Université de Paris **Programmation Orientée Objet**
L2 **Année 2021-2022**

TP n° 4

Champs statiques et héritage : gestion d'une médiathèque

On modélise une application devant servir à l'inventaire d'une médiathèque. La classe `Mediatheque` contiendra la méthode principale `main` permettant de tester les différentes classes de ce TP.

Exercice 1 Une médiathèque contient différents types de médias. Quelque soit le média, celui-ci possède un *titre* donné dès la création et qui par la suite ne change plus.

1. Définir la classe `Media` avec son constructeur public, la propriété `titre` privée et son accesseur.
2. On veut attribuer un *numéro d'enregistrement* unique dès que l'on crée un objet `Media` : le premier média créé doit avoir le numéro 0, puis ce numéro s'incrémente de 1 à chaque création de média. Ajouter les attributs nécessaires, modifier le constructeur puis ajouter une méthode `getNumero` renvoyant le numéro d'enregistrement du média. On ajoutera qu'il ne doit pas être possible de modifier ce numéro.
3. Définir la méthode `toString` renvoyant la chaîne de caractères constituée du numéro d'enregistrement et du titre du média.
4. Définir la méthode `plusPetit(Media doc)` qui vérifie que le numéro d'enregistrement de l'instance courante est plus petit que celui de *doc*.

Exercice 2 Nous nous intéressons à des médias de natures diverses : des livres, des dictionnaires, ou encore d'autres types de médias que l'on ne connaît pas précisément mais qu'il faudra certainement ajouter un jour (bandes dessinées, dictionnaires bilingues,...). A chaque livre est associé, en plus, un *auteur* et un *nombre de pages*, les dictionnaires ont eux pour attributs supplémentaires une *langue* et un *nombre de tomes*[1]. On veut manipuler tous les articles de la médiathèque au travers de la même représentation : celle de média.

1. Définissez les classes `Livre` et `Dictionnaire` étendant la classe `Media`. Définissez pour chacune un constructeur permettant d'initialiser toutes ses variables d'instances respectives.
2. Redéfinissez la méthode `toString()` dans les classes `Livre` et `Dictionnaire` pour qu'elle renvoie une chaîne de caractères décrivant un livre ou un dictionnaire, en plus de la description normale d'un média.
3. Définissez ensuite quelques classes supplémentaires : `DictionnaireBilingue`, `BandeDessinee` , `Manga`, (etc.) avec des propriétés en plus. Pour chacune posez-vous la question de sa place dans la hiérarchie des classes déjà présentes.

Exercice 3 Une médiathèque est une classe contenant l'attribut `LinkedList<Media> baseDeDonnees` , des méthodes opérant sur celle-ci et le `main` pour faire les tests de ce TP.

1. Définir la classe `Mediatheque` ainsi qu'un constructeur affectant une base de données vide.

1. On ne considère pas les dictionnaires comme un cas particulier de livres. En particulier, on ne leur attribuera ni auteur, ni nombre de pages.

2. Définir une méthode `ajouter(Media doc)` qui ajoute *doc* dans la base de données tout en la laissant triée pour l'ordre `plusPetit`.
3. Tester tout ceci dans le `main` de la classe `Mediatheque` : créer une liste contenant plusieurs médias puis l'afficher (redéfinir la méthode `toString()` si nécesaire).
4. Définissez une méthode `tousLesDictionnaires` qui imprime tous les dictionnaires (on pourra utiliser **`instanceof`**).

Exercice 4 Lorsque l'on affiche la base de données de la médiathèque, les médias sont classés par numéro d'enregistrement, mélangeant des médias de types différents. Ce n'est pas très lisible. Nous aimerions modifier l'ordre pour que les médias d'un même type apparaissent à côté, toujours classés par numéro. Nous essayons ici de séparer les livres du reste.

1. Redéfinissez la méthode `plusPetit(Media doc)` de la classe `Livre` de sorte que l'instance courante soit considérée comme plus grande que `doc` si celui-ci n'est pas un livre (on utilisera **`instanceof`**). Lorsque `doc` est un livre on réutilisera la version précédente de `plusPetit`.
2. Testez cette nouvelle version de `plusPetit` en reprenant les tests de l'exercice 3. Est-ce-que les livres sont bien séparés des autres médias ? Si non pourquoi ?
3. Surchargez la méthode `plusPetit(Media doc)` de la classe `Media` avec la méthode `plusPetit (Livre doc)`, de sorte que `doc` soit considéré comme plus grand que l'instance courante si celle-ci n'est pas un livre (et comparé par numéro sinon).
4. Cette version n'est pas satisfaisante, pourquoi ?
5. Pour arriver à notre but, plutôt que de passer par l'opérateur **`instanceof`**, on peut définir une méthode **`int`** `ordreMedia()` sans argument qui associe à un média un entier qui ne dépend que de son type (Livre, DVD. . .). On utilise ensuite cette méthode dans `plusPetit` pour classer les médias par type, puis par numéro. Cette méthode `ordreMedia`, doit-elle être une méthode statique ou une méthode d'instance ?
6. Implémentez `ordreMedia`, ainsi que la nouvelle version de `plusPetit` qui l'utilise. Vous penserez à enlever la redéfinition de `plusPetit` dans la classe `Livre`. Vérifiez avec vos tests qu'on obtient bien le résultat attendu.

Exercice 5 On veut maintenant pouvoir rechercher les médias grâce à des requêtes. Par exemple, on veut pouvoir rechercher un livre, écrit par Balzac, et qui fait moins de 300 pages.

1. Créez une classe `Predicat` qui contient une seule méthode **`boolean`** `estVrai(Media m)`, qui renvoie toujours **`false`** (Qui est une valeur par défaut comme nous le verrons par la suite, dans les sous-classes de `Predicat` cette méthode indiquera si le prédicat en question est vrai pour ce média).
2. Créez une sous-classe `EstUnLivre` de `Predicat` dont la méthode `estVrai(Media m)` indique si m est un livre ou non.
3. Ajoutez à la classe `Mediatheque` une méthode `ArrayList<Media> recherche(Predicat p)` qui retourne tous les médias pour lesquels `p` est vrai (ie. tels que `p.estVrai(m)` est **`true`**).
4. Créez de même une classe `TitreCommencePar` contenant un champ `lettre` de type **`char`** et dont la méthode `estVrai(Media m)` indique si le titre du média commence par cette lettre, de manière insensible à la casse (aidez-vous des méthodes `Character.toUpperCase` et `Character.toLowerCase`).

5. On veut maintenant pouvoir combiner ces prédicats : créez une classe `Et` qui hérite de `Predicat`, qui contient deux prédicats `p1` et `p2` et qui est vrai pour un média `m` lorsque `p1` et `p2` sont vrais pour `m`.
6. Pour tester, construisez le prédicat "livres dont les titres commencent par S".

Exercice 6 (*Bonus*) On veut maintenant pouvoir rechercher les documents grâce à des requêtes approximatives. Par exemple, on veut que si on cherche "Balzak" ou "Blzac", on puisse trouver "Balzac".

1. Ajoutez un nouveau type de prédicat `TitreEstAPeuPres` contenant un champ `titre` qui est vrai si le titre du média est égal à `titre` à une lettre près (une lettre qui change, ou en plus, ou en moins).
2. Modifiez la classe précédente pour qu'elle contienne un nouveau champ `distance` et qu'elle soit vraie quand le titre est, au plus, à `distance` lettres du titre recherché.
3. Ajoutez une classe pour permettre la recherche des médias dont le champs du titre contient une sous chaîne particulière.