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 №5

із дисципліни *«Технології розроблення програмного забезпечення»*

Тема: «ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»»

Виконала:
Студентка групи ІА-24
Сіденко Д.Д.

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

Adapter

Шаблон Adapter дозволяє об'єктам з різними інтерфейсами взаємодіяти між собою. Він дозволяє одному інтерфейсу виглядати так, наче він відповідає іншому інтерфейсу. Цей шаблон використовується для перетворення інтерфейсів класів, які не можуть працювати разом через несумісні інтерфейси.

Приклад використання: адаптер між старим і новим API, щоб старий код міг використовувати нові функції без змін у базовому коді.

Builder

Шаблон Builder дозволяє створювати складні об'єкти поетапно. Він відокремлює створення об'єкта від його представлення, що дозволяє створювати різні представлення об'єкта за однаковим процесом побудови. Використовується для створення об'єктів з багатьма варіантами налаштувань.

Приклад використання: створення об'єкта з великою кількістю конфігурацій (наприклад, різні типи вікон з багатьма налаштуваннями).

Command

Шаблон Command інкапсулює запит як об'єкт, що дозволяє параметризувати об'єкти зокрема запитами, чергувати або логувати запити, а також підтримувати операції скасування. Це дозволяє передавати запити або операції безпосередньо в об'єкти.

Приклад використання: система відправки команд у графічних інтерфейсах, де кожна дія є об'єктом команди.

Chain of Responsibility

Шаблон Chain of Responsibility дозволяє обробляти запити по черзі декількома обробниками. Кожен обробник може обробити запит або передати його далі по ланцюгу. Це дозволяє розподіляти обробку запитів серед кількох обробників без явного зв'язку між ними.

Приклад використання: обробка запитів у веб-сервісах, де кожен компонент може додавати нову логіку до обробки запиту.

Prototype

Шаблон Prototype дозволяє створювати нові об'єкти шляхом копіювання існуючих об'єктів (прототипів), що дозволяє створювати нові екземпляри без необхідності викликати конструктор. Це часто використовується, коли створення об'єктів є складним або ресурсоємним.

Приклад використання: створення копій складних об'єктів без необхідності відтворювати всю логіку їхнього створення.

Крок 2: Реалізація шаблону “Prototype”

Даний шаблон дозволяє створювати копії об’єктів зі збереженням і зміною певних їх властивостей. У випадку системи “Особиста бухгалтерія” даний функціонал зручно використати для:

- створення нових транзакцій подібних до попередніх (зарплата, комунальні послуги і тд)
- створення таких самих періодичних транзакцій у випадку закінчення терміну попередніх (поновлення підписки тощо)
- створення кредитів з однаковим відсотком

Для реалізації даного шаблону нам потрібно додати реалізацію інтерфейсу Cloaneable до певних моделей, щоб відбувалося клонування об’єктів.

У класі Loan шаблон Prototype реалізується через метод clone(), який викликає метод клонування батьківського класу Account та додає специфічні зміни для об'єкта Loan:

1. Викликається метод super.clone(), щоб скопіювати значення полів батьківського класу.
2. Клонованому об'єкту надається новий стан — LoanActiveState(), щоб вказати, що новий кредит починає працювати з активним станом.
3. Поле id обнуляється, оскільки новий об'єкт не має ідентифікатора до збереження в базі даних.

Цей підхід дозволяє створювати нові екземпляри кредитів з уже налаштованими значеннями, не створюючи їх вручну, а клонируючи наявні, що особливо корисно при створенні аналогічних об'єктів з різними значеннями в межах кредитного процесу.

У класі Transaction шаблон Prototype реалізується через метод clone(), який створює копію поточної транзакції:

1. Викликається метод super.clone(), щоб створити копію поточного об'єкта.
2. id обнуляється, тому що це новий екземпляр, який ще не був збережений у базі.

3. Поле transactionDate змінюється, додаючи один місяць до поточної дати, щоб створити нову транзакцію для наступного місяця.
4. Якщо транзакція перевищує дату закінчення, метод повертає null, що означає, що транзакція більше не може бути створена.

```
    @Override  dashushak
    public Transaction clone() {
        try {
            Transaction clonedTransaction = (Transaction) super.clone();
            clonedTransaction.id = null;
            clonedTransaction.transactionDate = this.transactionDate.plusMonths( monthsToAdd: 1);
            if (this.endDate != null && clonedTransaction.transactionDate.isAfter(this.endDate)) {
                return null;
            }

            return clonedTransaction;
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException("Клонування не підтримується", e);
        }
    }

    public Transaction createNextPeriodicTransaction() {  no usages  dashushak
        if (!this.isPeriodic) {
            throw new IllegalStateException("Ця транзакція не є періодичною.");
        }
        if (this.endDate != null && this.transactionDate.plusMonths( monthsToAdd: 1).isAfter(this.endDate)) {
            throw new IllegalStateException("Транзакція перевищує дату закінчення.");
        }
        return this.clone();
    }
}
```

Рис. 1 - реалізація шаблону Prototype для класу Transaction

```
@Override  dashushak
public Loan clone() {
    try {
        Loan clonedLoan = (Loan) super.clone();
        clonedLoan.setState(new LoanActiveState());
        clonedLoan.setId(null);
        return clonedLoan;
    } catch (CloneNotSupportedException e) {
        e.printStackTrace();
        return null;
    }
}
```

Рис. 2 - реалізація шаблону Prototype для класу Loan

Проєкт можна переглянути у репозиторії:
https://github.com/papiroskada/budget-mate

Висновок: у рамках виконання лабораторної роботи щодо проектування системи "Особиста бухгалтерія" було здійснено реалізацію шаблону проектування Prototype для класів Loan, Transaction. Шаблон Prototype дає можливість ефективно працювати з об'єктами, що часто змінюються, та знижує складність коду в разі необхідності повторного створення об'єктів із схожими властивостями.