Java Avancé

Atelier 2 – partie 2 : énumérés, collections

Table des matières

1 Objectifs 2
2 Concepts 2
2.1 Map 2
3 Exercices 2
3.1 Introduction 2
3.2 Consignes 2
3.3 Énuméré Type 2
3.3.1 Créer l'énuméré 2
3.3.2 Adapter la classe Plat 2
3.3.3 Adapter le main 2
3.4 La classe Livre 3
3.4.1 La classe 3
3.4.2 La structure de données 3
3.4.3 Modifier le livre 3
3.4.4 Afficher la table des matières 4
3.5 Création d'un livre de test 4
3.5.1 Premiers tests 4
3.5.2 Tests un peu plus poussés 5
3.6 Améliorer le Livre 6
3.7 Améliorer l'énuméré Type de plat 7
3.8 Challenge optionnel 7
3.8.1 StringBuilder 7
3.8.2 Trier tous les plats 7
3.8.3 Map triée ? 7

1 Objectifs

ID	Objectifs
AJ01	Structurer son code en utilisant des packages
AJ02	Savoir écrire des classes énumérées et les utiliser correctement
AJ03	Être capable de manipuler des collections d'objets
AJ04	Être capable d'écrire une **classe interne** et savoir l'utiliser

2 Concepts

2.1 Map

La théorie nécessaire se trouve dans le fichier collections.pdf

3 Exercices

3.1 Introduction

Dans cette partie, nous allons introduire la notion de livre de recettes.

Dans notre livre, on pourra y retrouver plusieurs plats, organisés par type (entrée, plat, dessert). Pour chaque type de plat, les recettes seront triées par ordre de difficulté.

On souhaiterait également pouvoir récupérer facilement tous les plats d'un certain type. Par exemple, toutes les entrées, ou tous les desserts.

3.2 Consignes

Pour cette partie, nous allons repartir de la solution de la partie 1.

Dans IntelliJ, créez un projet intitulé AJ_atelier02_partie2. Ajoutez-y les classes de la solution de l'atelier 2 partie 1 disponible sur Moodle. Faites attention aux packages !

Vérifier qu'il fonctionne correctement en démarrant le `main`.

3.3 Énuméré Type

3.3.1 Créer l'énuméré

Pour pouvoir trier les plats par type (entrée, plat, dessert) dans notre livre, il va falloir créer cette notion de « type de plat » dans notre application. Comme les types sont bien définis et à priori invariables, on pense immédiatement à un énuméré. Comme ce type n'a de sens que pour un plat, on va le mettre en classe interne.

Créez cet énuméré `Type` dans la classe `Plat`, avec 3 valeurs (`ENTREE, PLAT, DESSERT`).

3.3.2 Adapter la classe Plat

Ajoutez dans la classe plat un attribut `type`, avec un getter, pas de setter, et ajoutez le `type` également dans le constructeur.

3.3.3 Adapter le main

Vous remarquerez en modifiant le constructeur qu'il signale un « related problem ». Cliquez dessus pour vous retrouver dans le `main`. Corrigez-y le problème.

Lancer votre projet pour vérifier que tout fonctionne correctement. S'il n'y a pas de message d'erreur, et que tout s'affiche comme au début, c'est sans doute OK.

3.4 La classe Livre

3.4.1 La classe

Créez une classe Livre dans le package `domaine`. Elle reste vide pour l'instant, on va réfléchir à ce qu'on met dedans ensuite.

3.4.2 La structure de données

Un livre va être composé d'un ensemble d'objets `Plat`. Mais sous quelle forme retenir ces plats ? Une liste ? Un ensemble ? Autre chose ?

Pour pouvoir faire un choix, il faut réfléchir à la manière dont on va utiliser cette structure de données. Comment on va insérer des choses dedans, comment on va en supprimer, y accéder…

QUESTION : Relisez bien l'introduction… Quelle est la structure de données qui nous conviendrait ici ?

Lisez la réponse ci-dessous uniquement quand vous avez fait l'effort de réfléchir.

Commençons par réfléchir à l'élément suivant « *les recettes seront triées par ordre de difficulté* ». On parle de tri ! Il faut donc retenir un certain ordre. Pour cela, on pense tout de suite à une liste qui conserve l'ordre d'insertion. Mais il y a mieux ! Dans une liste, il peut y avoir des doublons. Or dans un livre de recettes, il n'y a pas de doublons… Un ensemble (`Set`) alors ? Mais il n'est pas trié… Il y a une solution à cela : `SortedSet` ; un ensemble trié.

Nous avons donc maintenant notre base. Un ensemble trié de recettes. Mais il y a un second point d'attention à ne pas oublier : « *On souhaiterait également pouvoir récupérer facilement tous les plats d'un certain type* ». On pressent donc qu'il y aura une méthode `getPlatsParType(Type type)` qui va renvoyer tous les plats de ce type. Un `SortedSet` ne permet pas de récupérer facilement tous les plats d'un certain type.

C'est là que la `Map` va nous aider. On va faire une `Map` dont la clé est le type de plat, et la valeur un ensemble trié de plats. Si on a 3 types de plat, on aura donc une `Map` qui va contenir 3 `SortedSet<Plat>`.

QUESTION : Pourquoi ne pas faire juste 3 attributs `SortedSet<Plat>` plutôt qu'une `Map` ?

Créez l'attribut `plats` dans la classe `Livre` avec la structure de données décrite ci-dessus.

3.4.3 Modifier le livre

Que serait un livre de recettes sans recettes ? Ajoutez les méthodes suivantes dans la classe `Livre` :

```
/**
 * Ajoute un plat dans le livre, s'il n'existe pas déjà dedans.
 * Il faut ajouter correctement le plat en fonction de son type.
 * @param plat le plat à ajouter
 * @return true si le plat a été ajouté, false sinon.
 */
public boolean ajouterPlat(Plat plat) {
    return false;
}

/**
 * Supprime un plat du livre, s'il est dedans.
```

```
 * Si le plat supprimé est le dernier de ce type de plat, il faut supprimer
ce type de
 * plat de la Map.
 * @param plat le plat à supprimer
 * @return true si le plat a été supprimé, false sinon.
 */
public boolean supprimerPlat (Plat plat) {
    return false;
}
```

Complétez ces méthodes.

ASTUCES :

- Lorsque vous souhaitez ajouter un plat, il faut d'abord récupérer le bon ensemble de plats en fonction du type. Au début, il n'y aura encore aucun ensemble dans la Map. Il faut donc à chaque fois vérifier si l'ensemble existe avant d'ajouter le plat.
- Vous n'allez pas pouvoir tester tout de suite vos méthodes… C'est normal. Attendez un peu avant d'exécuter.

3.4.4 Afficher la table des matières

Maintenant qu'on peut ajouter et supprimer des recettes de notre livre, ce serait bien d'avoir une méthode pour afficher la liste des plats du livre, de sorte qu'on puisse tester nos méthodes.

Ecrivez la méthode `toString()` de la classe `Livre`. Voici un exemple d'affichage :

```
ENTREE
=====
Croquettes au fromage

PLAT
=====
Waterzooi
```

3.5 Création d'un livre de test

3.5.1 Premiers tests

Pour tester nos méthodes `toString()`, `ajouterPlat()` et `supprimerPlat()`, nous avons besoin d'adapter le main. Ajoutez ceci à la fin de la méthode `main()` :

```
Livre livre = new Livre();
livre.ajouterPlat(plat);
System.out.println(livre);
livre.supprimerPlat(plat);
System.out.println(livre);
```

Exécutez le main et observez… Si vous n'avez pas fait d'erreur précédemment, vous devriez avoir une erreur comme ceci :

```
Exception in thread "main" java.lang.ClassCastException: class domaine.Plat
cannot be cast to class java.lang.Comparable
```

Corrigez les autres erreurs jusqu'à ce que vous arriviez à ce message d'erreur. Si vous n'avez pas le message d'erreur, vous avez sans doute résolu le souci spontanément. Bravo ! Mais lisez quand même la suite, on ne sait jamais 😜

Il nous dit qu'il y a un souci, car il n'arrive pas à transformer la classe `Plat` en une classe `Comparable`. C'est assez logique en soi… On essaye d'ajouter un Plat dans un ensemble trié. Mais comment est-ce qu'il doit trier les plats ? On ne lui a jamais dit comment le faire…

Il y a 2 solutions :

- Soit on laisse `Plat` implémenter l'interface « `Comparable` », et donc la méthode `compareTo()`. En faisant cela, on laisse la classe `Plat` décider de comment elle se juge comparable. Donc comment elle juge qu'un plat sera avant ou après un autre.
- Soit on donne un `Comparator` à l'ensemble trié. C'est une méthode qui définit comment comparer les objets qui seront dans l'ensemble. Dans ce cas, c'est l'ensemble qui a la responsabilité de la comparaison, et plus la classe `Plat`.

QUESTION : Quelle solution vous semble la plus adaptée ici ?

Lisez la réponse ci-dessous uniquement quand vous avez fait l'effort de réfléchir.

Si on laisse le choix à la classe `Plat` de décider comment elle se compare, on va y définir que c'est sur base de la difficulté. Car c'est ça dont on a besoin ici. Mais imaginons que demain on doit faire un 2e type de livre, qui lui contient une liste triée en fonction du coût. On va faire comment ?

Et si on utilise dans plusieurs classes cette comparaison, mais que demain on change la méthode `compareTo()`. On va influencer toutes les classes qui utilisent cette comparaison. Ce comportement est peut-être souhaitable. Mais dans notre cas, c'est clairement défini que la comparaison doit être fait sur base de la difficulté. Il ne faut donc pas risquer que quelqu'un change ce comportement (de la classe `Livre`) par erreur, en changeant le `compareTo()` de la classe `Plat`.

Modifiez la méthode `ajouterPlat()` en passant en paramètre un `new Comparator<Plat>` lors de la création du `TreeSet()`.

ASTUCES :

- C'est possible de faire la méthode `compare()` en une ligne.
- N'hésitez pas à déléguer la comparaison à une méthode compare d'une autre classe.
- Regardez ce qu'il existe comme méthodes pour un énuméré…

Exécutez votre main. Il devrait afficher, en plus de la recette du waterzooi, le livre de recettes :

```
PLAT
=====
Waterzooi
```

3.5.2 Tests un peu plus poussés

Modifiez votre méthode `main()` avec la création du livre, pour y mettre ce code-ci :

```
Livre livre = new Livre();
livre.ajouterPlat(plat);
livre.ajouterPlat(new Plat("Croquettes au fromage", 4, Difficulte.XXX,
Cout.$$, Plat.Type.ENTREE));
```

```
System.out.println(livre);
livre.supprimerPlat(new Plat("Toasts aux champignons", 5, Difficulte.XXX,
Cout.$$$, Plat.Type.ENTREE));
System.out.println(livre);
```

NE L'EXECUTEZ PAS ENCORE ! Lisez le code et essayez d'imaginer le résultat attendu.

QUESTION : Que va afficher ce bout de code ?

Une fois que vous avez une idée du résultat attendu, exécutez-le.

QUESTION : Si vous vous êtes trompés, pourquoi est-ce que ça affiche ce résultat-là ?

Lisez la réponse ci-dessous uniquement quand vous avez fait l'effort de réfléchir.

Lorsque nous recherchons, ou supprimons un élément de notre TreeSet<Plat>, il utilise la méthode `compare()` qu'on a défini pour trier l'ensemble, pour savoir si deux Plats sont identiques. Effectivement, la documentation de `compare()` signale que 0 doit être renvoyé lorsque 2 éléments sont identiques… Or nous on a trié uniquement sur le niveau de difficulté !

Modifions donc notre méthode `compare()` pour trier par niveau de difficulté, et ensuite par nom. Deux plats seront donc identiques si leur niveau de difficulté est identique, et si leur nom est identique.

Une fois la modification effectuée, vérifiez que tout est rentré dans l'ordre.

3.6 Améliorer le Livre

Ajoutez les méthodes suivantes dans la classe `Livre` :

```
/**
 * Renvoie un ensemble contenant tous les plats d'un certain type.
 * L'ensemble n'est pas modifable.
 * @param type le type de plats souhaité
 * @return l'ensemble des plats
 */
public SortedSet<Plat> getPlatsParType(Plat.Type type) {
    // L'ensemble renvoyé ne doit pas être modifiable !
    return null;
}

/**
 * Renvoie true si le livre contient le plat passé en paramètre. False
sinon.
 * Pour cette recherche, un plat est identique à un autre si son type, son
niveau de
 * difficulté et son nom sont identiques.
 * @param plat le plat à rechercher
 * @return true si le livre contient le plat, false sinon.
 */
public boolean contient(Plat plat) {
    // Ne pas utiliser 2 fois la méthode get() de la map, et ne pas
déclarer de variable locale !
    return false;
}

/**
 * Renvoie un ensemble contenant tous les plats du livre. Ils ne doivent
pas être triés.
```

```
 * @return l'ensemble de tous les plats du livre.
 */
public Set<Plat> tousLesPlats() {
    // Ne pas utiliser la méthode keySet() ou entrySet() ici !
    return null;
}
```

Complétez ces méthodes.

N'hésitez pas à modifier votre `main()` pour les tester.

3.7 Améliorer l'énuméré Type de plat

Lors de l'affichage du livre, on affiche directement le nom de l'énuméré. Il est donc affiché en majuscules.

Modifiez l'énuméré pour lui ajouter un attribut nom, avec son getter, mais pas de setter. Initialisez chaque énuméré avec son nom correct : Entrée, Plat, Dessert.

Adaptez la méthode toString() de la classe Livre pour qu'elle affiche correctement le nom.

3.8 Challenge optionnel

3.8.1 StringBuilder

Renseignez-vous sur la classe `StringBuilder`, et utilisez-la à bon escient dans la méthode `toString()` de la classe `Livre`.

3.8.2 Trier tous les plats

Dans l'énoncé, on vous demande de ne pas trier les plats de la méthode `tousLesPlats()`. Pour cet exercice, on vous demande de quand même trier cet ensemble, en faisant en sorte qu'ils soient triés par Type, puis par Niveau de difficulté, et enfin par Nom.

Exemple : les entrées en premier, et les entrées sont triées par niveau de difficulté ascendant, et pour les plus simples par exemple, on les trie par ordre alphabétique de nom.

3.8.3 Map triée ?

On souhaiterait faire en sorte que lorsqu'on appelle la méthode `keySet()` de la `Map` des plats de la classe `Livre`, on obtienne les clefs dans un ordre bien précis. A savoir, par ordre de service dans un restaurant. D'abord les entrées, puis les plats, puis les desserts.

Modifiez votre code pour arriver à ce résultat.

Astuces :

- Il faut modifier le type de l'attribut plats de la classe `Livre`.
- Il faut trouver une solution pour que les Types puissent être triés.