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

ШАБЛОНИ «COMPOSITE», «FLYWEIGHT»,
«INTERPRETER», «VISITOR»

Варіант 4

Виконав
студент групи ІА – 13:
Запотоцький І.А

Перевірив:
М’який М.Ю

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.

Варіант:

4)
Графічний редактор (proxy, prototype, decorator, bridge, flyweight, SOA) Графічний редактор повинен вміти створювати / редагувати растрові (або векторні на розсуд студента) зображення в 2-3 основних популярних форматах (bmp, png, jpg), мати панель інструментів для створення графічних примітивів, вибору кольорів, нанесення тексту, додавання найпростіших візуальних ефектів (ч/б растр, інфрачервоний растр, 2-3 на вибір учня), роботи з шарами.

Хід роботи

Шаблон проектування "Flyweight" (Легковага) відноситься до структурних шаблонів і використовується для ефективної підтримки багатьох малих об'єктів. Він передбачає розділення об'єкта на дві частини: внутрішню (інтрансивну), яка містить незмінні дані, і зовнішню (екстринсивну), яка зберігає змінюючіся дані.

Загальна ідея полягає в тому, щоб виділити спільні частини об'єктів і забезпечити відмінність лише для тих, що змінюються. Це дозволяє економити пам'ять і підвищувати продуктивність, особливо коли існує велика кількість схожих об'єктів.

Важливі елементи шаблону:

Легковага (Flyweight): Інтерфейс, який визначає операцію, яку можуть виконувати легковажі.

Конкретний легковага (Concrete Flyweight): Реалізація легковаги, яка містить інтрансивні дані.

Фабрика легковажів (Flyweight Factory): Відповідає за створення і управління легковажами, забезпечує використання вже існуючих легковажів або створення нових.

Клієнт: Використовує легковажі для виконання операцій з екстринсивними (змінюючимися) даними.

Цей шаблон допомагає оптимізувати використання пам'яті та покращувати продуктивність, зокрема в сценаріях, де існує велика кількість схожих об'єктів.

В моєму випадку шаблон використовується для обробки об’єктів Image, які поділяються на 2 Image та ImageType: в першому зберігається часто змінна

частина об’єкту а в другому – незмінні або дуже рідко змінні значення.
Таким чином за допомоги об’єкту фабрики ми створюємо цілий об’єкт картинки в пам’яті додатка – та, іноді, можемо використовувати його не роблячи затратний ресурсами виклик до бази даних:

```java
@Data
@NoArgsConstructor
@Getter
@Setter
@Entity
@Table(name = "images")
public class Image {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private Long views;
    @Embedded
    private ImageType imageType;
}
```

```java
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Embeddable

public class ImageType implements Comparable<ImageType>{

    private String code;

    private Long userId;

    private String path;

    private String extension;


    @Override
    public int compareTo(ImageType o) {
        return Objects.equals(this.getCode(), o.getCode()) ? 0 : this.getCode().hashCode() - o.getCode().hashCode();
    }

}
```

```java
3 usages
public class ImageFactory {
    3 usages
    private static final Map<String,ImageType> imageTypes = new HashMap<>();

    2 usages
    public static ImageType getImageType(String code, Long userId,String path,String extension){
        if(imageTypes.get(code) == null){
            imageTypes.put(code,new ImageType(code,userId,path,extension));
        }
        return imageTypes.get(code);
    }
}
```

В відповідному сервісі методи спочатку шукають об’єкт у Фабриці, а лише потім не знайшовши його роблять запрос до бд

```java
@Service
public class ImageServiceImpl implements ImageService{

    9 usages
    private final ImageRepository imageRepository;
    6 usages
    private final ImageTypeRepository imageTypeRepository;

    @Autowired
    public ImageServiceImpl(ImageRepository repo, ImageTypeRepository imageTypeRepository){
        this.imageRepository = repo;
        this.imageTypeRepository = imageTypeRepository;
    }

    1 usage
    @Override
    public ResponseEntity<List<Image>> findAllImages() {
        return new ResponseEntity<>(imageRepository.findAll(), HttpStatus.OK);
    }

    1 usage
    @Override
    public ResponseEntity<?> createImage(Image image) {
        this.imageRepository.save(image);
     ImageType storedImageType  =  ImageFactory.getImageType(image.getImageType().getCode(),image.getImageType().getUserId(),
             image.getImageType().getPath(),image.getImageType().getExtension());

        return new ResponseEntity<>(storedImageType,HttpStatus.CREATED);
    }
```

```
1 usage
@Override
public ResponseEntity<?> updateImage(Image image, String code) {
    if(imageTypeRepository.findById(code).isPresent()) {
        ImageType imageType = ImageFactory.getImageType(image.getImageType().getCode(), image.getImageType().getUserId(),
                image.getImageType().getPath(), image.getImageType().getExtension());
        if (imageRepository.findByImageType(imageType).isPresent()) {
            Image imageToChange = imageRepository.findByImageType(imageType).get();
            imageToChange.setImageType(imageType);
            imageToChange.setName(image.getName());
            imageToChange.setViews(image.getViews());
            imageRepository.save(imageToChange);

            return new ResponseEntity<>(imageToChange, HttpStatus.OK);
        }
    }
    return new ResponseEntity<>( body: "Image with this code does not exist",HttpStatus.BAD_REQUEST);
}
```

```
1 usage
@Override
public ResponseEntity<?> getImage(String code) {
if(imageTypeRepository.findById(code).isPresent()){
    ImageType imageType = imageTypeRepository.findById(code).get();
        return new ResponseEntity<>(imageRepository.findByImageType(imageType).get(), HttpStatus.OK);

}
    return new ResponseEntity<>( body: "Image with this code does not exist",HttpStatus.BAD_REQUEST);
}
```

```
1 usage
@Override
public ResponseEntity<?> deleteImage(String code) {
    if(imageTypeRepository.findById(code).isPresent()) {
        ImageType imageType = imageTypeRepository.findById(code).get();
            imageRepository.delete(imageRepository.findByImageType(imageType).get());

        return new ResponseEntity<>(HttpStatus.OK);
    }
```

```
    return new ResponseEntity<>( body: "Image with this code does not exist",HttpStatus.BAD_REQUEST);
}
```