Encapsulation

Objectif

Comprendre le concept d'encapsulation en programmation orientée objet (POO) à travers des exemples simples. Vous allez découvrir les problèmes liés à l'absence d'encapsulation, puis apprendre comment l'encapsulation permet de protéger les données et de rendre le code plus robuste.

1. Qu'est-ce que l'encapsulation ?

L'encapsulation est un principe de la POO qui consiste à :

- **Protéger les données** d'un objet en restreignant leur accès direct depuis l'extérieur.
- **Regrouper les données** (attributs) et les **méthodes** qui les manipulent dans une même classe.
- Utiliser des **modificateurs d'accès** (comme private, public) pour contrôler comment les données sont lues ou modifiées.

Pourquoi c'est important ?

- Protège l'intégrité des données (évite des modifications non valides).
- Facilite la maintenance du code en cachant les détails d'implémentation.
- Rend le code plus sécurisé et modulaire.

Exo 1 : Un (contre)exemple

Dans le contexte du milieu bancaire: créez la classe Client et la classe Compte, et proposez dans la classe Compte les méthodes suivantes :

- deposer : pour déposer un montant en paramètre ;
- retirer : pour retirer une certaine somme si les fonds sont suffisants ;
- virer : pour faire un virement d'une certaine somme vers le compte d'un autre client.
- consulterSolde : affiche le solde actuel du compte.

Écrire des tests (manuels ou automatiques, bonus de 2 points si tests automatiques) pour pouvoir vérifier que tout fonctionne comme prévu. Vous devez décider quels sont les autres attributs de Client et de Compte. Mais, est-ce que le solde est vraiment sécurisé ?

Exo 2 : Un autre contre-exemple

On veut vendre des produits électroniques, pour cela vous allez créer une classe Produit, avec les attributs suivants : un id, un nom, une description, un prix unitaire, une catégorie (Ordinateur, Telephone, Accessoire, et rien d'autre).

1. En cherchant sur internet, quelle structure Java permettrait de représenter une catégorie ?
2. Créer les classes nécessaires. Ajoutez une méthode *baisserPrix* qui prend en paramètre un pourcentage, et qui diminue le prix d'un certain nombre de pourcents.

Sentez-vous que les objets de type Produits sont sécurisés ? que les prix seront toujours cohérents, même quand il y aura des dizaines de classes qui vont les manipuler ?

Exo 3 : Auteurs et livres

On veut représenter en programmation orientée objet des livres, leurs auteurs, et une bibliothèque, tout en respectant les règles de l'encapsulation.

Pour cela, créer les classes suivantes :

- Livre : avec un identifiant, un titre, un nombre de pages, une date de parution, un résumé et un ensemble de genres déjà prédéfinis (ex: romance, action, thriller, policier, éducatif);
- Auteur : avec un identifiant, un prénom, un nom de famille, un nom d'auteur, une date de naissance, un genre, et un pays ;
- Bibliothèque : avec un identifiant, un nom, une adresse, et une ville.

En termes de composition, on sait qu'un livre est rédigé par un auteur, et qu'une bibliothèque contient un ensemble de livres. Par ailleurs, on devrait avoir dans la bibliothèque :

- une méthode pour ajouter un nouveau livre ;
- une méthode pour retirer un livre par son id;
- une méthode pour avoir la liste de tous les livres triés par titre croissant;
- une méthode pour chercher un livre par titre ;
- une méthode pour chercher un livre par genre ;
- une méthode pour chercher un livre par mot-clé dans leurs résumés (insensible à la casse);
- une méthode pour avoir la liste des auteurs distincts qu'ils ont;
- une méthode pour avoir l'ensemble des genres, et le nombre de livres par genre. Vous pourrez utiliser une Map<Genre, Integer> pour les représenter par exemple ;

Exo 4 : Commandes et Plats

On souhaite programmer une application pour gérer les commandes de plats en ligne. Pour représenter ces informations, nous allons créer les classes suivantes en respectant les règles de l'encapsulation :

- Plat : caractérisé par un id, un nom, un prix unitaire, et un type (entrée, résistance, dessert, snack, boisson, et rien d'autre);
- Ingrédient : caractérisé par un id, et un nom.
- Client : caractérisé par un id, un nom, un prénom, et un contact téléphonique.
- Commande : une commande est caractérisé par un id, une date de commande, et le client qui a fait la commande;
- PlatCommande : qui désigne un plat commandé, et qui se caractérise par un id, le plat commandé, la commande à laquelle elle est rattachée, et la quantité commandée.

Implémentez également les méthodes suivantes :

- Dans un plat, on devrait pouvoir voir s'il contient oui ou non, un ingrédient fourni en paramètre ;
- Dans une commande, on aimerait pouvoir savoir si elle est vide ou non ;
- Dans une commande, on devrait pouvoir retourner l'ensemble des plats commandés, leur quantité ;
- Dans une commande, on devrait pouvoir retourner son prix total ;
- Dans une commande il devrait être possible de la combiner avec une autre commande en paramètre, de sorte à ce qu'elle ne fasse plus qu'une seule commande, avec la liste des plats combinés des deux commandes ;
- Dans une commande, il devrait être possible de compter le total d'un type de plat fourni en paramètre. Par exemple : Le total des boissons seulement dans la commande.
- Dans une commande, il devrait être possible de retourner le plat le plus cher qui a été commandé dans celle-ci.

Points importants : quand est ce qu'on dit que deux objets sont égaux ? est ce que == est vraiment le bon opérateur pour comparer des objets ? il compare quoi en fait ce == au niveau des objets ? c'est quoi la méthode equals et à quoi pourrait elle nous servir dans ce contexte ?

Comment peut-on représenter une date en Java ? Pourquoi est-ce qu'il y a Date et LocalDate ? Quelle est la différence entre les deux ? Laquelle est conseillée ?

Exo 5 : Vente de boissons

On souhaite numériser la vente d'une brasserie comme Star Madagascar. Pour cela nous allons commencer par créer les classes qui nous permettront de représenter les données à l'intérieur d'un tel système.

En suivant les règles de l'encapsulation nous allons donc créer les classes suivantes :

- Boisson : avec un id, un nom, une quantité (une quantité est décrite par un nombre décimal et une unité, par exemple 1.5L, 500ml, 250ml, 2L), et un prix unitaire ;
- Commande : avec un id, une date et heure de la commande, ainsi que le client qui a initié la commande ;
- BoissonCommande : désigne une boisson commandée avec un id, et la quantité de chaque boisson ;
- Client : avec un id, un nom, un prénom, un contact téléphonique, et une adresse e-mail optionnelle ;
- Brasserie : avec un ensemble de boissons qu'ils proposent, un ensemble de clients et les commandes qu'elle a reçues.

Ajoutez les méthodes suivantes dans la classe Brasserie :

- Il doit être possible de retourner l'ensemble des produits dont le nom contient un certain mot clé, insensible à la casse ;
- Il doit être possible de retourner l'ensemble des clients dont le nom ou le prénom ou l'adresse e-mail contient un certain mot clé, insensible à la classe ;
- Il doit être possible de calculer combien de clients ont un numéro orange, yas, et airtel. Vous pouvez utiliser une Map<String, Integer> pour représenter cette information par exemple ;

Pour chaque commande, il doit être possible aussi de voir combien de litre de boisson elle représente en tout. Par exemple, une commande qui contient un coca 1.5L, et un fanta 250ml, le tout représente un total de 1.750L de boisson.

Exo 6 : Vers la notion d'héritage

On revient sur la notion de Robot qui a marqué notre tout premier exercice dans ce cours. Oui, ce robot qui a un x et un y (que l'on pourrait condenser en une classe Point justement), une direction, et qui peut avancer, mais aussi tourner à droite.

On vous demande de créer une nouvelle classe de robot appelé RobotNG (NG = nouvelle génération) avec exactement les mêmes attributs et méthodes mais avec de nouveaux éléments :

- tourner à gauche: qui peut s'obtenir en tournant à droite 3 fois ;
- avancer mais avec un paramètre qui est le nombre de pas : les RobotNG peuvent désormais aller plus vite en faisant plusieurs pas en un seul mouvement ;
- reculer : qui consiste à avancer dans la direction opposée ;

Quelques questions à se poser :

- Au lieu de créer cette classe de 0, n'aurions nous pas pu nous réutiliser le code de notre cher Robot ?
- Est-ce que c'est autorisé d'avoir deux méthodes du même nom tel que avancer ? sous quelles conditions ?

(La suite … vers l'abstraction et le polymorphisme).