conceptsofcs Instagram (@conceptsofcs)

Master SOLID Principles

1. Single Responsibility Principle (SRP)

A class should have only one reason to change.

Violating SRP (A class that manages user authentication, profile updates, and reporting user activities all in one place)

```
class UserService {

  void registerUser(String username, String password) {
      // User registration logic
  }

  void loginUser(String username, String password) {
      // User authentication logic
  }

  void updateProfile(String username, String bio) {
      // Update user profile
  }

  void reportUser(String reportedUser) {
      // Report user logic
  }
}
```

Example Following SRP

✓

```
class AuthenticationService {
  void registerUser(String username, String password) { }
  void loginUser(String username, String password) { }
}

class ProfileService {
  void updateProfile(String username, String bio) { }
}

class ReportingService {
  void reportUser(String reportedUser) { }
}
```

2. Open-Closed Principle (OCP) - Post Management.

Software entities should be open for extension but closed for modification.

Violating OCP (Adding a new post type (video, text, or image) requires modifying the existing PostService)

✗

```
class PostService {
  void createPost(String type) {
    if (type.equals("text")) {
      // Create text post
    } else if (type.equals("image")) {
      // Create image post
    } else if (type.equals("video")) {
      // Create video post
    }
  }
}
```

Example Following OCP(Using polymorphism to support new post types without modifying existing code)

```
interface Post {
   void create();
}

class TextPost implements Post {
   public void create() { System.out.println("Creating text post"); }
}

class ImagePost implements Post {
   public void create() { System.out.println("Creating image post"); }
}

class PostService {
   void createPost(Post post) {
      post.create();
   }
}
```

3. Liskov Substitution Principle (LSP)

Objects of a derived class must be replaceable for objects of the base class without affecting functionality.

<u>Scenario: Subscription-Based Payments in a Social Media Platform</u>

A social media platform offers three types of subscriptions:

1. **Free Subscription - No payment required.**
2. **Basic Subscription - Monthly fixed fee.**
3. **Premium Subscription - Pay-per-feature model (e.g., extra storage, ad-free experience).**

Bad Example (Violating LSP)

Here, Subscription is the base class, but the FreeSubscription subclass **throws an exception** because it doesn't support payments.

```java
class Subscription {
    void processPayment() {
        System.out.println("Processing payment for the subscription...");
    }
}

class FreeSubscription extends Subscription {
    @Override
    void processPayment() {
        throw new UnsupportedOperationException("Free subscription does not
require payment!");
    }
}

class BasicSubscription extends Subscription {
    @Override
    void processPayment() {
        System.out.println("Processing monthly subscription payment...");
    }
}

class PremiumSubscription extends Subscription {
    @Override
    void processPayment() {
        System.out.println("Processing pay-per-feature payment...");
    }
}
```

Example following LSP

✓

```java
interface Payment {
    void processPayment();
}

abstract class Subscription {
    abstract void subscribe();
}

class FreeSubscription extends Subscription {
    @Override
    void subscribe() {
        System.out.println("User subscribed for free!");
    }
}

class BasicSubscription extends Subscription implements Payment {
    @Override
    void subscribe() {
        System.out.println("User subscribed to the Basic plan.");
    }

    @Override
    public void processPayment() {
        System.out.println("Processing monthly subscription payment...");
    }
}

class PremiumSubscription extends Subscription implements Payment {
    @Override
    void subscribe() {
        System.out.println("User subscribed to the Premium plan.");
    }

    @Override
    public void processPayment() {
        System.out.println("Processing pay-per-feature payment...");
    }
}
```

4. Interface Segregation Principle (ISP) - Media Upload

Clients should not be forced to depend on interfaces they don't use.

Bad Example (Violating ISP)

```
interface MediaUpload {
    void uploadImage();
    void uploadVideo();
    void uploadGIF();
}

class ImageUploader implements MediaUpload {
    public void uploadImage() { }
    public void uploadVideo() { throw new UnsupportedOperationException(); }
    public void uploadGIF() { throw new UnsupportedOperationException(); }
}
```

Example following ISP

```
interface ImageUpload {
    void uploadImage();
}

interface VideoUpload {
    void uploadVideo();
}

class ImageUploader implements ImageUpload {
    public void uploadImage() { }
}

class VideoUploader implements VideoUpload {
    public void uploadVideo() { }
}
```

5. Dependency Inversion Principle (DIP) - Notification System

High-level modules should not depend on low-level modules. Both should depend on abstractions.

Bad Example (Violating DIP)

Tightly coupling the notification system to a specific implementation (Email):

```java
class EmailNotification {
   void sendEmail(String message) { }
}

class NotificationService {
   private EmailNotification emailNotification = new EmailNotification();

   void notifyUser(String message) {
      emailNotification.sendEmail(message);
   }
}
```

Example following DIP

```java
interface Notification {
   void send(String message);
}

class EmailNotification implements Notification {
   public void send(String message) { }
}

class SmsNotification implements Notification {
   public void send(String message) { }
}

class NotificationService {
   private Notification notification;

   NotificationService(Notification notification) {
      this.notification = notification;
   }

   void notifyUser(String message) {
      notification.send(message);
   }
}
```

Summary of SOLID Principles

Principle	Use Case	Problem	Solution
SRP	User Management	UserService is handling multiple responsibilities	Separate AuthenticationService, ProfileService, and ReportingService
OCP	Post Creation	PostService needs modification for every new post type	Use a Post interface and create separate classes for text, image, and video posts
LSP	Payments	FreeSubscription throws an exception in processPayment() method	Add a separate Interface called Subscription so that FreeSubscription will implement Subscription Interface as it is nothing to do with payment
ISP	Media Upload	ImageUploader is forced to implement uploadVideo()	Split into separate interfaces for ImageUpload and VideoUpload
DIP	Notifications	NotificationService is tightly coupled with EmailNotification	Use a Notification interface and support multiple implementations