JAVA

EXERCICES

Travail sur les Collections

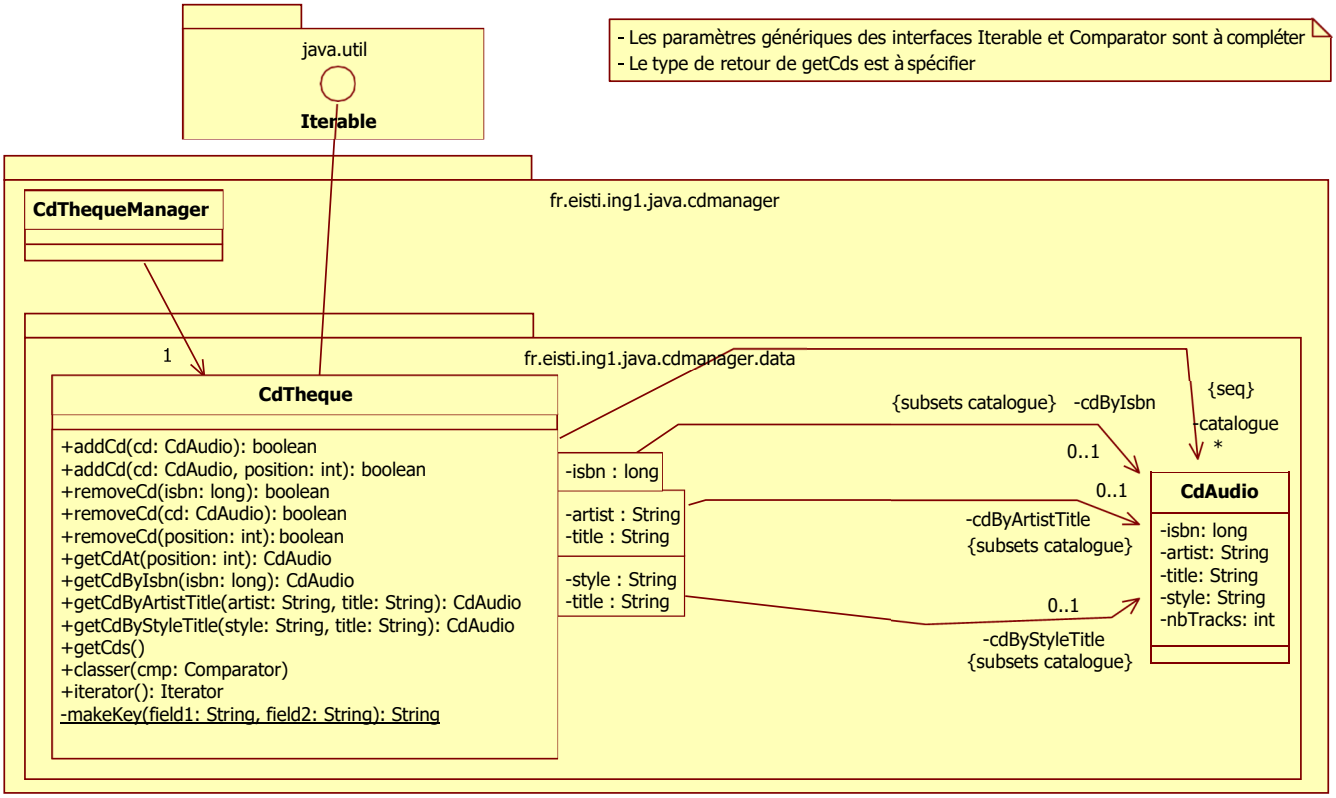
A. La classe LigneBrisee en collections

On veut créer des classes **LigneBriseeCollection,** comme celle vue précédemment dans les exercices sur les objets, qui contiendra une liste de points, mais stockée maintenant dans les différents types de collections. On mettra le nom de la collection traitée à la place de **Collection** pour chaque nouvelle classe.

Ainsi, après avoir créé un package **ligneBrisee** spécifique dans lequel vous transférerez la classe **LigneBrisee** en tableau faite dans le TD sur les objets (vous la renommerez en **LigneBriseeTab**), vous essaierez l'exercice avec les différents types de collection (ArrayList, HashSet, LinkedList, Map et TreeSet), en créant une classe spécifique à chaque fois :

1. Écrire la classe **LigneBriseeArrayList** (vous changerez le nom de la classe pour chaque type de collection), contenant la liste de points, avec ses accesseurs, mutateurs et toString. Une instance contiendra au moins 2 points au départ.
2. Écrire une méthode **contientPoint** qui renvoie vrai si le point passé en paramètre est dans la ligne brisée et faux sinon.
3. Écrire une méthode **addPoint** permettant d'ajouter un point, à condition qu'il ne soit pas déjà dedans.
4. Écrire une méthode **nbPoints** permettant d'afficher le nombre de points de la ligne brisée.
5. Écrire une méthode **deletePoint** permettant de supprimer un point.
6. Ecrire une classe **Main** qui permettra de tester la création de la ligne brisée, l'ajout et la suppression d'éléments (penser à tester des actions qui fonctionnent et d'autres qui ne fonctionnent pas, comme la suppression d'un élément qui n'existe pas ou dans une collection vide, etc…), avec à chaque fois un affichage de la ligne brisée.
7. Quelles sont les avantages inconvénients des collections testées ? Et par rapport à la classe **LigneBriseeTab** gérée par un tableau ?

B. Création d'une CDThèque



1. Écrire la classe **CdAudio** avec un constructeur complet et des accesseurs en lecture.
2. Écrire la classe **CdTheque** qui permet de gérer un catalogue de **CdAudio** avec 3 systèmes d'indexation. Vous respecterez le diagramme UML ci-dessus avec les indications suivantes :
   a. Un constructeur par défaut qui initialise un catalogue vide et les index associés ;
   b. La méthode **addCd** renvoie **true** si l'ajout a bien été effectué et false si un **CdAudio** existe déjà pour une des 3 indexations ; dans ce dernier cas, l'ajout est annulé ;
   c. La méthode **removeCd** renvoie **true** si un **CdAudio** a été effectivement supprimé ;
   d. Les méthodes de recherche renvoient **null** si aucun **CdAudio** est associé au critère de recherche ;
   e. La méthode **getCds** renvoie une liste non modifiable de Cd pour respecter l'encapsulation de la classe **CdTheque** ;
   f. La méthode **classer** permet de choisir un classement du catalogue différent de l'ordre d'ajout, avec un ordre passé en paramètre ;
   g. La méthode **makeKey** permet d'obtenir une clé normalisée à partir de deux informations sous la forme de chaînes de caractères ; on pourra par exemple mettre les chaînes en majuscules et supprimer les caractères spéciaux (espace, quote, virgule, etc…) ; la clé finale est obtenue par concaténation des 2 chaînes normalisées séparées par un # ; par exemple :

      artist : Pink
      titre : F*ckin' Perfect
      clé obtenue : PINK#FCKINPERFECT

3. Écrire une application **CdThequeManager** créant une **CdThèque** et testant les différentes méthodes produites. Il n'est pas demandé de faire une application interactive, ce sera l'objet de futurs TDs.

C. Étude de performance

L'objectif de ce TP est de manipuler et de comparer l'usage des différentes classes de Collections (List et Set)

1. int et Integer

En Java, il existe des types de base **(int, long, float, double, boolean**…) et leurs équivalents sous forme d'objets (**Integer, Long, Double, Boolean**…).
Examiner dans la documentation, la classe Integer.
Quels sont les avantages et les inconvénients de manipuler des entiers sous la forme d'objets plutôt que de types simples ?

2. Écrire un programme Java permettant de créer la liste des multiples de 3 inférieurs à 9999, dans un objet de la classe **ArrayList**. Afficher cette liste en utilisant la méthode toString de cette liste.

3. Mesures du temps écoulé et de la mémoire utilisée.

Utiliser la méthode non statique **freeMemory()** de la classe **Runtime** pour mesurer la mémoire utilisée et la méthode statique **System.currentTimeMillis()** pour calculer le temps de calcul du programme précédent. N'hésitez pas à consulter la documentation Java pour connaître l'usage exact de ces méthodes. NB : selon la puissance de votre ordinateur, n'hésitez pas à modifier la valeur 9999 pour obtenir des résultats significatifs.

4. Ajouter au programme les instructions nécessaires à la suppression dans la liste des multiples de 3, un par un, dans l'ordre décroissant, de la valeur 9999 jusqu'à 3. Mesurer le temps nécessaire à cette suppression et la mémoire utilisée à la fin de cette suppression. Pourquoi ne revient-on pas à la valeur mémoire initiale ?

Utiliser la méthode **gc()** de la classe **Runtime** (voir la documentation de cette méthode) entre la suppression des éléments de la liste et le calcul de la mémoire utilisée. Que constatez-vous ?

5. Transformez le programme en définissant une méthode statique **testCollection(Collection c).** Dans le programme principal, lancer cette méthode en lui passant successivement comme paramètre une **ArrayList**, une **LinkedList**, un **TreeSet** puis finalement un **HashSet**.

Comparer les performances respectives (mémoire et temps de calcul) de ces 4 structures de données. Comment expliquez-vous de telles différences de performances ?

6. Les structures ArrayList et LinkedList semblent moins performantes que les structures Set. En revanche, quelles fonctionnalités possèdent les listes que n'ont pas les Set ?

Illustrer ces fonctionnalités en cherchant le $100^{ème}$ élément de la liste.
On prendra soin de tester chaque exercice pour en valider le résultat.