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 №6

із дисципліни *«Технології розроблення програмного забезпечення»*

Тема: « ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»»

Виконав:

Студент групи ІА-23 Хохол М.В.

Перевірив:

Мягкий М.Ю.

Київ 2024

Варіант №7

..7 Редактор зображень (state, prototype, memento, facade, composite, client-server)

Редактор зображень має такі функціональні можливості: відкриття/збереження зображень у найпопулярніших форматах (5 на вибір студента), застосування ефектів, наприклад поворот, розтягування, стиснення, кадрування зображення, можливість створення колажів шляхом «нашарування» зображень.

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.

2. Реалізувати частину функціоналу робочої програми у вигляді класів та їх взаємодій для досягнення конкретних функціональних можливостей.

3. Застосування одного з розглянутих шаблонів при реалізації програми.

Зміст

Хід роботи:

Крок 1. Короткі теоретичні відомості.

1. Abstract Factory Тип:

Створюючий паттерн.

Призначення: Дозволяє створювати родини пов’язаних об'єктів без прив’язки до їх конкретних класів.

Ключові моменти: Використовується, коли потрібно створювати кілька об’єктів, які мають працювати разом. Забезпечує гнучкість при зміні сімейств продуктів. Приклад: Інтерфейси для створення UI-елементів (кнопки, поля) для різних операційних систем (Windows, macOS).

2. Factory Method

Тип: Створюючий паттерн.

Призначення: Дозволяє створювати об’єкти, делегуючи цей процес підкласам.

Ключові моменти: Замість виклику конструктора напряму використовується метод фабрики. Підходить для створення об’єктів, тип яких визначається під час виконання. Приклад: Створення різних типів документів (PDF, Word) у текстовому редакторі.

3. Memento

Тип: Поведінковий паттерн.

Призначення: Зберігає та відновлює попередній стан об'єкта без порушення інкапсуляції.

Ключові моменти: Використовується для реалізації функціоналу Undo/Redo. Складається з об’єкта-опікуна (Caretaker), об’єкта-власника стану (Originator) та знімка стану (Memento). Приклад: Збереження стану текстового документа під час редагування.

4. Observer Тип:

Поведінковий паттерн.

Призначення: Дозволяє одному об’єкту (спостерігачу) автоматично отримувати сповіщення про зміни стану іншого об’єкта (суб’єкта).

Ключові моменти: Забезпечує зв’язок "один-до-багатьох". Використовується для оновлення інтерфейсів у реальному часі.

Приклад: Система підписок, де користувач отримує сповіщення про новини.

5. Decorator

Тип: Структурний паттерн.
Призначення: Дозволяє динамічно додавати нову поведінку об'єктам без зміни їхнього коду.

Ключові моменти: Використовує композицію замість успадкування. Декоратор обгортає основний об'єкт і додає нову функціональність.

Приклад: Додавання додаткових функцій до UI-компонентів, таких як прокрутка або межі.

6. SOLID

Це набір принципів об'єктно-орієнтованого програмування, спрямованих на створення зрозумілого, гнучкого та підтримуваного коду:

1. S – Single Responsibility Principle (Принцип єдиної відповідальності): Клас повинен мати тільки одну причину для зміни (одну відповідальність).
2. O – Open/Closed Principle (Принцип відкритості/закритості):

Клас має бути відкритим для розширення, але закритим для модифікації.

3. L – Liskov Substitution Principle (Принцип підстановки Лісков): Об'єкти підкласу повинні замінюватися об'єктами батьківського класу без порушення логіки програми.
4. I – Interface Segregation Principle (Принцип розділення інтерфейсів): Краще мати кілька специфічних інтерфейсів, ніж один великий.
5. D – Dependency Inversion Principle (Принцип інверсії залежностей): Модулі високого рівня не повинні залежати від модулів низького рівня. Обидва мають залежати від абстракцій.

Крок 2. Реалізація шаблону Memento: 1.

Клас ProjectSnapshot

```
13 usages
public class ProjectSnapshot {
    1 usage
    private final Long projectId;
    2 usages
    private final String name;
    2 usages
    private final Boolean editingEnabled;
    2 usages
    private final Boolean downloadEnabled;
    3 usages
    private final Set<Collage> collages;

    2 usages
    private final List<Image> images;

    1 usage
    public ProjectSnapshot(Project project) {
        this.projectId = project.getId();
        this.name = project.getName();
        this.editingEnabled = project.getEditingEnabled();
        this.downloadEnabled = project.getDownloadEnabled();

        this.collages = project.getCollages().stream()
                .map(collage -> {
                    Collage copiedCollage = collage.copy();
                    copiedCollage.setCells(
                            collage.getCells().stream()
                            .map(cell -> {
                                Cell copiedCell = cell.copy();
                                if (cell.getImage() != null) {
                                    copiedCell.setImage(cell.getImage().copy());
                                }
                                return copiedCell;
                            })
                            .collect(Collectors.toSet())
                    );
                    return copiedCollage;
                })
                .collect(Collectors.toSet());


        this.images = project.getImages().stream()
                .map(Image::copy)
                .collect(Collectors.toList());
    }
```

```
2 usages
public void restore(Project project) {
    project.setName(this.name);
    project.setEditingEnabled(this.editingEnabled);
    project.setDownloadEnabled(this.downloadEnabled);
    project.setCollages(this.collages);
    project.setImages(this.images);


    this.collages.forEach(collage ->
            collage.getCells().forEach(cell -> {
                cell.setCollage(collage);
                if (cell.getImage() != null) {
                    cell.getImage().setProject(project);
                }
            })
    );
}
```

Рис 1. Клас ProjectSnapshot

ProjectSnapshot (Мементо): Клас ProjectSnapshot зберігає стан об'єкта Project. Це знімок, який містить необхідні атрибути проекту: його ідентифікатор (projectId), назву (name), статуси (editingEnabled, downloadEnabled), набір колажів (collages), а також список зображень (images). **Конструктор ProjectSnapshot(Project project)**: У цьому методі створюється глибока копія проекту. Копіюються всі колажі проекту та їхні клітинки, а також усі зображення. Кожна клітинка має копію посилань на колаж та, якщо є, на зображення. Це забезпечує ізоляцію даних між знімком та оригінальним проектом. **Метод restore(Project project)**: Цей метод відновлює стан проекту зі знімка. Він присвоює проекту збережені в знімку значення атрибутів та налаштовує зворотні посилання між клітинками, колажами та зображеннями, щоб вони знову коректно працювали в контексті проекту.

2. **Метод saveSnaphots в класі ProjectService**

```java
@Transactional
@Scheduled(fixedRate = 60_000)
public void saveSnapshots() {
    List<Project> projects = projectRepository.findAllWithDetails();

    for (Project project : projects) {
        try {
            project.getCollages().forEach(collage -> {
                collage.getCells().forEach(cell -> {
                    if (cell.getImage() != null) {
                        cell.getImage().getName();
                    }
                });
            });
            snapshotService.saveSnapshot(project);
            projectRepository.save(project);
        } catch (Exception e) {
            System.err.println("Error processing project: " + project.getId());
            e.printStackTrace();
        }
    }
}
```

Рис 2. Метод saveSnaphots в класі ProjectService

Метод saveSnapshots періодично (кожні 60 секунд) викликається для створення знімків стану проектів. Анотація @Transactional забезпечує виконання всієї операції в межах однієї транзакції, а @Scheduled задає інтервал запуску.
Метод завантажує всі проекти разом зі зв'язаними об'єктами (колажі, клітинки, зображення) за допомогою findAllWithDetails(), який попередньо завантажує дані через @EntityGraph. Для кожного проекту:

1. Перевіряються зв’язки між об’єктами.
2. Викликається snapshotService.saveSnapshot() для створення знімка стану.
3. Проект повторно зберігається в базі (projectRepository.save()).

Блок try-catch обробляє винятки, щоб забезпечити стабільність виконання навіть при збої одного з проектів.

3. **Метод restoreProjectToSnapshot в класі ProjectService**

```
1 usage
public void restoreProjectToSnapshot(Long projectId, int snapshotIndex) {
    ProjectSnapshot snapshot = snapshotService.getSnapshot(projectId, snapshotIndex);
    Project project = projectRepository.findById(projectId)
            .orElseThrow(() -> new RuntimeException("Project not found"));
    snapshot.restore(project);

    projectRepository.save(project);
}
```

Рис 3. Метод restoreProjectToSnapshot в класі ProjectService

restoreProjectToSnapshot - метод який використовується для відновлення стану проекту до певного знімка (snapshot). snapshotService.getSnapshot(projectId, snapshotIndex): Отримує збережений знімок (ProjectSnapshot) для проекту з ідентифікатором projectId і індексом snapshotIndex. Якщо знімок існує, він повертається для подальшого використання. projectRepository.findById(projectId): Шукає проект у базі даних за ідентифікатором projectId. Якщо проект не знайдено, метод викидає RuntimeException з повідомленням "Project not found". snapshot.restore(project): Викликає метод restore у знімку, який змінює об'єкт проекту (project), повертаючи його до стану, збереженого в знімку. projectRepository.save(project): Зберігає оновлений об'єкт проекту в базі даних, щоб усі зміни набули чинності.

4**. Інтерфейс ProjectRepository**

```
2 usages
@Repository
public interface ProjectRepository extends JpaRepository<Project, Long> {

    1 usage
    List<Project> findByUserId(Long userId);


    1 usage
    @EntityGraph(attributePaths = {"collages.cells.image"})
    @Query("SELECT p FROM Project p")
    List<Project> findAllWithDetails();
}
```

Рис 4. Інтерфейс ProjectRepository

@EntityGraph(attributePaths = {"collages.cells.image"}): Анотація задає граф вибірки (fetch graph), що дозволяє явно завантажувати певні пов'язані сутності. У цьому випадку під час вибірки проектів також завантажуються зв'язані collages, їхні cells і відповідні image. @Query("SELECT p FROM Project p"): Явно задає JPQL-запит для вибірки всіх проектів з бази даних. Використовується разом із @EntityGraph для вибірки зв'язаних даних. findAllWithDetails(): Метод, що використовує вищезгаданий граф вибірки і запит, щоб завантажити всі проекти з усіма необхідними пов'язаними сутностями (collages, cells, image). Це дозволяє уникнути проблеми лінивого завантаження (LazyInitializationException) під час доступу до зв'язаних об'єктів поза межами транзакції.

5. **Метод createSnapshot в класі Project**

```
1 usage
public ProjectSnapshot createSnapshot() {

    return new ProjectSnapshot(this);
}
```

Рис 5. Метод createSnapshot в класі Project

Цей метод реалізує створення знімка (snapshot) в рамках патерну "Memento". Основна ідея полягає в тому, щоб зафіксувати поточний стан об'єкта Project, зберігши його у

вигляді окремого об'єкта ProjectSnapshot. Це дозволяє згодом відновити стан об'єкта до цього моменту часу. return new ProjectSnapshot(this);: У цьому рядку створюється новий екземпляр класу ProjectSnapshot, передаючи поточний об'єкт (this) як аргумент в конструктор. Це дозволяє передати весь поточний стан об'єкта Project у знімок.

6. Метод restoreProject в ProjectController

```
@PostMapping("/{projectId}/restore/{snapshotIndex}")
public ResponseEntity<Void> restoreProject(@PathVariable Long projectId, @PathVariable int snapshotIndex) {
    try {
        projectService.restoreProjectToSnapshot(projectId, snapshotIndex);
        return ResponseEntity.ok().build();
    } catch (IllegalArgumentException e) {
        return ResponseEntity.badRequest().build();
    }
}
```

Рис 6. Метод restoreProject в ProjectController

Цей метод який допомагає відновити стан проекту до конкретного збереженого знімка. Він приймає ідентифікатор проекту (projectId) та індекс знімка (snapshotIndex) через URL, передає їх у відповідний сервіс і забезпечує обробку можливих помилок, повертаючи відповідний HTTP-статус. projectService.restoreProjectToSnapshot(projectId, snapshotIndex);: Викликає метод сервісу restoreProjectToSnapshot, який реалізує відновлення проекту до певного збереженого стану на основі projectId і snapshotIndex. return ResponseEntity.ok().build();: Якщо операція успішна, метод повертає HTTPвідповідь зі статусом 200 OK. catch (IllegalArgumentException e): Якщо метод restoreProjectToSnapshot генерує виняток IllegalArgumentException (наприклад, через неправильний індекс знімка або відсутність проекту), виконання передається в блок catch. return ResponseEntity.badRequest().build();: У разі помилки метод повертає HTTP-відповідь зі статусом 400 Bad Request, сигналізуючи про некоректний запит.

Детальніше код можна переглянути в репозиторії проекту:

https://github.com/Maxim-Khokhol/image-editor

Висновок: В цій лабораторній роботі я познайомився з паттернами Abstract Factory, Factory Method, Memento, Observer, Decorator. Було програмно реалізовано паттерн **Memento**, що дозволило впровадити динамічне управління станами об'єкта Project, розділивши логіку на окремі класи та підвищивши гнучкість та підтримуваність системи.