Java Avancé

Atelier 2 : énumérés, collections

Table des matières

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 Package

Pour la théorie sur le concept de package, référencez-vous à ce qui est donné au cours de Conception Orienté Objet.

Pour créer un package avec IntelliJ, faites un clic droit sur le dossier source devant le contenir, sélectionnez New → Package, entrez le nom du package et appuyez sur enter. Un nouveau dossier apparaît ayant comme nom celui du package créé. Pour créer une classe dans ce package, faite un clic droit sur le dossier du package et sélectionnez New → Java Class.

2.2 Énumérés

2.2.1 Définition d'un énuméré en java

Un énuméré est en fait une classe particulière qui permet de définir une énumération c.-à-d. un nouveau type de données ayant un nombre fini de valeurs qui sont connues à l'avance. Par exemple, on peut définir un énuméré pour les jours de la semaine comme suit :

```
//Le mot clé enum indique qu'il s'agit d'un énuméré
public enum JourDeLaSemaine {
    //déclaration de toutes les instances (constantes) de l'énuméré
    LUNDI,MARDI,MERCREDI,JEUDI,VENDREDI,SAMEDI,DIMANCHE;
}
```

Comme indiqué dans le commentaire, `LUNDI`, `MARDI`, …, `DIMANCHE` sont des constantes (`public static final`) de type `JourDeLaSemaine`. Comme il n'y rien de mis après le nom des constantes, elles sont initialisées en invoquant le constructeur sans paramètre (comme pour une autre classe, ce dernier existe tant qu'on ne met pas explicitement mis un constructeur mais, dans le cas d'un énuméré, il est privé). Il ne sera pas possible de créer d'autres objets de type `JourDeLaSemaine`.

Remarque : Il est obligatoire que la déclaration des constantes se fasse comme dans l'exemple ci-dessus c.-à-d. que :

- Elle doit se situer juste après l'en-tête.
- Les constantes sont séparées par une virgule.
- Un point-virgule est mis pour indiquer qu'il n'y a plus d'autres constantes

Comme dit précédemment, un énuméré est une classe. On peut donc y définir des attributs, des constructeurs, des méthodes , … comme dans une autre classe. Ajoutons à notre énuméré

`JourDeLaSemaine` deux attributs (le premier correspondra à un numéro de jour et le deuxième au nom du jour en anglais), un constructeur prenant en paramètres les valeurs pour ces deux attributs et des getters.

```
public enum JourDeLaSemaine {

    LUNDI(1,"monday"),MARDI(2,"tuesday"),MERCREDI(3,"wednesday"),
    JEUDI(4,"thursday"), VENDREDI(5,"friday"),SAMEDI(6,"saturday"),
    DIMANCHE(7,"sunday");

    private int numeroJour;
    private String nomAnglais;
    JourDeLaSemaine(int numeroJour, String nomAnglais) {
        this.numeroJour = numeroJour;
        this.nomAnglais = nomAnglais;
    }

    public int getNumeroJour() {
        return numeroJour;
    }

    public String getNomAnglais() {
        return nomAnglais;
    }
}
```

Dans l'exemple ci-dessus, on voit que les constantes sont toujours notées en premier lieu. Cependant, il y a maintenant des valeurs mises entre parenthèses après chaque constante. Ces valeurs correspondent aux valeurs à passer en paramètre au constructeur invoqué pour initialiser les constantes. Dans le cas ci-dessus, cela indique donc que, pour initialiser les constantes, il faut utiliser un constructeur qui prend deux paramètres.

Pour le reste, on voit que cela se fait comme dans une classe habituelle avec cependant des restrictions :

- Les constructeurs doivent être `private`. C'est la raison pour laquelle il n'est pas nécessaire de préciser la visibilité car, dans un énuméré, ils sont, par défaut, `private`.
- Il est interdit de mettre des méthodes abstraites.
- Il est interdit de redéfinir les méthodes `equals`/`hashCode` dans un énuméré. La comparaison se fait toujours sur la référence. Par conséquent, on peut utiliser == pour comparer deux instances d'un énuméré

Remarque : pour comparer des instances d'un énuméré, on peut utiliser ==

2.2.2 Méthodes d'un énuméré

Tout énuméré hérite de la classe `Enum`. Ce qui ajoute des méthodes et modifie le comportement de certaines par rapport à la classe `Object`.

Les méthodes statiques ci-dessous sont définies pour tous les énumérés :

- `values()` qui renvoie un tableau contenant toutes les constantes, dans l'ordre de leur déclaration, de l'énuméré.
- `valueOf(String)` qui renvoie, si elle existe, la constante ayant pour nom la valeur passée en paramètre et lance une `IllegalArgumentException` sinon.

Les méthodes d'instance suivantes sont aussi définies pour tout énuméré :

- `ordinal()` qui renvoie un `int` correspondant au numéro d'ordre de la constante dans la déclaration (la première ayant le numéro 0)
- `compareTo(XXX)`, où `XXX` correspond au type de l'énuméré, qui repose sur l'ordre dans lequel les constantes ont été déclarées. Avec notre exemple, `JourDeLaSemaine.LUNDI.compareTo(JourDeLaSemaine.JEUDI)` renverra un négatif. Cette méthode ne peut pas non plus être redéfinie.

De plus, dans la classe `Enum`, la méthode `toString()` a été redéfinie afin de renvoyer le nom de la constante sur laquelle on l'appelle. Avec notre exemple, `JourDeLaSemaine.JEUDI.toString()` renverra "JEUDI".

2.2.3 Switch … case

Le switch peut se faire pour des énumérés. Considérons la classe `Date` ci-dessous

```
public class Date {
    private JourDeLaSemaine jourDeLaSemaine;
    …

    public Date (JourDeLaSemaine jourDeLaSemaine, …){
        this.jourDeLaSemaine = jourDeLaSemaine;
        …
    }
    public JourDeLaSemaine getJourDeLaSemaine() {
        return jourDeLaSemaine;
    }
    …
}
```

Si `date` est une variable de type `Date`, on peut donc écrire le code suivant pour décider du traitement à faire selon le jour de la semaine :

```
switch (date.getJourDeLaSemaine()) {
    case LUNDI,MARDI,JEUDI,VENDREDI :
        …
        break;
    case MERCREDI:
        …
        break;
    case SAMEDI,DIMANCHE :
        …
        break;
}
```

2.3 Collections

La théorie nécessaire se trouve dans le fichier collections.pdf

2.4 Classes internes

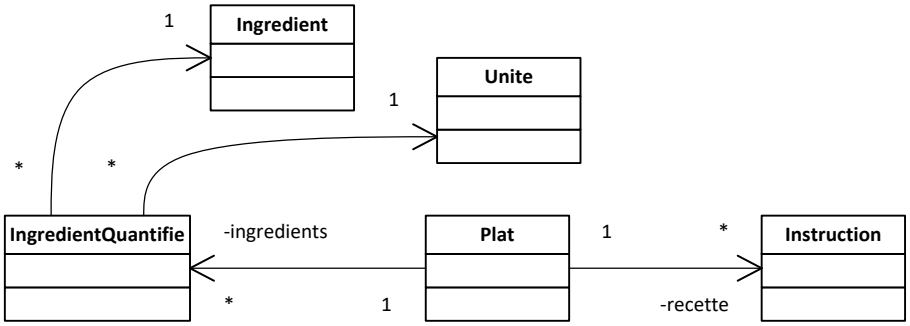
La théorie nécessaire se trouve dans le fichier classesInternes.pdf

3 Exercices

3.1 Introduction

Il s'agit d'implémenter une application permettant de gérer des recettes de cuisine. L'application doit permettre de gérer les instructions et les ingrédients de la recette.

Ci-dessous, se trouve une ébauche du diagramme de classes de l'application :



3.2 Consignes

Vous allez devoir implémenter les classes non fournies du diagramme ci-dessus. Tous les constructeurs et méthodes doivent tester leurs arguments. En cas d'erreur, une exception de type `IllegalArgumentException` se produit. Afin de réaliser ces tests, utilisez l'interface `Util` fournie. Les seules méthodes/constructeurs qui peuvent ne pas tester leurs arguments sont celles pour lesquelles cela n'a aucune importance. Soyez attentifs !

Dans IntelliJ, créez un projet intitulé AJ_atelier02. Récupérez les classes `Ingredient`, `IngredientQuantifie` et l'interface `Util` qui se trouvent sur moodle et mettez-les dans les bons packages adéquats (observez le code afin de trouver les bons package). Il est normal d'avoir une erreur de compilation car il manque l'énuméré `Unite`.

3.3 Énuméré Unite

Commencez par créer, dans le package domaine, l'énuméré `Unite`**.** Celui-ci doit contenir les constantes `GRAMME`, `KILOGRAMME`, `LITRE`, `MILLILITRE`, `CENTILITRE`, `DECILITRE`, `CUILLER_A_CAFE`, `CUILLER_A_THE`, `CUILLER_A_DESSERT`, `CUILLER_A_SOUPE`, `PINCEE`, `UN_PEU` et `NEANT`. Elle contient de plus un champ `abreviation` qui est initialisé dans le constructeur. Les abréviations des constantes sont dans l'ordre : `"gr"`, `"kg"`, `"l"`, `"ml"`, `"cl"`, `"dl"`, `"cc"`, `"ct"`, `"cd"`, `"cs"`, `"pincée"`, `"peu"`, `""`. On y redéfinit la méthode `toString` afin qu'elle renvoie l'abréviation.

3.4 La classe Instruction

Créez, dans le package domaine, la classe `Instruction`. Celle-ci garde deux attributs : `description` de type `String` et `dureeEnMinutes` de type `Duration` qui représente le temps nécessaire en minutes (éventuellement 0) pour la réalisation de cette instruction. Elle possède un constructeur prenant en paramètres une `String` pour la description et un `int` pour la durée en minutes dans cet ordre et fournit tous les getters et setters possibles. Elle redéfinit la méthode `toString` afin de renvoyer la description de l'instruction précédée de sa durée entre parenthèses (cf. l'exemple de sortie fourni dans le fichier sortie.txt). Dans un premier temps, ne vous souciez pas de faire apparaître les heures et les minutes de la durée sur 2 caractères.

3.5 La classe Plat

La classe `Plat` définit en interne deux énumérés : `Difficulte` et `Cout`.

L'énuméré `Difficulte` stocke les constantes `X`, `XX`, `XXX`, `XXXX` et `XXXXX` et ne garde aucun attribut. On y redéfinit `toString` afin qu'à l'affichage de ces constantes, on ait des `*` en place des `X` comme vous le constatez dans le fichier de sortie.

L'énuméré **Cout** définit les constantes `$`, `$$`, `$$$`, `$$$$` et `$$$$$`**.** Bien entendu, on s'arrange pour avoir en sortie des **€** au lieu de **$**.

`Plat` stocke les champs suivants : `nom` (`String`), `nbPersonnes` (`int`), `niveauDeDifficulte` (`Difficulte`), `cout` (`Cout`) et `dureeEnMinutes` (`Duration`). De plus, elle garde deux collections (respectez bien les rôles présentés dans le diagramme de classes) :

- La première, de type `List`, gardant toutes les instructions de la recette. Le type `List` a été choisi afin de pouvoir facilement garder les instructions dans l'ordre (séquentiel) dans lequel il faut les exécuter.
- La deuxième, de type `Set`, gardant tous les ingrédients quantifiés de la recette. Ici, l'ordre des ingrédients quantifiés n'a pas d'importance. C'est pourquoi on peut utiliser le type `Set`.

Le constructeur de `Plat` prend en paramètres le `nom`, le `nbPersonnes`, le `niveauDeDifficulte` et le `cout` dans cet ordre. La durée en minute est initialisée à une `Duration` de 0 minute. Elle sera mise à jour dans les méthodes de gestion des instructions (ajout, suppression, …)

La classe `Plat` fournit des getters triviaux pour le `nom`, le `nbPersonnes`, le `niveauDeDifficulte`, le `cout` et la `dureeEnMinutes`. Il n'y a pas de setter.

La gestion des instructions se fait via les méthodes publiques suivantes (Les méthodes ayant une position en paramètre lance une `IllegalArgumentException` si la position passée est inférieure ou égal à 0 ou est trop grande par rapport au nombre d'instructions déjà présentes)

- ```
  public void insererInstruction(int position, Instruction instruction)
  //insère l'instruction à la position précisée, position commençant à 1.
  ```
- ```
  public void ajouterInstruction (Instruction instruction)
  //ajoute l'instruction en dernier.
  ```
- ```
  public Instruction remplacerInstruction (int position, Instruction instruction)
  //remplace l'instruction à la position précisée par celle en paramètre.
  //renvoie l'instruction qui a été remplacée
  ```
- ```
  public Instruction supprimerInstruction (int position)
  //supprimer l'instruction à la position précisée
  //renvoie l'instruction qui a été supprimée
  ```

La consultation des instructions peut se faire via la méthode :

- ```
  public Iterator<Instruction> instructions()
  //fournit un itérateur d'instructions ne permettant pas de supprimer des
  //instructions (la méthode remove de l'itérateur renvoyé doit lancer une
  UnsupportedOperationException)
  ```

La gestion des ingrédients quantifiés se fait via les méthodes :

- ```
  public boolean ajouterIngredient(Ingredient ingredient, int quantite, Unite
  unite)
  //crée un IngrédientQuantifie et l'ajoute si l'ingrédient n'est pas encore
  //présent. Cela renvoie false si l'ingrédient est déjà présent.
  ```
- ```
  public boolean  ajouterIngredient(Ingredient ingredient, int quantite)
  //idem précédente.
  //l'unité mise par défaut est NEANT
  ```
- ```
  public boolean modifierIngredient(Ingredient ingredient, int quantite, Unite
  unite)
  //modifie l'unité et la quantité de l'ingrédient quantifié correpondant
  // à l'ingrédient passé en paramètre.
  //renvoie false si l'ingredient n'est pas présent.
  ```
- ```
  public boolean supprimerIngredient(Ingredient ingredient)
  //supprime l'ingrédient quantifié correspondant à l'ingrédient passé en
  //paramètre.
  //renvoie false si l'ingredient n'est pas présent
  ```
- ```
  public IngredientQuantifie trouverIngredientQuantifie(Ingredient ingredient)
  ```

```
//renvoie l'ingrédient quantifié correspondant à l'ingrédient
//renvoie null si l'ingredient n'est pas présent
```

Ajoutez la méthode `toString` fournie dans le fichier toString.txt dans la classe `Plat`.

On vous fournit une classe `Main` que vous devez copier dans votre projet (à vous de voir dans quel package la mettre) et exécuter. La sortie de ce programme doit être identique identique à celle fournie dans le fichier sortie.txt (excepté l'ordre des ingrédients qui peut varier).

3.6 Challenge optionnel

3.6.1 Création d'un SortedSet en utilisant un Comparator

Ajoutez, dans la classe `Plat`, la méthode :

- ```
  public SortedSet<Ingredient> ingredients()
  // renvoie l'ensemble des ingrédients, utilisés dans le plat, triés par nom
  //d'ingrédient.
  ```

Remarque : essayez de le faire avec une classe anonyme pour définir le `Comparator`.

3.6.2 Formatage de l'affichage des durées

- Dans la méthode `toString` de la classe `Instruction`, utilisez `String.format` afin d'avoir la durée au format HH:mm
- Dans la méthode `toString` de la classe `Plat`, utilisez `String.format` afin d'avoir la durée au format _ h __ m (les minutes sont toujours mises sur 2 caractères)
  ```