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 №7

ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE»,
«TEMPLATE METHOD»

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 "Міст" (Bridge) належить до категорії структурних шаблонів. Він використовується для відокремлення абстракції від реалізації так, щоб обидві частини могли змінюватися незалежно одна від одної. Це забезпечує гнучкість та розширюваність коду.

Основні учасники шаблону "Міст":

Абстракція (Abstraction): Визначає високорівневий інтерфейс та утримує посилання на об'єкт реалізації.

Реалізація (Implementation): Визначає інтерфейс для реалізації об'єкта.

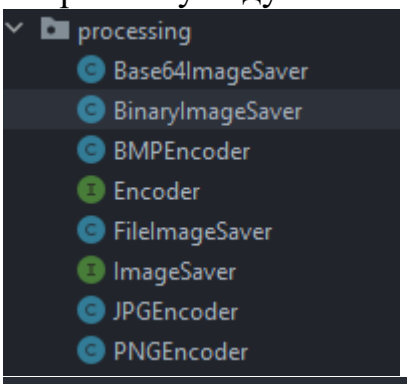
Розширена абстракція (Refined Abstraction): Розширює функціональність абстракції та може мати власний стан, який не залежить від реалізації.

Конкретна реалізація (Concrete Implementation): Реалізує інтерфейс реалізації та надає конкретну реалізацію.

Шаблон "Міст" використовує принцип композиції замість наслідування для розширення функціональності. Це дозволяє мати дві незалежні ієрархії класів: одну для абстракції та іншу для реалізації. Обидві ієрархії можуть змінюватись незалежно одна від одної.

В моєму випадку шаблон використовується наступним чином – об`єкт Image зберігається вибраним кодуванням – byteArray,file,base64
хоча приходить він завжди ук вигляді base64.
Цим займається інтерфейс Encoder та його реалізації,який переводить строку у об'єкт Зображення.
Потім, інтерфейс ImageSaver та його реалізації приймають у конструктор об’єкт Encoder та мають метод save який зберігає фото у вибраному форматі у

вибранному кодуванні.




```java
8 usages  3 implementations
public interface Encoder {

    1 usage  3 implementations
    ImageWithExtension encode(String base64) throws IOException;

}
```

```java
public interface ImageSaver {

    1 usage  3 implementations
    void save(ImageWithExtension image) throws IOException;

    1 usage  3 implementations
    Encoder getEncoder();
}
```

```java
1 usage
public class JPGEncoder implements Encoder {

    1 usage
    @Override
    public ImageWithExtension encode(String base64) throws IOException {

        byte[] imageBytes = Base64.getDecoder().decode(base64);
        ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(imageBytes);
        BufferedImage bufferedImage = ImageIO.read(byteArrayInputStream);
        byteArrayInputStream.close();
        return new ImageWithExtension(bufferedImage, extension: "jpg");

    }
```

```java
public class PNGEncoder implements Encoder{
    1 usage
    @Override
    public ImageWithExtension encode(String base64) throws IOException {

        byte[] imageBytes = Base64.getDecoder().decode(base64);
        ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(imageBytes);
        BufferedImage bufferedImage = ImageIO.read(byteArrayInputStream);
        byteArrayInputStream.close();
        return new ImageWithExtension(bufferedImage, extension: "png");

    }
}
```

```java
    @Override
    public ImageWithExtension encode(String base64) throws IOException {

        byte[] imageBytes = Base64.getDecoder().decode(base64);
        ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(imageBytes);
        BufferedImage bufferedImage = ImageIO.read(byteArrayInputStream);
        byteArrayInputStream.close();
        return new ImageWithExtension(bufferedImage, extension: "bmp");

    }
```

```java
@Getter
@Setter
@AllArgsConstructor
public class BinaryImageSaver implements ImageSaver{

    private final Encoder encoder;
    1 usage
    private final ImageRepository imageRepository;
    1 usage
    @Override
    public void save(ImageWithExtension image) throws IOException {
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        ImageIO.write(image.getImage(),image.getExtension(),byteArrayOutputStream);
        imageRepository.save(new ImageModel(Arrays.toString(byteArrayOutputStream.toByteArray()),image.getExtension(), savingType: "byteArray"));
    }
}
```

```java
@Getter
@Setter
@AllArgsConstructor
public class Base64ImageSaver implements ImageSaver {

    private final Encoder encoder;
    1 usage
    private final ImageRepository imageRepository;
    1 usage
    @Override
    public void save(ImageWithExtension image) throws IOException {
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        ImageIO.write(image.getImage(),image.getExtension(),byteArrayOutputStream);
        String base64value = Base64.getEncoder().encodeToString(byteArrayOutputStream.toByteArray());
        imageRepository.save(new ImageModel(base64value,image.getExtension(), savingType: "base64"));
    }
}
```

```java
@Getter
@Setter
@AllArgsConstructor
public class FileImageSaver implements ImageSaver{
    private final Encoder encoder;

    1 usage
    private final ImageRepository imageRepository;
    1 usage
    @Override
    public void save(ImageWithExtension image) throws IOException {
        saveRenderedImage(image.getImage(),image.getExtension(),image.getImage().toString().substring(0,8));

        imageRepository.save(new ImageModel( value: "images\\"+image.getImage().toString().substring(0,8),image.getExtension(), savingType: "file"));
    }

    1 usage
    private static void saveRenderedImage(RenderedImage renderedImage, String targetExtension, String outputFileName) {
        try {
            // Create the output file path
            String outputFilePath = outputFileName + "." + targetExtension;

            // Write the RenderedImage to a file
            ImageIO.write(renderedImage, targetExtension, new File(outputFilePath));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    }
```

Таким чином у контролері можуть використовуватися різні енкоудери у різних типах зберігання фото:

```java
    ImageSaver imageSaver;
    Encoder encoder;
    switch (req.getEncodeTo()) {
        case ("png") -> encoder = new PNGEncoder();
        case ("jpg") -> encoder = new JPGEncoder();
        case ("bmp") -> encoder = new BMPEncoder();
        default -> {
            return new ResponseEntity<>( body: "Provided extension's saving isn't supported.\n " +
                    "Supported extensions are: .png, .jpg, .bmp",HttpStatus.BAD_REQUEST);
        }
    }
    switch (req.getSavingType()){

        case ("base64") -> imageSaver = new Base64ImageSaver(encoder,imageRepository);
        case ("byteArray") -> imageSaver = new BinaryImageSaver(encoder,imageRepository);
        case ("file") -> imageSaver = new FileImageSaver(encoder,imageRepository);
        default -> {
            return new ResponseEntity<>( body: "Provided savingType isn't supported.\n " +
                    "Supported types are: base64, byteArray, file",HttpStatus.BAD_REQUEST);
        }
    }
```