**Rapport sur l'Implémentation des Concepts de Programmation Orientée Objet et de la Gestion des Exceptions**

Dans cette application de gestion de location de véhicules, plusieurs concepts de la Programmation Orientée Objet (POO) ont été utilisés pour structurer le code de manière efficace. Ce rapport décrit comment l'héritage, le polymorphisme, l'encapsulation et la gestion des exceptions ont été implémentés.

**1. Héritage**

L'héritage permet de créer une nouvelle classe basée sur une classe existante, facilitant la réutilisation du code. Dans notre application :

- **Classe Abstraite** Vehicule :

- La classe Vehicule est une classe abstraite qui définit les attributs et méthodes communs à tous les véhicules, tels que immatriculation, marque, et kilometrage.

- Elle déclare également la méthode abstraite calculerPrixLocation(), qui doit être implémentée par les classes dérivées.

- **Classes** Voiture et Camion :

- Ces classes étendent Vehicule, héritant de ses attributs et méthodes. Chaque classe implémente sa propre version de calculerPrixLocation(), fournissant ainsi des tarifs de location spécifiques.

Cette structure permet d'ajouter facilement d'autres types de véhicules à l'avenir, en créant simplement de nouvelles classes qui héritent de Vehicule.

**2. Polymorphisme**

Le polymorphisme permet d'utiliser des objets de classes dérivées comme s'ils étaient de la classe de base, facilitant ainsi l'interaction avec des collections d'objets.

- **Méthode** calculerPrixLocation()\*\* :

- Grâce au polymorphisme, nous pouvons appeler **calculerPrixLocation()** sur n'importe quel objet de type Vehicule, qu'il s'agisse d'une Voiture ou d'un Camion, sans se soucier du type spécifique. Cela simplifie la logique de traitement dans le reste de l'application.

**3. Encapsulation**

L'encapsulation consiste à restreindre l'accès direct aux données d'une classe, protégeant ainsi son état interne.

- **Attributs Privés** :

- Dans la classe Vehicule, l'attribut estDisponible est déclaré privé. Cela signifie qu'il ne peut être accédé directement que par les méthodes de la classe, comme louer() et retourner().

- De même, d'autres attributs comme immatriculation, marque, et modele sont encapsulés avec des méthodes d'accès (getters) pour permettre un accès contrôlé.

- **Méthodes** louer() et retourner():

- Ces méthodes modifient l'état de estDisponible, assurant que les règles de location sont respectées (c'est-à-dire qu'un véhicule ne peut pas être loué s'il n'est pas disponible).

**4. Gestion des Exceptions**

La gestion des exceptions est essentielle pour créer une application robuste. Dans notre code, nous avons implémenté deux exceptions personnalisées :

- **VehiculeIndisponibleException :**

- Cette exception peut être lancée lors de la tentative de location d'un véhicule qui n'est pas disponible. Cela permet de signaler clairement cette condition d'erreur à l'utilisateur.

- **ClientNonAutoriseException** :

- Bien que cette exception soit définie, son utilisation doit encore être intégrée dans le processus de location pour gérer les cas où un client n'est pas autorisé à louer un véhicule.

Les exceptions personnalisées permettent une meilleure gestion des erreurs en fournissant des messages clairs et spécifiques, ce qui améliore l'expérience utilisateur.

Cette application utilise les concepts de POO de manière cohérente pour créer une architecture modulaire et extensible. L'héritage et le polymorphisme facilitent l'ajout de nouvelles fonctionnalités, tandis que l'encapsulation protège les données internes et assure la cohérence de l'état des objets. La gestion des exceptions renforce la robustesse de l'application, permettant de traiter les erreurs de manière efficace. Ces principes contribuent à la maintenabilité et à l'évolutivité du code, rendant l'application prête pour des améliorations futures.