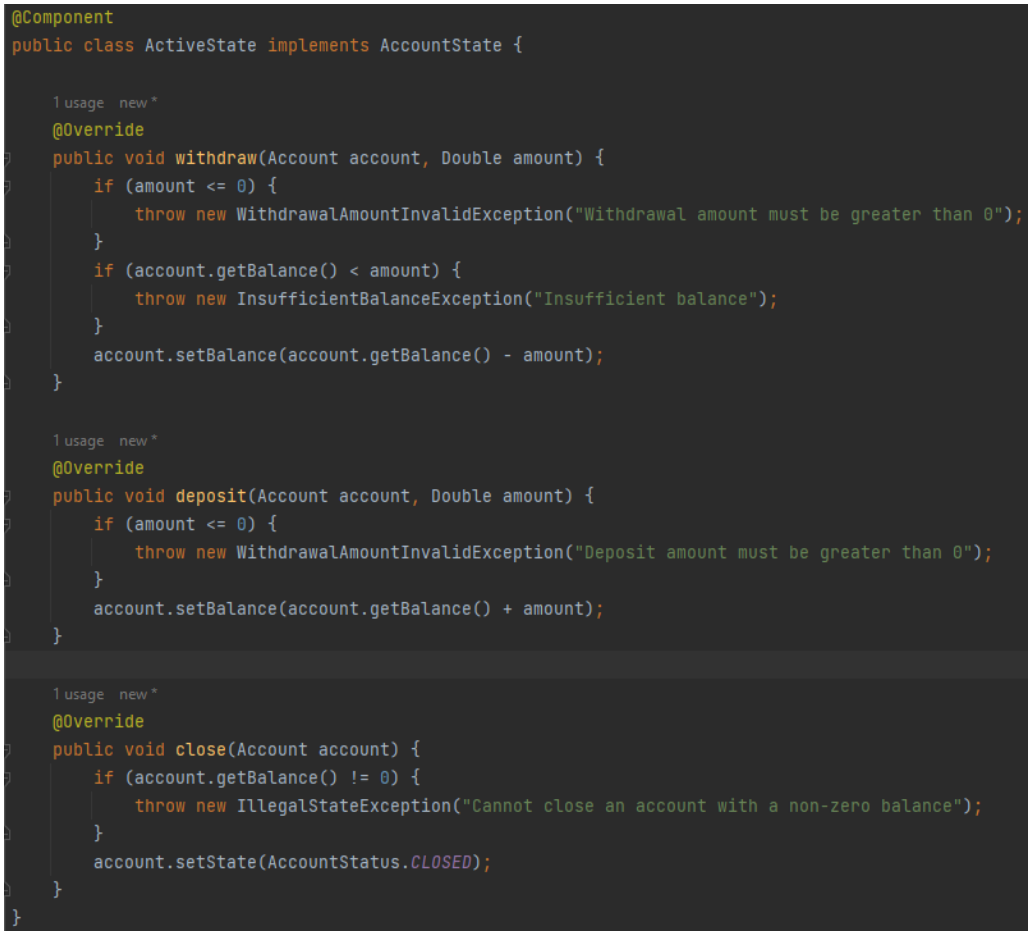
Цей інтерфейс визначає набір методів для реалізації поведінки кожного стану рахунку

```
package com.example.personal_accounting.services.Accounts.State;

import com.example.personal_accounting.models.Account;

8 usages  3 implementations  new *
public interface AccountState {
    1 usage  3 implementations  new *
    void withdraw(Account account, Double amount);
    1 usage  3 implementations  new *
    void deposit(Account account, Double amount);
    1 usage  3 implementations  new *
    void close(Account account);
}
```

Рис1. Інтерфейс AccountState

Класи станів

Для кожного стану рахунку створюється окремий клас, що реалізує інтерфейс AccountState. Наприклад, клас ActiveState описує поведінку активного рахунку, коли дозволені операції поповнення та зняття коштів, а також можливість закриття рахунку за умови, що баланс дорівнює нулю. У класі SuspendedState реалізується поведінка призупиненого рахунку, для якого всі операції заборонені. Клас ClosedState описує закритий рахунок, з яким жодні операції не можуть бути виконані.



```
@Component
public class ActiveState implements AccountState {

    1 usage  new *
    @Override
    public void withdraw(Account account, Double amount) {
        if (amount <= 0) {
            throw new WithdrawalAmountInvalidException("Withdrawal amount must be greater than 0");
        }
        if (account.getBalance() < amount) {
            throw new InsufficientBalanceException("Insufficient balance");
        }
        account.setBalance(account.getBalance() - amount);
    }

    1 usage  new *
    @Override
    public void deposit(Account account, Double amount) {
        if (amount <= 0) {
            throw new WithdrawalAmountInvalidException("Deposit amount must be greater than 0");
        }
        account.setBalance(account.getBalance() + amount);
    }

    1 usage  new *
    @Override
    public void close(Account account) {
        if (account.getBalance() != 0) {
            throw new IllegalStateException("Cannot close an account with a non-zero balance");
        }
        account.setState(AccountStatus.CLOSED);
    }
}
```

Рис2. Реалізація ActiveState

Фабрика

Для визначення відповідного об’єкта стану використовується фабрика AccountStateFactory, яка приймає поточний статус рахунку (AccountStatus) і повертає об’єкт відповідного класу стану. Наприклад, якщо статус рахунку ACTIVE, фабрика поверне об’єкт ActiveState, якщо SUSPENDED — об’єкт SuspendedState, і так далі. Цей підхід спрощує управління станами та ізолює специфічну логіку кожного стану в окремих класах.

```java
@Component
public class AccountStateFactory {

    2 usages
    private final ActiveState activeState;
    2 usages
    private final SuspendedState suspendedState;
    2 usages
    private final ClosedState closedState;

    new *
    public AccountStateFactory(ActiveState activeState, SuspendedState suspendedState, ClosedState closedState) {
        this.activeState = activeState;
        this.suspendedState = suspendedState;
        this.closedState = closedState;
    }

    3 usages  new *
    public AccountState getState(AccountStatus status) {
        switch (status) {
            case ACTIVE:
                return activeState;
            case SUSPENDED:
                return suspendedState;
            case CLOSED:
                return closedState;
            default:
                throw new IllegalArgumentException("Unknown account state: " + status);
        }
    }
}
```

Рис2. Реалізація AccountStateFactory

[Посилання на репозиторій проєкту]

Висновок

У ході виконання лабораторної роботи з проектування системи "Особиста бухгалтерія" було реалізовано шаблон проектування State для управління станами сутності Account. Попередньо було проаналізовано функціональні вимоги, створено інтерфейс для уніфікованої роботи зі станами та класи для кожного стану, що забезпечило ізоляцію логіки та спрощення модифікацій.

Реалізація шаблону дозволила зробити систему більш масштабованою та модульною, спростивши підтримку й тестування. Окрім шаблону State, я ознайомився з іншими шаблонами проектування, такими як Singleton, Iterator, Proxy, та Strategy, що розширило мої знання в проектуванні програмних систем.