Design Patterns Report:
Proxy и Flyweight

Student: Сабиева Арайлым

1. Proxy Pattern – Image Management

Problem: High-resolution images load slowly and require a lot of memory. Also, control access to sensitive actions. How it's solved: Proxy implements lazy loading (only when needed), and Protected Proxy implements access control. This reduces the load on the system and improves security.

```
public interface Image {
    void displayThumbnail();
    void displayFullImage();
    void uploadImage();
    void replaceImage();
}
```

```
public class RealImage implements Image {
    private String filename;

    public RealImage(String filename) {
        this.filename = filename;
        loadFromDisk();
    }

    private void loadFromDisk() {
        System.out.println("Загрузка полного изображения: "
+ filename);
    }
```

```
    @Override
    public void displayThumbnail() {
        System.out.println("Показ миниатюры: " + filename);
    }

    @Override
    public void displayFullImage() {
        System.out.println("Отображение полного изображения:
" + filename);
    }

    @Override
    public void uploadImage() {
        System.out.println("Изображение загружено: " +
filename);
    }

    @Override
    public void replaceImage() {
        System.out.println("Изображение заменено: " +
filename);
    }
}
```

```
public class ProxyImage implements Image {
    private String filename;
    private RealImage realImage;

    public ProxyImage(String filename) {
        this.filename = filename;
    }

    @Override
    public void displayThumbnail() {
        System.out.println("Показ миниатюры (быстро): " +
filename);
```

```
    }

    @Override
    public void displayFullImage() {
        if (realImage == null) {
            realImage = new RealImage(filename);
        }
        realImage.displayFullImage();
    }

    @Override
    public void uploadImage() {
        if (realImage == null) {
            realImage = new RealImage(filename);
        }
        realImage.uploadImage();
    }

    @Override
    public void replaceImage() {
        if (realImage == null) {
            realImage = new RealImage(filename);
        }
        realImage.replaceImage();
    }
}

public class ProtectedProxyImage implements Image {
    private ProxyImage proxyImage;
    private boolean isLoggedIn;

    public ProtectedProxyImage(String filename, boolean
isLoggedIn) {
        this.proxyImage = new ProxyImage(filename);
        this.isLoggedIn = isLoggedIn;
    }

    @Override
```

```
    public void displayThumbnail() {
        proxyImage.displayThumbnail();
    }

    @Override
    public void displayFullImage() {
        if (isLoggedIn) {
            proxyImage.displayFullImage();
        } else {
            System.out.println("Доступ запрещён:
пользователь не авторизован.");
        }
    }

    @Override
    public void uploadImage() {
        if (isLoggedIn) {
            proxyImage.uploadImage();
        } else {
            System.out.println("Доступ запрещён: только
авторизованные агенты могут загружать изображения.");
        }
    }

    @Override
    public void replaceImage() {
        if (isLoggedIn) {
            proxyImage.replaceImage();
        } else {
            System.out.println("Доступ запрещён: только
авторизованные агенты могут заменять изображения.");
        }
    }
}
```

```
public class ProxyPatternDemo {
    public static void main(String[] args) {
        Image image1 = new ProtectedProxyImage("house1.jpg",
false);
        image1.displayThumbnail();
        image1.displayFullImage();
        image1.uploadImage();
        image1.replaceImage();

        System.out.println();

        Image image2 = new ProtectedProxyImage("house2.jpg",
true);
        image2.displayThumbnail();
        image2.displayFullImage();
        image2.uploadImage();
        image2.replaceImage();
    }
}
```

2. Flyweight Pattern – Map Markers
   Problem: Duplicating the same objects for each marker wastes memory.
   How to solve: Flyweight pattern creates a shared style object that is reused by multiple markers. So with 10,000 markers, only 3 unique style objects are created

```
public class MarkerStyle {
    private String icon;
    private String color;
```

```
    public MarkerStyle(String icon, String color) {
        this.icon = icon;
        this.color = color;
    }

    public void displayStyle() {
        System.out.println("Стиль: " + icon + ", Цвет: " +
color);
    }
}
```

```
import java.util.*;

public class MarkerStyleFactory {
    private static final Map<String, MarkerStyle> styles =
new HashMap<>();

    public static MarkerStyle getStyle(String icon, String
color) {
        String key = icon + color;
        if (!styles.containsKey(key)) {
            styles.put(key, new MarkerStyle(icon, color));
        }
        return styles.get(key);
    }

    public static int getTotalStyles() {
        return styles.size();
    }
}
```

```
public class MapMarker {
    private int x, y;
    private MarkerStyle style;

    public MapMarker(int x, int y, MarkerStyle style) {
        this.x = x;
        this.y = y;
        this.style = style;
    }

    public void display() {
        System.out.print("Маркер в (" + x + ", " + y + ") с
");
        style.displayStyle();
    }
}
```

```
import java.util.*;

public class FlyweightPatternDemo {
    public static void main(String[] args) {
        List<MapMarker> markers = new ArrayList<>();
        String[] types = {"Gas", "Hospital", "Restaurant"};
        String[] colors = {"Red", "Green", "Blue"};

        Random rand = new Random();

        for (int i = 0; i < 10000; i++) {
            String type = types[i % types.length];
            String color = colors[i % colors.length];
            MarkerStyle style =
MarkerStyleFactory.getStyle(type, color);
            markers.add(new MapMarker(rand.nextInt(1000),
rand.nextInt(1000), style));
        }
```

```
        System.out.println("Всего маркеров: " +
markers.size());
        System.out.println("Уникальных объектов стиля: " +
MarkerStyleFactory.getTotalStyles());
    }
}
```