Fiche 5 : ArrayList et égalité structurelle

Objectifs

- Pouvoir utiliser une `ArrayList`.
- Pouvoir implémenter une association multiple.
- Pouvoir implémenter la méthode `equals`.
- Pouvoir implémenter la méthode `hashCoce` dans un cas simple.

Vocabulaire

Association multiple	ArrayList	equals	hashCode

Exercices

1. Catalogue

Récupérez la classe `Livre` sur moodle.

a) Modifiez la classe Livre de la façon suivante :
- Faites-en sorte que les constructeurs lancent une `IllegalArgumentException` si on leur passe une chaîne de caractères vide ("").
- Ajoutez les méthodes `equals` et `hashCode`. Un livre sera identifié par son ISBN.

La classe `Catalogue` permet de gérer un catalogue de livres. Pour cela, elle doit garder une « liste » de livres. Il faut implémenter un constructeur sans paramètre (il permet d'initialiser la « liste ») et une méthode `toString()`. De plus, il faut pouvoir effectuer les opérations suivantes :

- voir si un livre est présent dans le catalogue c.-à-d. s'il existe déjà un livre avec le même ISBN dans le catalogue ;
- ajouter un livre au catalogue (si celui-ci n'est pas déjà présent) ;
- retirer un livre du catalogue ;
- donner le nombre de livres présents dans le catalogue ;
- dire si le catalogue est vide ;
- récupérer un livre du catalogue en fonction de son ISBN.

b) Complétez le diagramme UML des classes Catalogue et Livre.

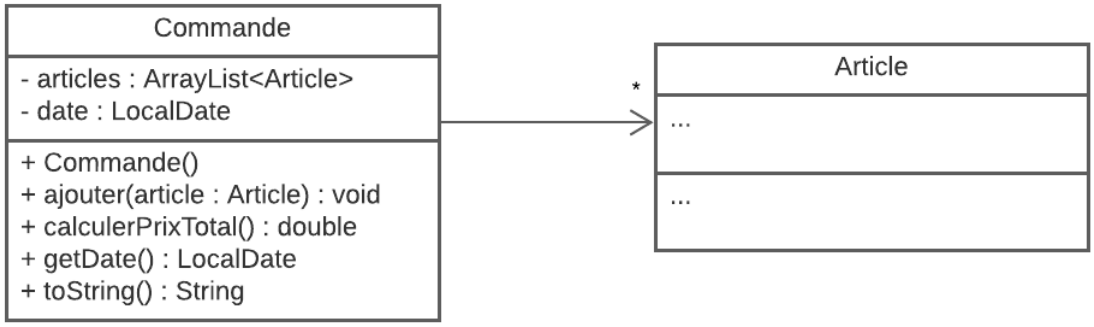
Livre
- isbn : String - nom : String - prenom : String - titre : String - nbPages : int - prixConseille : double - anneeEdition : int
+ Livre(isbn : String,nom : String,prenom : String, titre : String, nbPages : int, prixConseille : double, anneeEdition : int) + Livre(isbn : String,nom : String,titre : String, nbPages : int, prixConseille : double, anneeEdition : int) + getIsbn() : String + getTitre() : String + getNom() : String + getPrenom() : String + getNbPages() : int + getPrixConseille() : double + getAnneeEdition() : double + setPrixConseille(prix : double) : void + toString() : String

Catalogue

c) Implémentez la classe `Catalogue` en java.
d) Récupérez la classe `TestCatalogue` sur moodle et adaptez-là pour qu'elle compile avec vos classes. Exécutez `TestCatalogue` et vérifiez que vous obtenez bien l'affichage attendu (fourni dans le fichier affichage_testCatalogue).
e) Représentez les objets qui se trouvent en mémoire à la fin de l'exécution du programme `TestCatalogue`.

2. Commande

Récupérez les classes `Article`, la classe `TestCommande` ainsi que le canevas de la classe `Commande` sur moodle.
Modifiez la classe `Article` afin qu'un article soit identifié par sa référence.
On vous demande de compléter la classe `Commande` qui contiendra une liste d'articles correspondant aux articles d'une commande.



a) Observez attentivement le code qui se trouve déjà dans la classe `Commande`.

La classe `Commande` garde une `LocalDate`[1] afin de garder la date de la commande.
Pour l'instant, lorsqu'on **crée une commande**, le constructeur **initialise** la date à la date du jour. Complétez ce constructeur afin qu'il initialise également l'`ArrayList`.

Complétez ensuite la méthode `toString` en tenant compte des commentaires indiqués.

Ajoutez les méthodes manquantes en tenant compte des indications ci-dessous :

- Lorsqu'on ajoute un article, celui-ci est ajouté après les articles déjà présents même s'il se trouve déjà dans la commande. Il y aura donc autant d'éléments dans la liste que d'articles dans la commande. On ne considère pas la quantité pour le moment. Si l'on essaie d'ajouter un article `null` alors on lance une `IllegalArgumentException`.
- La méthode `calculerPrixTotal` renvoie le prix total de la commande ; elle parcourt la liste et somme les prix des articles (tvac).

b) Complétez la classe de **test** `TestCommande` fournie afin de tester la classe `Commande`. Il s'agit de créer 2 commandes et de les afficher :
   - l'une va contenir le vélo de femme (`article1`) , le tandem (`article2`) et encore une fois le vélo de femme.
   - l'autre uniquement le tandem.

3. Ligne de commande

Bien entendu, la gestion des articles en quantité unique n'est pas optimale. C'est pour cela que nous allons créer un nouveau type, `LigneDeCommande`, qui correspond à une ligne de la commande dans une commande. Maintenant, si un article est commandé plusieurs fois dans une commande alors il apparaîtra dans une seule ligne avec la quantité adéquate. Dans chaque ligne on référence donc un article et sa quantité désirée.

Dans la classe `LigneDeCommande` on place les attributs, l'un de type `Article` et l'autre de type entier pour représenter la quantité. On y met également un constructeur qui prend en paramètre uniquement l'article concerné par cette ligne de commande et place la quantité à 1. Un autre constructeur prendra en paramètres les valeurs des deux attributs. Les constructeurs lanceront une `IllegalArgumentException` si on leur passe un paramètre invalide (l'article ne peut pas être `null` et la quantité doit être strictement positive).
On ajoute des getters pour les deux attributs mais un setter uniquement pour la quantité. Le setter lancera également une `IllegalArgumentException` si la quantité passée en paramètre n'est pas valide.
Il faut aussi pouvoir calculer le prix total de la ligne de commande (tva comprise).
On ajoute finalement la méthode `toString` reprenant la quantité suivie d'un x suivi du `toString` de l'article.

Il faut aussi modifier la classe `Commande` afin qu'elle conserve maintenant une liste de lignes de commandes. Attention, il faut aussi une méthode supplémentaire qui permettra d'ajouter une quantité donnée d'un article à la commande. La méthode `toString` sera adaptée afin de renvoyer une chaîne de caractères constituée de la date de la commande, de la liste des

[1] En consultant l'API, il sera aisé d'utiliser cette classe. Constatez qu'il faut importer une librairie via l'instruction `import java.time.LocalDate;` juste au-dessus de la déclaration de la classe dans le fichier.

articles commandés avec la quantité commandée et le prix pour cette quantité (un article par ligne) et du prix total de la commande.

a) **Modifiez vos classes en UML.**
b) Implémentez ces modifications en java.

Complétez et exécutez votre classe `TestCommande`.

4. BONUS - Commande

Ajoutez, dans la classe `Commande`, des méthodes pour :

- supprimer un article et donc la ligne de commande qui le concerne ;
- modifier la quantité commandée d'un article ;
- vérifier si un article est déjà commandé ;
- trouver la quantité d'un article commandé ;
- donner la liste des articles commandés.

Toutes les méthodes recevant un article en paramètre lanceront une `IllegalArgumentException` si celui-ci est `null`.