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: « «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR»**»**

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) **Flyweight** (Легковаговий) – це структурний патерн, який дозволяє зменшити споживання пам’яті шляхом спільного використання однакових об’єктів. Замість створення безлічі однакових екземплярів, він зберігає їх у пулі й видає за запитом. Об’єкти розділяються на внутрішній стан (спільний між усіма екземплярами) і зовнішній стан (унікальний для кожного). Наприклад, у графічному редакторі можна використовувати один об’єкт для відображення однакових символів шрифту.
2) **Composite** (Композиція) – це структурний патерн, що дозволяє працювати з деревоподібними структурами даних однаково, незалежно від того, чи це окремий елемент, чи група елементів. Клієнтський код не відрізняє індивідуальні об'єкти від контейнерів. Наприклад, у файловій системі файл і папка можуть бути представлені загальним інтерфейсом, а операції застосовуються однаково до обох.
3) **Visitor** (Відвідувач) – це поведінковий патерн, який дозволяє додавати нові операції до групи об’єктів без зміни їхніх класів. Відвідувач визначає окремий об’єкт, що містить логіку операцій, а класи, які обробляються, приймають цього відвідувача. Це корисно для виконання різних обробок на структурах об’єктів. Наприклад, для серіалізації або підрахунку різних типів об’єктів у структурі.
4) **Interpreter** (Інтерпретатор) – це поведінковий патерн, який дозволяє реалізувати мову або граматику для обчислення чи інтерпретації виразів. Він визначає клас для кожного символу граматики й використовує композицію цих класів для обчислення. Наприклад, для роботи з математичними виразами (парсинг і обчислення) чи SQL-запитами.

Крок 2. Реалізація шаблону Composite:

1) Інтерфейс Controllable

```
4 usages   2 implementations
public interface Controllable {
    3 usages   2 implementations
    void scale(double scaleFactor);
    3 usages   2 implementations
    void move(int step, String action);
    1 usage   2 implementations
    void addImage(Image image);
    no usages   2 implementations
    void removeImage(Image image);

    1 usage   2 implementations
    Image getChild(int index);
}
```

Рис 1. Реалізація інтерфейсу Controllable

Інтерфейс Controllable визначає набір методів, які дозволяють керувати об'єктами в рамках патерну Composite. Метод scale(double scaleFactor) відповідає за зміну розміру об'єкта з використанням переданого коефіцієнта масштабування. Метод move(int step, String action) забезпечує переміщення об'єкта, де параметр step визначає величину зсуву, а action вказує напрямок, наприклад "right", "left", "top" або "bottom". Метод addImage(Image image) дозволяє додавати дочірні елементи типу Image до контейнера, тоді як метод removeImage(Image image) використовується для їх видалення. Метод getChild(int index) дозволяє отримати дочірній елемент за індексом. Ці методи створюють єдиний інтерфейс для роботи з об'єктами, незалежно від того, чи є вони листами (наприклад, клас Image), чи контейнерами (наприклад, клас Collage).

2) Імплементація Controllable в класі Collage

```java
@Override
public void addImage(Image image) {
    if (images == null) {
        images = new ArrayList<>();
    }
    images.add(image);

}
```

no usages

```java
@Override
public void removeImage(Image image) {
    if (images != null) {
        images.remove(image);
    }
}
```

3 usages

```java
@Override
public void scale(double scaleFactor) {
    this.width = (int) (this.width * scaleFactor);
    this.height = (int) (this.height * scaleFactor);

    for (Image image : images) {
        image.scale(scaleFactor);
    }
}
```

3 usages

```java
@Override
public void move(int step, String action){
    switch (action) {
        case "right" -> this.left -= step;
        case "left" -> this.left += step;
        case "top" -> this.top += step;
        case "bottom" -> this.top -= step;
        default ->
                throw new UnsupportedOperationException("Unsupported operation, use only 'right', 'left', 'top', 'bottom'
    }
    for (int i = 0; i < images.size(); i++){
        getChild(i).move(step, action);
    }
}
```

1 usage

```java
@Override
public Image getChild(int index){
    if (index < 0 || index >= images.size()) {
        throw new IndexOutOfBoundsException("Index " + index + " is out of bounds.");
    }
    return images.get(index);
}
```

Рис 2. Імплементація Controllable в класі Collage

Реалізація 5 методів з класу Collage, які реалізують інтерфейс Controllable. Метод addImage(Image image) перевіряє, чи колекція images ініціалізована. Якщо ні, створюється новий список, після чого додається переданий об'єкт Image. Метод removeImage(Image image) видаляє переданий об'єкт Image з колекції, якщо вона не порожня. Метод scale(double scaleFactor) змінює розміри об'єкта Collage пропорційно переданому коефіцієнту масштабування. Після цього метод рекурсивно викликається для кожного зображення у списку images, забезпечуючи масштабування всіх дочірніх елементів. Ці методи демонструють використання Composite, дозволяючи об'єктам контейнера керувати своїми підлеглими.

3) Імплементація Controllable в класі Image

```
@Override
public void scale(double scaleFactor) {
    this.width = (int) (this.width * scaleFactor);
    this.height = (int) (this.height * scaleFactor);
}

3 usages
@Override
public void move(int step, String action){
    switch (action) {
        case "right" -> this.left -= step;
        case "left" -> this.left += step;
        case "top" -> this.top += step;
        case "bottom" -> this.top -= step;
        default ->
                throw new UnsupportedOperationException("Unsupported operation, use only 'right', 'left', 'top', 'bottom' ");
    }
}
1 usage
@Override
public Image getChild(int index) { throw new UnsupportedOperationException("Cannot get child from a leaf"); }
1 usage
@Override
public void addImage(Image image) { throw new UnsupportedOperationException("Cannot add an image to a leaf"); }
no usages
@Override
public void removeImage(Image image) {
    throw new UnsupportedOperationException("Cannot remove an image from a leaf");
}
```

Рис 3. Імплементація Controllable в класі Image

Реалізація методів інтерфейсу Controllable для класу Image, який є листом у структурі Composite. Метод scale(double scaleFactor) змінює розміри об'єкта Image відповідно до заданого коефіцієнта масштабування, модифікуючи ширину (width) і висоту (height).

Метод move(int step, String action) переміщує зображення, змінюючи його координати left і top залежно від напряму (action), який може бути "right", "left", "top" або "bottom". Якщо напрямок не відповідає підтримуваним значенням, викликається виключення UnsupportedOperationException.

Методи getChild(int index), addImage(Image image) і removeImage(Image image) викидають виключення UnsupportedOperationException із відповідними повідомленнями. Це підкреслює, що об'єкти типу Image є листами і не можуть мати дочірніх елементів чи взаємодіяти з іншими елементами як контейнер. Така реалізація забезпечує чітке розділення відповідальностей між листами та контейнерами у структурі Composite.

4) Class ProjectService

```
public void scaleBlock(Long projectId, Long collageId, double scaleFactor) {
    Collage collage = collageRepository.findByPersonalId(collageId)
            .orElseThrow(() -> new RuntimeException("Collage not found"));
    collage.scale(scaleFactor);
    collageRepository.save(collage);
}

1 usage
public void scaleImage(Long projectId,Long imageId, double scaleFactor) {
    Image image = imageRepository.findByPersonalId(imageId)
            .orElseThrow(() -> new RuntimeException("Image not found"));
    image.scale(scaleFactor);
    imageRepository.save(image);

}

1 usage
public void moveBlock(Long projectId, Long collageId, int step, String action){
    Collage collage = collageRepository.findByPersonalId(collageId)
            .orElseThrow(() -> new RuntimeException("Collage not found"));
    collage.move(step, action);
    collageRepository.save(collage);

}

1 usage
public void moveImage(Long projectId, Long imageId, int step, String action){
    Image image = imageRepository.findByPersonalId(imageId)
            .orElseThrow(() -> new RuntimeException("Image not found"));
    image.move(step, action);
    imageRepository.save(image);
}
```

Рис 4. Логіка в ProjectService

На зображенні представлено чотири методи з класу ProjectService, які відповідають за масштабування та переміщення об'єктів у проекті. Методи scaleBlock і moveBlock працюють з об'єктами типу Collage, які є контейнерами в структурі Composite. У кожному з цих методів спочатку отримується об'єкт Collage з репозиторію за його ідентифікатором collageId. Якщо об'єкт не знайдено, викидається виключення RuntimeException. Потім викликаються методи scale або move відповідно, які реалізовані в класі Collage і виконують дію як для самого колажу, так і для його дочірніх елементів (зображень). Після виконання змін об'єкт зберігається в репозиторії.

Методи scaleImage і moveImage працюють з об'єктами типу Image, які є листами в структурі Composite. Аналогічно, об'єкт отримується з репозиторію за ідентифікатором imageId, і якщо об'єкт не знайдено, викидається виключення. Далі викликаються методи scale або move, реалізовані в класі Image, які застосовують зміни тільки до самого зображення. Після цього зміни зберігаються в репозиторії.

Ці методи підкреслюють гнучкість Composite, дозволяючи однаковим чином масштабувати чи переміщати як контейнери (колажі), так і листи (зображення), з використанням загальних операцій.

5) Class ProjectController

```java
@PutMapping("/{projectId}/scale")
public ResponseEntity<Void> scaleComponent(
        @PathVariable Long projectId,
        @RequestParam(required = false) Long collageId,
        @RequestParam(required = false) Long imageId,
        @RequestParam double scaleFactor,
        @RequestParam String selectionMode) {

    if(Objects.equals(selectionMode, b: "block")){
        projectService.scaleBlock(projectId, collageId, scaleFactor);
    } else {
        projectService.scaleImage(projectId, imageId, scaleFactor);
    }
    return ResponseEntity.ok().build();
}

@PutMapping("/{projectId}/move")
public ResponseEntity<Void> moveComponent(
        @PathVariable Long projectId,
        @RequestParam(required = false) Long collageId,
        @RequestParam(required = false) Long imageId,
        @RequestParam int step,
        @RequestParam String selectionMode,
        @RequestParam String action) {

    if(Objects.equals(selectionMode, b: "block")){
        projectService.moveBlock(projectId, collageId, step, action);
    } else {
        projectService.moveImage(projectId, imageId, step, action);
    }
    return ResponseEntity.ok().build();
}
```

rc > main > java > com > prolmg > image_editor > controllers > ProjectController

Рис 5. ProjectController

Два методи з класу ProjectController, які забезпечують масштабування та переміщення компонентів проекту через HTTP-запити. Метод scaleComponent приймає параметри: projectId (ідентифікатор проекту), collageId (ідентифікатор колажу), imageId (ідентифікатор зображення), scaleFactor (коефіцієнт масштабування) та selectionMode (режим вибору: "block" або інше). Якщо режим вибору дорівнює "block", викликається

метод scaleBlock сервісу ProjectService для масштабування колажу. В іншому випадку викликається метод scaleImage для масштабування окремого зображення. В кінці метод повертає HTTP-відповідь зі статусом 200 OK. Метод moveComponent працює аналогічно, але замість масштабування забезпечує переміщення компонентів. Він приймає параметри projectId, collageId, imageId, step (крок переміщення), selectionMode (режим вибору) та action (напрямок переміщення, наприклад, "left" чи "right"). Залежно від режиму вибору, викликається або moveBlock для переміщення колажу, або moveImage для переміщення зображення. Після виконання дій метод повертає HTTP-відповідь зі статусом 200 OK. Ці методи забезпечують єдиний інтерфейс для роботи з компонентами Composite (колажами та зображеннями) через REST API, підвищуючи гнучкість і масштабованість системи.

Висновок: В цій лабораторній роботі я познайомився з паттернами «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR» Було програмно реалізовано паттерн **Composite**. Патерн Composite дозволив уніфікувати роботу з різними типами об'єктів, такими як колажі (Collage) і зображення (Image), за допомогою єдиного інтерфейсу Controllable. Це забезпечило можливість виконання однакових операцій, наприклад, масштабування чи переміщення, незалежно від типу об'єкта. Рекурсивна обробка стала ключовою перевагою, оскільки дозволила реалізувати операції, які автоматично поширюються на всі дочірні елементи, наприклад, при зміні масштабу колажу всі зображення, які до нього належать, також масштабуються. Патерн спростив динамічне управління структурою, дозволяючи додавати або видаляти зображення з колажу в будь-який момент. Крім того, Composite забезпечив можливість гнучкого розширення, адже можна легко додати нові типи об'єктів або рівні ієрархії без значних змін у вже існуючому коді.