Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ПРАКТИЧНА РОБОТА № 3

з дисципліни «Інженерія програмного забезпечення»
на тему «СТРУКТУРНІ ШАБЛОНИ ПРОЕКТУВАННЯ. ШАБЛОНИ COMPOSITE, DECORATOR, PROXY»

ВИКОНАВ:
студент ІІ курсу ФІОТ
групи ІО-32
Оніщенко Євгеній
Залікова № 3221

ПЕРЕВІРИВ:
Ст.викладач кафедри ОТ
Васильєва М.Д.

Київ – 2024

Завдання

Мета: Ознайомлення з видами шаблонів проектування ПЗ. Вивчення структурних шаблонів. Отримання базових навичок з застосування шаблонів Composite, Decorator та Proxy.

Варіант

Номер варіанту завдання обчислюється як залишок від ділення номеру залікової книжки на 12. 3221/12 = 5:

5. Розробити специфікації класів та реалізацію методів для представлення графічного елемента у редакторі векторної графіки, який дозволяє динамічно змінювати його відображення в залежності від різних параметрів. Наприклад, користувач може створити червоне коло, потім додати до нього рамку товщиною 2 пікселі, а потім додати тінь. При виклику draw() на фінальному об'єкті, система повинна спочатку відобразити тінь, потім рамку, а потім саме коло. Виклик getDescription() повинен повернути опис з усіма атрибутами: "Коло червоного кольору з рамкою товщиною 2 пікселі і тінню".

Код

```
/**
 * The Main class. It demonstrates
 * the use of decorators to modify the behavior of graphical objects.
 * @author Onischenko Yevhenii
 */
public class Main {
    /**
     * The main method where the program starts.
     * It creats a red circle and then adds border and shadow
     */
    public static void main(String[] args) {
        Graphic Circle = new Figure("червоне");

        Graphic CirleBorder = new BorderDecorator(Circle, 2);

        Graphic CirleBorderAndShadow = new ShadowDecorator(CirleBorder);

        CirleBorderAndShadow.draw();
        System.out.println(CirleBorderAndShadow.getDescription());
    }
}
```

```
/**
 * The Graphic interface represents objects that can be drawn and described.
 */
public interface Graphic {

    /**
     * Draws the graphic object.
     */
    public void draw();
    /**
     * Returns a description of the graphic object.
     *
     * @return a string describing the object
     */
    String getDescription();
}
```

```java
/**
 * The Decorator class is an abstract class that adds extra features to a
Graphic object.
 * It passes the work of drawing and getting descriptions to the object it
decorates.
 */
abstract public class Decorator implements Graphic {

    /**
     * The Graphic object being decorated.
     */
    protected Graphic object;

    /**
     * Creates a Decorator for the given Graphic object.
     *
     * @param object the Graphic object to be decorated
     */
    public Decorator(Graphic object) {
        this.object = object;
    }

    /**
     * Calls the draw method of the decorated object.
     */
    @Override
    public void draw() {
        object.draw();
    }

    /**
     * Returns the description from the decorated object.
     *
     * @return a string describing the object
     */
    @Override
    public String getDescription() {
        return object.getDescription();
    }
}
```

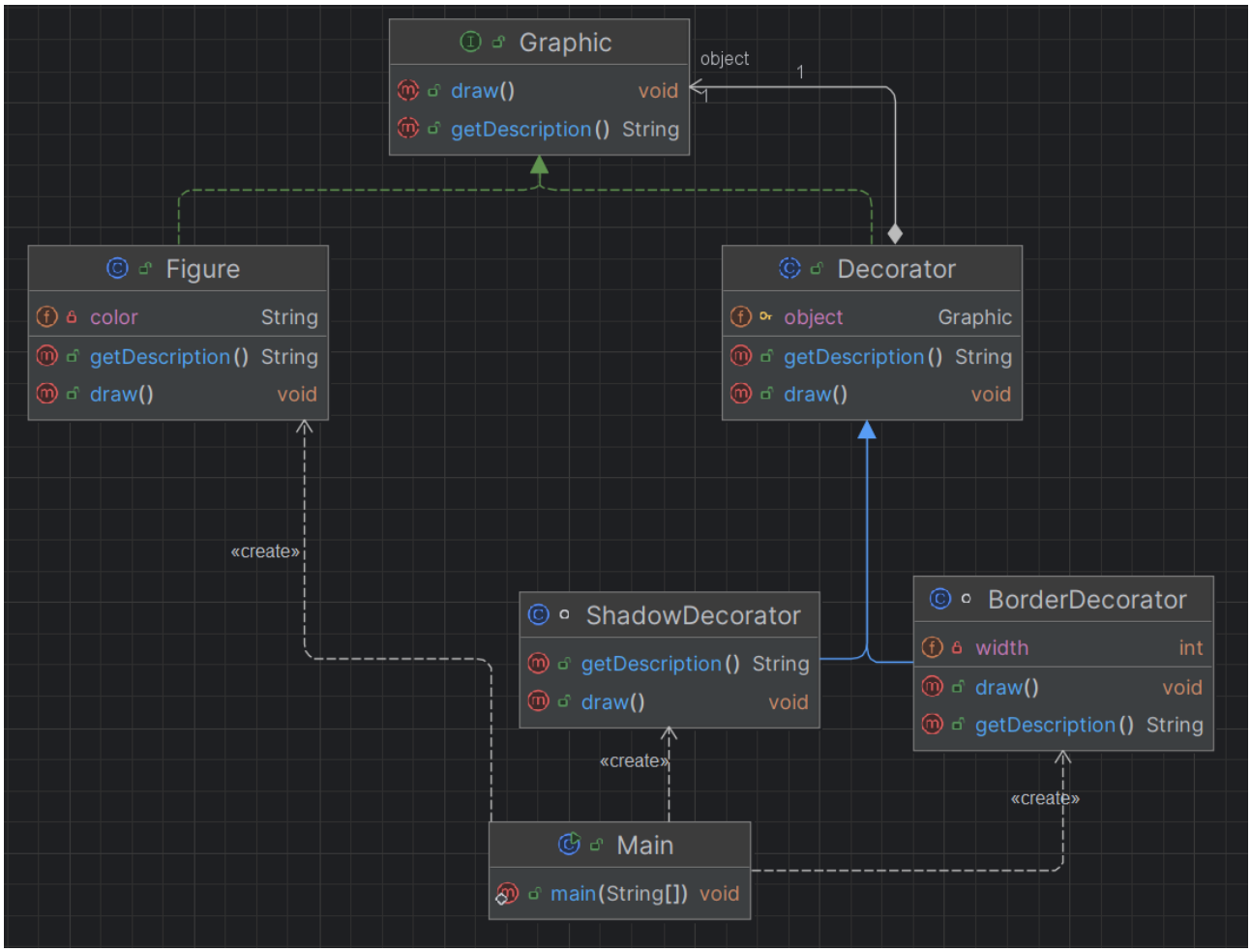
```java
/**
 * The BorderDecorator class is a decorator that adds a border to a graphic
object.
 * It extends the functionality of the base graphic object by adding a border of
specified width.
 */
class BorderDecorator extends Decorator {
    /**
     * The width of the border in pixels.
     */
    private int width;
    /**
     * Constructs a BorderDecorator that adds a border with the specified width
     * to the given graphic object.
     *
     * @param object the graphic object to decorate
     * @param width the width of the border in pixels
     */
    public BorderDecorator(Graphic object, int width) {
        super(object);
        this.width = width;
```

```
    }
    /**
     * Draws the border around the graphic object and then draws the object
itself.
     * This method first draws the border with the specified width, then calls
the
     * draw method of the decorated object.
     */
    @Override
    public void draw() {
        System.out.println("Малюємо рамку з товщиною " + width + " пікселі.");
        super.draw();
    }
    /**
     * Returns a description of the graphic object with the added border.
     *
     * @return a string describing the graphic object with the border, including
     *         the border thickness in pixels
     */
    @Override
    public String getDescription() {
        return super.getDescription() + " з рамкою товщиною " + width + "
пікселі";
    }
}
```

```
/**
 * The ShadowDecorator class is a decorator that adds a shadow effect to a
graphic object.
 * It extends the functionality of the base graphic object by applying a shadow.
 */
public class ShadowDecorator extends Decorator {

    /**
     * Constructs a ShadowDecorator that adds a shadow effect to the given
graphic object.
     *
     * @param object the graphic object to decorate with a shadow
     */
    public ShadowDecorator(Graphic object) {
        super(object);
    }

    /**
     * Draws the shadow effect and then the graphic object itself.
     * This method first adds the shadow, then calls the draw method of the
decorated object.
     */
    @Override
    public void draw() {
        System.out.println("Додаємо тінь");
        super.draw();
    }
    /**
     * Returns a description of the graphic object with the added shadow effect.
     * @return a string describing the graphic object with a shadow
     */
    @Override
    public String getDescription() {
        return super.getDescription() + " з тінню";
    }
}
```

```java
/**
 * The Figure class represents a simple figure, specifically a circle,
 * that can be drawn with a specified color.
 * It implements the Graphic interface.
 */
public class Figure implements Graphic {

    /**
     * The color of the circle.
     */
    private String color;

    /**
     * Constructs a Figure with the specified color.
     *
     * @param color the color of the circle
     */
    public Figure(String color) {
        this.color = color;
    }

    /**
     * Draws the circle with the specified color.
     * This method prints out a message indicating the color of the circle being
drawn.
     */
    @Override
    public void draw() {
        System.out.println("Малюємо " + color + " коло.");
    }

    /**
     * Returns a description of the circle including its color.
     *
     * @return a string describing the color of the circle
     */
    @Override
    public String getDescription() {
        return "Коло " + color + " кольору";
    }
}
```

Діаграма



Результат

```
C:\Users\evgon\.jdks\openjdk-21.0.2\bin\java.exe "-javaage
Додаємо тінь
Малюємо рамку з товщиною 2 пікселі.
Малюємо червоне коло.
Коло червоне кольору з рамкою товщиною 2 пікселі з тінню
```

Висновки: В результаті ознайомлення з видами шаблонів проектуванняя отримав базові навички застосування таких шаблонів, як Composite, Decorator та Proxy. Це допомогло мені зрозуміти, як ці шаблони покращують архітектуру програмного забезпечення, роблячи її більш гнучкою, масштабованою та легкою у підтримці.