Polytech'Paris-Saclay ET4-Info TP noté Novembre 2020

On veut réaliser un système de recherche bibliographique pour une bibliothèque. La bibliothèque contient divers types de documents, dont des livres, des livres d'images et des revues. Tout document stocké dans la bibliothèque a un titre, un nom d'éditeur et une année de publication. Les livres ont en plus un auteur. Les revues ont un numéro de volume. La bibliothèque est susceptible de stocker d'autres types de documents dans le futur et votre application doit être évolutive.

On veut offrir aux lecteurs des outils de recherche de documents selon différents critères. On considère des critères simples comme « la liste des documents publiés en 2000 » ou « la liste des documents dont l'éditeur est X », mais aussi des critères obtenus par combinaison d'autres critères.

Partie A

1. On s'intéresse à des critères simples (sans combinaison). Écrire une classe `Critere` munie de la méthode `match` ci-dessous ainsi que d'une méthode `toString()`.

```
abstract boolean match(Document d) throws ErreurCritere
```

L'exception `ErreurCritere` est levée si le critère n'est pas applicable au document en question.

2. Définir une sous-classe `DateComp` de `Critere` munie du constructeur `DateComp(int year)` et d'une définition de la méthode `match` qui renvoie `true` si et seulement si le document passé en paramètre a la même année de publication que celle définie par le constructeur du critère.

3. Définir deux sous-classes `TitleComp` et `AuthorComp`, la première étant adaptée à une recherche sur le titre d'un document et la seconde à une recherche sur des livres uniquement et qui renvoie `true` si et seulement si l'auteur du livre est celui défini dans le critère.

4. Définir la méthode d'en-tête `select(Critere c) throws ErreurCritere` dans `Bibliotheque` qui renvoie la collection des documents de la bibliothèque qui satisfont le critère.

5. On veut maintenant pouvoir combiner des critères par des connecteurs comme «et», «ou» et «non» et construire dynamiquement de tels critères. Une classe qui implémente un critère composé est munie d'un constructeur vide et d'une méthode `void add(Critere c) throws ErreurCritere` qui ajoute `c` à la liste des critères à appliquer par l'opérateur. Exemple :

```
Critere cl = new And();
cl.add(new DateComp(2000)); cl.add(new TitleComp("Mickey"));
```

Définir des classes `And`, `Or`, `Not` munies des méthodes nécessaires. Une instance de `And` ou `Or` n'est applicable que si elle contient au moins un critère. Une instance de `Not` n'est n'applicable que si elle comporte exactement un critère.

Partie B (schéma observer/observable)[1] : Les bibliothécaires voudraient connaître les critères (ou combinaisons de critères) qui ont retenu certains documents particuliers lors des sélections lancées par les utilisateurs. En s'inspirant du schéma Observer/Observable, écrire un ensemble de méthodes qui enregistre pour un document « observé » la liste des critères lancés par les utilisateurs qui ont retenu ce document parmi leur résultat. On doit pouvoir débuter ou arrêter l'observation d'un ouvrage, et imprimer la liste des critères qui ont sélectionné l'ouvrage observé

Attention : on ne doit pas notifier un observateur pour un document qui ne satisfait pas globalement le critère : par exemple si on applique un critère de la forme `Not(c)`, un document sélectionné par `c` ne doit pas être signalé !

On pourra se servir de la classe `HashMap<Key, Value>` qui est munie des fonctions suivantes :

[1] Les parties B et C sont indépendantes et peuvent être traitées dans un ordre quelconque.

`containsKey(Object k):`	Returns `true` if this map contains `k` as a key
`get(Object k):`	Returns the value to which `k` is mapped, or `null` if absent
`keySet():`	Returns the set of keys contained in this map.
`put(Key k, Value v):`	Associates `v` with `k` in this map.

Partie C: à réaliser dans une hiérarchie séparée, les en-têtes de classes étant incompatibles.

Dans la première partie on ne peut pas définir de critères qui statiquement ne s'appliquent qu'à un sous-type de documents. On veut donc paramétrer la classe `Critère` par le type de documents concernés. Une instance de `Critere<Livre>` ne s'appliquera qu'à des instances (au sens large) de `Livre` et non pas des documents quelconques. L'en tête de la classe `Critere` devient :

```
public abstract class Critere<T extends Document> { … }
```

1. Définir la méthode `match` de la classe `Critere`. Redéfinir `DateComp` et `AuthorComp` : en particulier `AuthorComp` ne s'applique plus qu'à un objet typé comme `Livre`.

2. Modifiez la hiérarchie des critères complexes pour qu'ils soient eux-mêmes paramétrés. L'en-tête de la méthode `add` doit garantir qu'on ne combine que des sous-critères adaptés au type de critère : un critère complexe sur les livres peut combiner des critères généraux comme `CompDate` et des critères spécifiques aux livres comme `AuthorComp`. Par exemple :

```
And<Livre> cl = new And<Livre>();
cl.add(new DateComp(2000)); cl.add(new AuthorComp("Victor Hugo"));
```

Si `cl` avait été déclaré de type `And<Document>`, le compilateur devra signaler une erreur sur l'ajout de `AuthorComp` celui-ci n'étant pas applicable à tout document. De même, si on cherchait à combiner le critère `AuthorComp` avec le critère `VolumeComp` : aucun document n'est compatible avec ces deux critères simultanément !

3. Définir dans `Bibliothèque` une **méthode** `static` **générique** `filtre` prenant en paramètre une collection `col` de documents et un critère, tous les deux adaptés au type du résultat attendu, qui renvoie la collection des éléments de `col` qui satisfont le critère. Le type de la collection et celui du critère dépendent donc du type des résultats qu'on veut obtenir. Exemples :

```
ArrayList<Livre> col1 = …;       // contient des instance de Livre ou LivreImage
ArrayList<Document> col2 = …;    // contient des documents quelconques
ArrayList<LivreImage> col3 = …;// contient des instance de  LivreImage
AuthorComp comp = new AuthorComp("Victor Hugo")
ArrayList<Livre> r1 = Bibliotheque.filtre(col1, comp);    // OK
ArrayList<Livre> r2 = Bibliotheque.filtre(col2, comp);    // KO
ArrayList<Livre> r3 = Bibliotheque.filtre(col3, comp);    // OK
```

4. Peut-on généraliser la méthode `select` de `Bibliotheque` en des méthodes d'en-tête :

```
<T extends Document> Collection<T> select(Critere<T> c)
```

ou

```
<T extends Document> Collection<T> select(Critere<? super T> c)
```