TP n°1 sur les collections (ArrayList, itérateurs, Set)

V. Aponte

13 septembre 2023

Exercice 1

Les solutions à cet exercice sont à ajouter dans la classe `ExerciceArrayListe` du projet gitinfo fourni.

1. Créer dans la méthode main un `ArrayList` d'entiers que vous devez initialiser avec 50 entiers tirés au hasard en utilisant la méthode `Math.random`.
2. Complétez la méthode `AfficheListeInt` qui affiche une liste d'entiers à l'aide d'une boucle *for each*. Notez que l'argument de cette méthode est de type `List<Integer>`. Ajoutez dans le main un appel à cette méthode pour afficher l'`ArrayList` d'entiers crées. Expliquez pourquoi il est possible de passer un `ArrayList<Integer>` à la méthode, alors qu'elle attend un paramètre de type `List<Integer>`.
3. Utilisez un *itérateur* pour enlever de votre liste tous les éléments compris entre 10 et 20.
4. Trouvez dans la bibliothèque la méthode permettant de tester si la liste contient le nombre 35 et affichez un message indiquant si ce nombre est ou non dans la liste.
5. Lisez la documentation de la classe `Collections` afin de trouver les méthodes pour (1) trier et afficher cette liste en ordre croisant, (2) trouver et afficher l'élément le plus grand de la liste, puis (3) trier et afficher la liste en ordre décroissant.
6. Ecrivez une méthode statique `insertTrie` qui prend en argument un entier n et une liste d'entiers **dont on suppose qu'elle est triée en ordre croissant**. Votre méthode doit modifier la liste argument pour y ajouter n à la bonne place de sorte que la liste modifiée soit toujours triée en ordre croissant. **Attention, cette méthode ne doit faire aucun tri !** Ajoutez dans le main des appels pour tester cette méthode.

Exercice 2

Créez une liste de mots, initialisée avec quelques mots en double.

Question 1

— Lisez la documentation des classes qui implantent `Set` et trouvez le moyen de créer un ensemble contenant tous les mots de votre liste sans doublons et triés en ordre alphabétique.
— Utilisez ensuite un itérateur pour créer une chaîne formée par concaténation de tous les mots de l'ensemble, séparés par des blancs.
— Trouvez la méthode permettant de renvoyer l'élément le plus petit de l'ensemble et affichez-le. **Attention** : dans le contexte d'ensembles triés, on entend par *mot le plus petit*, celui qui est le plus petit **selon l'ordre de tri de l'ensemble**. Comme nous avons un ensemble trié de `String` et que l'ordre par défaut sur ce type-là est l'ordre alphabétique, au final, ce que l'on veut obtenir est le plus petit alphabétiquement parlant, parmi les mots de l'ensemble.
— Trouvez la méthode permettant d'obtenir le sous-ensemble contenant tous les élément plus petits (au sens de l'ordre de l'ensemble) qu'un élément donné afin de calculer le sous-ensemble contenant touts les mots plus petits que le mot "pourquoi". Utilisez un itérateur pour afficher ce sous-ensemble.

Question 2

Ecrivez une méthode statique `charsIn(String s)` qui renvoie une collection contenant les caractères différents qui apparaissent dans la chaîne s. Quel type de collection utilisez vous ? Ecrivez une autre méthode qui affiche un mot et tous ses caractères différents. Ecrivez une méthode qui enlève de cette collection tous les caractères chiffres et qui retourne la liste de tous ces chiffres triés en ordre croissant.

Exercice 3

Cet exercice utilise la classe `Compte` fournie dans le projet gitinfo fourni (`package exercice3`)

Question 1

- On veut empêcher la création des comptes avec un solde négatif. Dans la classe `Compte` modifiez le constructeur de sorte qu'il lève `IllegalArgumentException` si le solde initial du compte est négatif.
- Ajoutez une méthode `toString()` et une méthode `affiche()` qui l'utilise pour afficher les données courantes d'un compte.
- Rédéfinissez la méthode `retrait` de sorte qu'elle échoue avec `RuntimeException` si le solde du compte est insuffisant pour réaliser un retrait.
- Notez que dans cette classe il n'y a pas de mutateur pour les attributs `titulaire` et `numero`. Quelles raisons ont pu motiver ce choix ?

Question 2

Définissez une nouvelle classe `CompteDecouvert` par héritage à partir de la classe `Compte`. Vous pouvez vous inspirer de la version donnée dans les transparents du cours sur l'héritage. L'attribut `solde` de `Compte` est `protected`. Que se passe-t-il si on le change en `private`, comme pour les autres attributs de cette classe ? Essayez cette solution, et justifiez votre réponse.

Question 3

Créez une nouvelle classe `PortefeuilleComptes` destinée à réunir l'ensemble de comptes d'un même titulaire. Cette classe aura comme attributs le nom du titulaire du portefeuille et un ArrayList de ses comptes. On va considérer que les instances de cette classe devront respectecter à tout moment de leur vie les deux **invariants** :

1. *à tout moment, les comptes du portefeuille possèdent le même nom de titulaire que celui du portefeuille*,
2. *à tout moment, les comptes du portefeuille ont des adresses distinctes*

Vos méthodes devront faire en sorte de ne pas violer ces invariants. Par ailleurs, on va supposer que deux instances distinctes de `Compte` ont forcément des numéros de compte différents (i.e., on n'invoque pas le constructeur `Compte` avec deux fois le même numéro de compte). Vous n'oublierez pas d'ajouter un constructeur pour la classe `PortefeuilleComptes`. Quels paramètre lui passez vous ?

Dans la suite, toutes boucles sur la liste de comptes seront écrites soit avec un for-each, soit avec un itérateur.

- Ajoutez les méthode accesseur qui vous semblent pertinentes. Est-il raisonnable de fournir un accesseur retournant la liste des comptes ? Est-il raisonnable de fournir une opération permettant de retourner un objet compte du portefeuille ? Expliquez.
- Ecrire une méthode qui renvoie true si le portefeuille ne contient aucun compte.
- Ecrire une méthode qui renvoie true si le portefeuille contient un compte d'un certain numéro.
- Ajoutez une méthode permettant l'ajout d'un nouveau compte au portefeuille. Cette méthode doit échouer si le compte à ajouter et le portefeuille n'ont pas le même nom de titualire. Pourquoi cela ? Elle ne fera rien si un compte de même adresse se trouve déjà dans la liste. Utilisez vous la *délégation* ?

— Ecrivez des méthodes permettant de réaliser toutes les opératjons courantes sur un compte du portefeuille (dépôt, retrait, obtenir le solde). Ces opérations prendront en paramètre le numéro du compte concerné et les autres paramètres nécessaires. Si aucun compte de ce numéro n'existe dans le portefeuille, elles devront léver `IllegalArgumentException` . Vos opératiions utilisent-elles la *délégation* ?
— Ajoutez une méthode permettant de retirer du portefeuille le compte d'un certain numéro. Cette suppression ne sera possible que si le compte retiré a un solde nul.
— Ajoutez une méthode qui renvoie le bilan du portefeuille.
— Ecrivez une méthode `toString()` qui retourne une chaîne composée du titulaire et des tous les comptes du portefeuille. Utilisez vous la *délégation* ?
— Ecrivez une méthode qui retourne la liste de tous les numéros de compte ayant un solde négatif (est-ce possible ?)
— Créez une nouvelle classe `TestPortefeuille` avec une méhode main où vous créerez des comptes et des comptes avec découvert et les ajouterez dans un ou plusieurs portefeuilles.

Question 4 (à faire à la maison)

Dans cette question, on suppose qu'un titulaire peut avoir plusieurs objets portefeuilles à son nom. Ecrivez une méthode `fusion` qui prend en paramètre un portefeuille p et le fusionne avec le porteuille courant. Cela revient à rétirer tous les comptes du portefeuille p et les verser dans le portefeuille courant. Bien entendu, si les deux portefeuilles n'ont pas le même nom de titulaire l'opération échoue, et si des comptes existent des deux côtés ils ne seront pas ajoutés une deuxième fois au portefeuille courant, mais ils seront tout de même rétiré de p. En fin d'opération, le portefeuille p doit être vide. Quel tests doit faire cette méthode pour ne pas violer l'invariant de la classe ?

Question 5 (à faire à la maison)

La liste de comptes aurait pû être implanté avec un `Set`. La raison est qu'un de nos invariants interdit les doublons de comptes ayant la même adresse, ce qui est le comportément *par défaut* de `Set`. Ce point technique sera étudié lors du deuxième cours sur les collections. En attendant, modifiez ce code pour utiliser un `Set<Compte>`.