Laboratory Work Nr.0 SOLID PRINCIPLES

Course: Tehnici și mecanisme de proiectare software

Author: Tofan Liviu

Group: FAF-223

Theory

SOLID is an acronym that stands for five key design principles:

- Single responsibility principle.
- Open-closed principle.
- Liskov substitution principle.
- Interface segregation principle.
- Dependency inversion principle.

SOLID principles seek to reduce dependencies so that engineers can change one area of the software without impacting others. They make it easier to understand, maintain, and extend designs. Software engineers avoid issues and build adaptive, effective, and agile software using these SOLID principles.

Objectives:

Implement 2 SOLID letters in a simple project.

Implementation description

I have implemented a simple system where a user can log in, and upon login, the system can send a notification via different channels *(Email, SMS, Push Notification)*. I applied first 2 principles from SOLID, the **SRP** to separate responsibilities for authentication and notification sending. For **OCP**, I can created new notification methods like *Push Notifications* to be added without modifying the existing code.

Project Structure:

1. *`User:`* Represents the user entity.

```
public class User {  11 usages
    private String username;  2 usages
    private String email;  2 usages
    private String phoneNumber;  2 usages

    public User(String username, String email, String phoneNumber) {  1 usage
        this.username = username;
        this.email = email;
        this.phoneNumber = phoneNumber;
    }

    public String getUsername() {  5 usages
        return username;
    }

    public String getEmail() {  1 usage
        return email;
    }

    public String getPhoneNumber() {  1 usage
        return phoneNumber;
    }
}
```

Figure1. Class User

2. *`AuthService`:* Handles user authentication using to **SRP**.

```java
public class AuthService {  2 usages
    public boolean login(User user, String password) {  1 usage

        if ("password123".equals(password)) {
            System.out.println(user.getUsername() + " has logged in successfully!");
            return true;
        }
        System.out.println("Login failed for " + user.getUsername());
        return false;
    }
}
```

Figure2. Class AuthService handling Authentification Process

3. *`Notification`:* The base interface for sending notifications using tne **OCP**. It can be extended to support different types of notifications.

```java
import Lab0.User;

public interface Notification {  7 usages  3 implementations
    void send(User user);  3 usages  3 implementations
}
```

Figure3. Interface Notification

```java
public class EmailNotification implements Notification {  2 usages

    @Override  3 usages
    public void send(User user) {
        System.out.println("Sending Email to " + user.getEmail() + ": Welcome " + user.getUsername() + "!");
    }
}
```

Figure4. Email Notification

```java
public class PushNotification implements Notification {  2 usages

    @Override  3 usages
    public void send(User user) {
        System.out.println("Sending Push Notification to " + user.getUsername() + ": Welcome to the app!");
    }
}
```

Figure5. Push Notification

```java
public class SMSNotification implements Notification {  2 usages

    @Override  3 usages
    public void send(User user) {
        System.out.println("Sending SMS to " + user.getPhoneNumber() + ": Welcome " + user.getUsername() + "!");
    }
}
```

Figure6. SMS Notification

Results

```
Liviu Tofan has logged in successfully!
Sending Email to liviutofan@example.com: Welcome Liviu Tofan!
Sending SMS to 123-456-7890: Welcome Liviu Tofan!
Sending Push Notification to Liviu Tofan: Welcome to the app!
```

Figure7. Login Output

Conclusions

I used first two letters from SOLID and implemented them in a simple login/push notification

SRP: The `AuthService` class is only responsible for logging in the user, and the different notification classes are responsible for sending notifications. Each class does one thing and follows the Single Responsibility Principle.

OCP: The `Notification` interface is open for extension. We can add new notification types without modifying the existing notification classes.

So the SOLID principles are fundamental guidelines in software design that help developers create systems that are more

maintainable, scalable, and flexible. Principles contribute to building systems that are more robust, easier to test, and adaptable to change.

References

1. Solid Principles: https://www.bmc.com/blogs/solid-design-principles/

2. Solid Principles: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design