

## TP Analyse et amélioration d'un programme orienté objet

1. Dans VSCode, importez le projet EmpruntBibliotheque.zip fourni avec ce TP.

Fait :)

2. Analysez le code du programme.

Plusieurs fichiers ne respectent pas le principe SOLID. Ce qui compromet les bonnes pratiques et le bon fonctionnement du code.

3. Pour chaque principe SOLID

a) Expliquez en quoi le code du programme ne le respecte pas.

I. S – Single Responsibility Principle - SRP (Responsabilité unique)

```
public class Livre implements ArticleEmpruntable{
    private String titre;
    private boolean emprunte;

    public Livre(String titre) {
        this.titre = titre;
    }

    @Override
    public void emprunter() {
        emprunte = true;
        System.out.println("Livre emprunté : " + titre);
    }

    @Override
    public void rendreArticle() {
        emprunte = false;
        System.out.println("Livre rendu : " + titre);
    }

    // LSP violé : comportement incohérent
    @Override
    public void calculerPenaliteDeRetard(int jours) {
        if (jours < 0) {
            throw new IllegalArgumentException("Le nombre de jours ne peut pas être négatif !");
        }
        System.out.println("Pénalités de retard : " + (jours * 10) + " euros");
    }

    @Override
    public void imprimeEtiquette() {
        System.out.println("Etiquette : " + titre);
    }

    // DIP violé : dépendance directe à une classe concrète
    @Override
    public void connexionBDD() {
        ConnecteurMySql db = new ConnecteurMySql();
        db.connect("localhost", 3306, "root", "root");
    }

    @Override
    public void envoyerNotifEmail(String message) {
        EmailSmtp sender = new EmailSmtp();
        sender.send("user@example.com", "Message de votre bibliothèque", message);
    }
}
```

Cette classe ne respecte pas le principe SRP car elle a beaucoup de responsabilité au lieu d'une seule.

## II. O – Open/Closed Principle - OCP (Ouvert/Fermé)

```
public class GestionnaireBibliotheque {  
    private List<Object> items = new ArrayList<>();  
  
    public void addItem(Object item) {  
        items.add(item);  
    }  
  
    public void processItem(Object item, String type) {  
        if ("livre".equalsIgnoreCase(type)) {  
            Livre l = (Livre) item;  
            l.emprunter();  
            l.imprimeEtiquette();  
            l.connexionBDD();  
            l.envoyerNotifEmail("Livre emprunté !");  
        } else if ("dvd".equalsIgnoreCase(type)) {  
            DVD d = (DVD) item;  
            d.rendreArticle();  
            d.imprimeEtiquette();  
            d.connexionBDD();  
        } else {  
            System.out.println("Type d'article inconnu : " + type);  
        }  
    }  
}
```

Cette classe ne respecte pas le principe d'ouvert/fermé car on serait obligé de modifier la méthode processItem si l'on souhaite rajouter un type d'article autre qu'un DVD ou un livre.

## III. L – Liskov Substitution Principle - LSP (Substitution de Liskov)

```
@Override  
public void calculerPenaliteDeRetard(int jours) {  
    if (jours < 0) {  
        throw new IllegalArgumentException("Le nombre de jours ne peut pas être négatif !");  
    }  
    System.out.println("Pénalités de retard : " + (jours * 10) + " euros");  
}
```

Cette méthode ne respecte pas le principe LSP car DVD.emprunter() lève une UnsupportedOperationException, ce qui viole le contrat de ArticleEmprunable. Un objet dérivé ne peut pas être utilisé à la place du parent sans casser le comportement.

## IV. I – Interface Segregation Principle – ISP - (Séparation des interfaces)

```

public interface ArticleEmpruntable {
    void emprunter();
    void rendreArticle();
    void calculerPenaliteDeRetard(int jours);
    void imprimeEtiquette();
    void connexionBDD();
    void envoyerNotifEmail(String message);
}

```

Cette interface ne respecte pas le principe ISP car il y a trop de méthodes. La classe DVD est forcée d'implémenter des méthodes qui ne le concernent pas.

#### V. D – Dependency Inversion Principle –DIP

```

class MySqlDatabase {
    public void connect(String host, int port, String user, String pass) {
        System.out.println("Connexion MySQL à " + host + ":" + port);
    }
}

class SmtpEmailSender {
    public void send(String to, String subject, String body) {
        System.out.println("Email envoyé à " + to + ": " + subject);
    }
}

@Override
public void connexionBDD() {
    ConnecteurMySql db = new ConnecteurMySql();
    db.connect("localhost", 3306, "root", "root");
}

@Override
public void envoyerNotifEmail(String message) {
    EmailSmtp sender = new EmailSmtp();
    sender.send("user@example.com", "Message de votre bibliothèque", message);
}

```

Les classes Livre et DVD dépendent directement des classes concrètes ConnecteurMySql et EmailSmtp, au lieu de dépendre d'abstractions.

#### 4. Refactorisez le code en incluant les modifications proposées.

Voir code sur GitHub :)