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», «CHAINOF RESPONSIBILITY», «PROTOTYPE»»

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 **Adapter** використовується для узгодження двох несумісних інтерфейсів. Він діє як проміжний клас, перетворюючи інтерфейс одного класу у вигляд, зрозумілий іншому класу. Це дозволяє класам з різними інтерфейсами працювати разом, не змінюючи їхнього коду.

Builder призначений для пошагового створення складних об'єктів. Він забезпечує можливість створення об'єктів із різними конфігураціями, приховуючи логіку створення. Завдяки цьому шаблону можна уникнути створення конструктора з багатьма параметрами.

Command дозволяє інкапсулювати запити, дії чи операції в об'єкти, надаючи можливість відкладеного виконання, зберігання історії операцій чи скасування виконаних дій. Це досягається шляхом створення класу команди, який містить методи виконання операції.

Chain of Responsibility організовує обробку запитів у ланцюг об'єктів, кожен із яких вирішує, чи може він обробити запит, чи передати його наступному об'єкту в ланцюгу. Це зменшує залежність між відправником і отримувачем запиту, забезпечуючи гнучкість у додаванні нових ланок.

Шаблон **Prototype** дозволяє створювати нові об'єкти шляхом копіювання існуючих, використовуючи метод клонування. Це корисно, коли створення об'єктів є дорогим або складним процесом, і значно спрощує створення подібних об'єктів.

Крок 2. Реалізація шаблону “Prototype”

Для реалізацібю шаблону проєктування "Prototype" було обрано категорії витрат (ExpenseCategory) в системі "Особиста бухгалтерія". Цей шаблон дозволяє створювати нові об'єкти категорій на основі вже існуючих шляхом клонування. Це корисно для повторного використання властивостей базових категорій, таких як назва чи опис, що значно спрощує процес адміністрування системи.

Модель ExpenseCategory

Було додано клас ExpenseCategory, який реалізує інтерфейс Cloneable. Він містить властивість name (назва категорії). Для реалізації клонування в цьому класі використовується метод clone, який дозволяє створити точну копію об'єкта з можливістю модифікації його параметрів перед збереженням у базі даних.

```
@Entity(name = "expense_categories")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ExpenseCategory implements Cloneable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    @Column(nullable = false, updatable = false)
    private Instant createdAt = Instant.now();
    new *
    @Override
    public ExpenseCategory clone() {
        try {
            return (ExpenseCategory) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException("Cloning not supported for ExpenseCategory", e);
        }
    }
}
```

Рис 1. Модель ExpenseCategory

Сервіс ExpenseCategoryPrototypeService

Створено спеціальний сервіс, що відповідає за операції клонування категорій. Метод cloneCategory у цьому сервісі:

- Знаходить оригінальну категорію за її ID.
- Клонує категорію, створюючи новий об'єкт без ID.
- Модифікує нову категорію, задаючи їй унікальну назву.
- Зберігає клоновану категорію у базі даних.

```java
@Service
@RequiredArgsConstructor
public class ExpenseCategoryService {
    private final ExpenseCategoryRepository expenseCategoryRepository;
    1 usage  new *
    public ExpenseCategory cloneCategory(Long categoryId, String newName) {
        ExpenseCategory originalCategory = expenseCategoryRepository.findById(categoryId)
                .orElseThrow(() -> new IllegalArgumentException("Category not found"));

        ExpenseCategory clonedCategory = originalCategory.clone();
        clonedCategory.setId(null);
        clonedCategory.setName(newName);

        return expenseCategoryRepository.save(clonedCategory);
    }
}
```

Рис 2. Сервіс ExpenseCategoryPrototypeService

Контролер ExprenseCategoryController

Для адміністратора було створено REST-ендпоінт admin/categories/clone, який дозволяє клонувати категорії. Адміністратор вказує ID існуючої категорії та назву нової, після чого система створює клон і зберігає його в базі даних.

```java
@RestController
@RequestMapping("/admin/categories")
@RequiredArgsConstructor
public class ExpenseCategoryController {
    private final ExpenseCategoryService expenseCategoryService;
    new *
    @PostMapping("/clone")
    public ResponseEntity<ExpenseCategory> cloneCategory(@RequestParam Long categoryId, @RequestParam String newName) {
        ExpenseCategory clonedCategory = expenseCategoryService.cloneCategory(categoryId, newName);
        return ResponseEntity.status(HttpStatus.CREATED).body(clonedCategory);
    }
}
```

Рис 3. Реалізація ExprenseCategoryController

Посилання на репозиторій проєкту

Висновок

У рамках лабораторної роботи було реалізовано функціонал для системи "Особиста бухгалтерія" з використанням шаблону "Prototype". Після ознайомлення з теоретичними відомостями про шаблони, було спроєктовано взаємодію класів для досягнення поставлених функціональних цілей.

Шаблон "Prototype" був застосований для управління категоріями витрат, дозволяючи створювати нові категорії на основі існуючих шляхом клонування. Це дало змогу значно спростити та прискорити процес додавання нових категорій, забезпечивши ефективне повторне використання існуючих даних.

Отриманий досвід під час виконання роботи показав, як шаблони проєктування можуть допомогти оптимізувати функціонал і підвищити гнучкість системи.