TD 6 - Principes SOLID

Exercice 1. Single Responsability Principle

Pourquoi le code qui suit ne respecte pas ce principe ? Comment le corriger ?

```
public class PayRollService {
    ...

    public void payEmployee (Employee employee) {
        if (employee.getType() == EmployeeType.PERMANENT ) {
            ...
        } else if (employee.getType() == EmployeeType.CONSULTANT) {
            ...
        }
        paycheck = checkService.issuePaychek(employee);
        this.sendPaycheck(paycheck);
    }
}
```

Exercice 2. Open/Closed Principle

En quoi le code qui suit respecte-t'il ce principe, et en quoi le viole-t'il ?

```
public class Vehicle {
  public void startVehicle() { ... }
}

public class Car extends Vehicle {
  @Override
  public void startVehicle() {
    super.startVehicle();
  }
}

public class Truck extends Vehicle {
  @Override
  public void startVehicle() {
    super.startVehicle();
  }
}

public class GasStation {
  public void refillVehicle(Vehicle vehicle) {
    if(vehicle instanceof Car) { ... }
    else { ... }
  }
}
```

Exercice 3. Liskov Substitution Principle

On modifie l'exercice précédent en modifiant la class `Truck` :

```
public class Truck extends Vehicle {
    @Override
    public void startVehicle() {
        this.startMileageTracker();
        super.startVehicle();
    }
  }
}
```

et en ajoutant la classe `PremiumGasStation` :

```
public class PremiumGasStation extends GasStation {
    @Override
    public void refillVehicle(Vehicle vehicle) {
        if(vehicle.isType(Car)) {
            throw new RefillCarException();
        }
```

```
        super.refillVehicle();
    }
    ...
}
```

Le principe de substitution est-il respecté ?

Exercice 4. Interface Segregation Principle

Une voiture doit afficher sa vitesse sur 3 types d'affichage possibles en fonction du modèle de la voiture : digital, à aiguille ou en texte.

Concevez un système (un simple diagramme statique suffit) qui permet de répondre à ce besoin.

Exercice 5. Dependency Inversion Principle

Question 1. - *Cognage de clous* - Créez un projet en respectant les critères suivants :

- Un marteau peut cogner un clou.
- Un clou ne peut se faire cogner qu'une seule fois.
- Lorsqu'un clou se fait cogner, il envoi un message à l'utilisateur disant qu'il a été cogné.

Question 2. - *Par Minou* - Dans un jeu où les lions prétendent être des chats afin de mieux les manger, créez un projet tout en respectant les critères suivants :

- Le jeu commence avec une quantité de 100 souris.
- Le jeu détient une liste de 8 chats et 2 lions. Chaque espèce a accès à une action différente :
  - Chat : Manger une souris. Ceci décrémente de 1 le nombre de souris du jeu.
  - Lion : Manger un chat. Ceci retire le chat en question de la liste des chats du jeu.
- Respectez le DIP.

Pour vous aider, nous vous fournissons deux classes que vous pouvez modifier comme vous voulez :

```
public class Application {

  private static final int INITIAL_NUMBER_OF_MICE = 100;
  private static final int INITIAL_NUMBER_OF_CATS = 8;
  private static final int INITIAL_NUMBER_OF_LIONS = 2;

  public Application() {
    Game game = new Game(INITIAL_NUMBER_OF_MICE, INITIAL_NUMBER_OF_LIONS,
    INITIAL_NUMBER_OF_CATS);
  }
}

public class Game {

  private int numberOfMice;
  private final List<Cat> cats;
  private final List<Lion> lions;

  public Game(int numberOfMice, int numberOfLions, int numberOfCats) {
    this.numberOfMice = numberOfMice;
    this.cats = new ArrayList<>();
    this.lions = new ArrayList<>();

    for (int i = 0; i < numberOfLions; i++) {
      lions.add(new Lion());
    }

    for (int i = 0; i < numberOfCats; i++) {
      cats.add(new Cat());
    }
  }
}
```