Tokimahery Ramarozaka : Le but de ces exercices est que vous compreniez bien le “pourquoi” de l’orienté objet, non pas seulement “comment on le fait”, car le “Comment” est vraiment routinier au bout d’un moment que tout le monde peut le faire.

Comme je vous l’ai déjà expliqué au tout début du cours, son objectif est de rassembler le code dans des structures qui s’appellent des OBJETS qui sont des instances de classes. Les classes spécifient en fait la structure que doit respecter chaque objet.

En utilisant des classes et en les instanciant un objet, nous créons en fait nos propres types qui ont leurs propres méthodes, plutôt cool que d’utiliser seulement des tableaux à longueur de journée pour tout stocker, non ?

Pour vous résumer le plan de bataille, les quatre piliers de la POO (Programmation Orientée-Objet) sont notamment :

1) L’encapsulation
2) L’abstraction
3) L’héritage ou généralisation
4) Le polymorphisme

Que des mots compliqués me direz-vous ? nous verrons. Ces 4 premiers exercices vont vous introduire à l’encapsulation : l’art de pouvoir cacher certains attributs / méthodes. On peut notamment faire en sorte que certaines méthodes / attributs ne soient pas accessibles, ou bien qu’ils soient en écriture / lecture seulement.

Dans l’exo 1, vous allez donc voir comment on le fait, avec les mots clés *private, protected, public*. Et surtout, faire vos premiers “*getters*” et “*setters*”, notamment vous devez comprendre (ou poser la question sinon, je vous vois chers élèves archi-timides) la convention de tout mettre en *private* et de faire ces getters et setters. Nous verrons ça en correction notamment. A terme, avec ces exos, l’encapsulation, on saura le faire ... mais, surtout pourquoi le faire?

Pour finir, l’exo 1 sera un exemple tout simple pour se mettre dans le bain, dès l’exo 2 et 3 vous allez comprendre le concept de composition : comment utiliser des classes pour construire d’autres classes ? (ex: utiliser la classe Author, pour créer un Book, parce qu’un Book a un Author, n’est ce pas ?). Quant à l’exo 4, il sera notre point d’entrée vers le prochain pilier : l’héritage !). Tchaooo!

Exo 1 :

Créer une classe *Employee*, en respectant les règles d'encapsulation (getter, setters, constructeur) avec les attributs suivants : id int, lastName : String, firstName : String, salary : double.

Ajoutez la méthode :

- raise(int percent) qui va augmenter le salaire de l'employé de tel pourcentage.
- toString() qui va retourner une représentation en String de l'objet

Créez **une autre classe** EmployeeTest pour vérifier que toutes les méthodes marchent bien.

Exo 2 :

Créer une classe *OrderItem* (représentant un article dans une commande) en respectant les règles d'encapsulation, avec les attributs suivants : id (int), description (String), quantity (int), unitPrice (double).

Ajoutez la méthode :

- getTotal(), qui va retourner le prix total de l'article commandé : ce total s'obtient notamment avec la formule unitPrice * quantity.
- toString() qui va retourner une représentation en String (@Override);

1) Créez **une autre classe** *OrderItemTest* pour tester que toutes les méthodes marchent.

2) Ensuite, créez la classe *Order*, qui représente une commande, et qui contient 1) Un string décrivant le nom du client; puis 2) plusieurs articles (plusieurs OrderItem). Proposez une classe pour représenter *Order*, et ajoutez-y une méthode pour calculer le grand total de la commande, c'est-à-dire la somme des valeurs totales de chaque article (*OrderItem*).

Exo 3 :

Dans cet exercice on souhaite modéliser des auteurs et des livres. Pour cela, créez les classes *Author* et *Book* en suivant les règles d'encapsulation.

Un *Author* est caractérisé par : un nom, un e-mail, et un sexe (‘M’ ou ‘F’), tandis qu’un *Book* est caractérisé par : un titre, un *Author* (un auteur), un prix (double), et une quantité (le nombre d’exemplaires en stock). Donnez un moyen de savoir si un Book est en rupture de stock (quantité = 0).

Vous devez mettre les méthodes *toString*() appropriées pour les deux classes. Dans une autre classe *BookTest.java,* créez des instances de Author et de Book. Enfin, essayez d’accéder depuis une instance de Book : le nom de son auteur (à afficher avec sout).

Exo 4 :

On souhaite créer des classes pour modéliser des comptes bancaires, pour cela créer deux classes en suivant les règles d’encapsulation : *Customer* & *Account*.

Un client (customer) est caractérisé par un nom, prénom, numéro de téléphone, e-mail, date de naissance et une adresse e-mail.

Un compte (account) contient : un numéro de compte (int), un Customer propriétaire du compte, et un solde (de type double, initialisé par défaut à 0); mais également les méthodes suivantes :

- credit(double amount) : va créditer le compte d’une certaine somme;
- debit(double amount) : va débiter le compte d’une certaine somme;
- transferToAccount(Account target, double amount) : va faire un virement d’une certaine somme à un autre compte.

Créez une classe *AccountTest* pour tester que ces 3 méthodes marchent.