Sandwicherie
Devoir Maison de Modélisation

Groupe *B*
Romain Campillo, Tony Nguyen
https://github.com/tony-nguyen1/sandwicherie
L2 informatique
Faculté des Sciences
Université de Montpellier.

April 22, 2022

Sommaire

1 Sandwich générique

1.1 Généricité

Listing 1: Entête de class Sandwich

```
public class Sandwich <P extends Pain, S extends Sauce, G extends Garniture>
```

Sandwich est une classe nécessairement paramétrée par le type de pain utilisé, le type de sauce et le type des autres ingrédients à l'intérieur du sandwich.

Création d'un sandwich en 2 temps

Instanciation d'un sandwich Lors de la création du Sandwich, les attributs monPain et maSauce sont initialisés avec les objets passés en paramètre du constructeur, et l'attribut garniture est initialisé par une liste chaînée vide utilisée comme une pile.

Le pain et la sauce doivent être de type P et S respectivement ou les étendre.

Remplissage d'un sandwich Après la création d'un sandwich, l'utilisateur manipulant nos classes doit faire attention à remplir correctement les sandwichs. Un sandwich doit être composé de Garniture présente seulement chez cette instance là et pas dans un autre sandwich. Une instance d'une Garniture représente cet aliment avec une quantité égale à 1. Ainsi, si nous voulons 2 tomates dans notre sandwich, il faut 2 instances de la garniture tomate.

Les garnitures ajoutées doivent être de type G ou l'étendre.

Déplacer un ingrédient d'un sandwich à un autre

Puits de donnée Il serait souhaitable que l'on puisse mettre de la garniture végétarienne dans un sandwich acceptant toute garniture (cas n°1).

Mais l'inverse n'est pas vrai (cas n°2), nous ne voulons *PAS* mettre une garniture quelconque dans un sandwich composé de garniture végétarienne seulement. Car dans le sandwichA, nous ne sommes pas garantis que la garniture que nous allons déplacer est végétarienne.

Pour achever cela, nous avons utilisé "? super G" comme type du puits.

Listing 2: Déplacer l'ingrédient d'indice i depuis this aux puits

```
public void deplacerIngredientVers(Sandwich< ?, ?, ? super G>
 puits, int i) {
    G uneGarniture = this.getNthGarniture(i);
    this.garniture.removeLastOccurrence(uneGarniture);
    puits.ajouterIngredient(uneGarniture);
    }
```

Listing 3: Exemple puis de donnée

```
Sandwich<Pain, Sauce, Garniture> sandwichA;
Sandwich<Pain, Sauce, GarnitureVege> sandwichB;

// on les instancie correctement
// on remplit les sandwichs

// cas 1, ce qu'on veut
sandwichB.deplacerIngredientVers(sandwichA, 0);
// cas 2, ce qu'on ne veut pas
sandwichA.deplacerIngredientVers(sandwichB, 0);
```

Listing 4: Déplacer l'ingrédient d'indice i du puits jusqu'à this

```
public void deplacerIngredientDepuis(
 Sandwich< ?, ?, ? extends G> source, int i) {
    G uneGarniture = source.garniture.get(i);
    source.garniture.removeLastOccurrence(uneGarniture);
    this.ajouterIngredient(uneGarniture);
    }
```

Source de donnée C'est la même chose mais dans l'autre sens. Du coup, nous avons utilisé "? extends G".

2 Les aliments

Aliment Dans nos sandwichs, il y a : du pain, de la sauce et éventuellement de la garniture. Ils ont un parent abstrait (Aliment) pour factoriser un peu de code.

Hiérarchie Le pain le plus général est Pain (il inclut le pain avec de la viande). Ses enfants sont plus spécialisés.

Donc si on veut un pain plus précis il faut *étendre* Pain. Si on veut une sauce plus particulière, il faut *étendre* Sauce. Si on veut une garniture moins orthodoxe, il faut *étendre* Garniture.

Si veut qu'un pain/une sauce/une garniture soit végétarienne il faut *implémenter* l'interface estVegetarien.

Si veut qu'un pain/une sauce/une garniture soit vegan il faut *implémenter* l'interface estVegan et étendre la version végétarienne.

Constructeurs et nom

Puisque la classe Aliment possède un attribut "nom" qui est une chaine de caractères, rien n'empêcherait à l'utilisateur d'écrire *GarnitureVegan("poulet", 100)*.

Pour empêcher cela et "sécuriser" la saisie des aliments, toutes les sous-classes d'Aliment sont associées à une énumération précise.

Ainsi, pour initialiser une garniture Vegan, il faut préciser en paramètre l'énumération qui correspond à la GarnitureVegan.

Listing 5: Initialisation GarnitureVegan

```
new GarnitureVegan(NomGarnitureVegan.Cornichon, 10f)
```

Etant donné qu'une énumération ne peut pas hériter d'une autre énumération, un appel à super() n'est pas envisageable car le constructeur du parent ne prend pas en paramètre la même énumération.

Pour contourner ce problème, nous avons décidé d'utiliser 2 constructeurs pour chaque type d'aliment :

- Un constructeur public prenant en paramètre l'énumération correcte

- Un constructeur protected, destiné aux appels à super() des classes filles.

Le constructeur publique transforme l'objet de l'énumération en objet String et appelle super(String nomAliment, float kcal), ce qui est autorisé puisque les constructeurs prenant en paramètre les chaines de caractères sont protected. De cette manière le nom de l'aliment remontera jusqu'à la classe Aliment en évitant les incohérences.

Exception Tous les constructeurs peuvent renvoyer une *OutOfRangeKilocariesException* exception. Cela vient de l'appel à *setKilocalories* dans la classe Aliment, qui vérifie si le nombre de kilocalories entré est compris entre 0 et 1000 et lance une erreur le cas échéant.

3 Les autres sandwichs

3.1 Sandwich végétarien

Listing 6: Entête de la classe SandwichVege

```
public class SandwichVege<P extends PainVege,S extends SauceVege,
 G extends GarnitureVege> extends Sandwich<P,S,G>
```

La classe représentant les sandwich végétariens (SandwichVege) hérite du sandwich de base (Sandwich) et est elle-même paramétrée par des classes implémentant forcément l'interface estVegetarien.

PainVege, SauceVege et GarnitureVege implémentent estVegetarien et héritent de Pain, Sauce et Garniture respectivement.

3.2 Sandwich végan

Listing 7: Entête de la classe SandwichVegan

```
public class SandwichVegan<P extends PainVegan, S extends SauceVegan,
 G extends GarnitureVegan> extends SandwichVege<P, S, G>
```

La classe représentant les sandwich vegan (SandwichVegan) hérite du sandwich végétarien (SandwichVege) et est elle-même paramétreée par des classes implémentant forcément l'interface estVegan.

PainVegan, SauceVegan et GarnitureVegan implémentent estVegan et héritent de PainVege, SauceVege et GarnitureVege respectivement.
L'interface estVegan hérite de estVegetarien.

4 Iterator et Stream

4.1 L'itérateur

SandwichIterator implements Iterator<Aliment> Nos sandwichs peuvent avoir une quantité de garniture *sans restriction.*

Avec un attribut privé dans l'itérateur, la fonction next() retourne le pain en premier, puis la sauce. Ensuite, nous réutilisons l'Iterator de notre collection contenant notre garniture (pas besoin de reécrire l'Iterator pour une LinkedList alors qu'il existe déjà).

4.2 Le flot

Le stream nous envoie les aliments du sandwich. La fonction max() les compare pour nous et garde le plus grand d'après le Comparator donné en argument.

Comparator C'est un petit robot qui compare pour nous deux aliments entre eux par rapport à leur valeur énergétique. Il prend les kilocalories de chacun, fait la soustraction et renvoie une valeur. Cette valeur nous indique si o1 est supérieur, inférieur ou égal à o2.