Java Avancé

# Atelier 1 : Rappel orienté objet – partie 1

## Table des matières

# 1 Objectifs

Cette séance a pour but de rafraichir les concepts orientés Objet abordés dans le cadre du cours d'Analyse et Programmation Orientée Objet du bloc1.

# 2 Concepts

Les thèmes abordés sont : attributs de classe et d'instance, `ArrayList`, égalité, constructeurs (chaîne des constructeurs), encapsulation, exceptions (checked et unchecked), héritage, interface, dates, `Iterator`, dynamic binding, polymorphisme, overriding, …

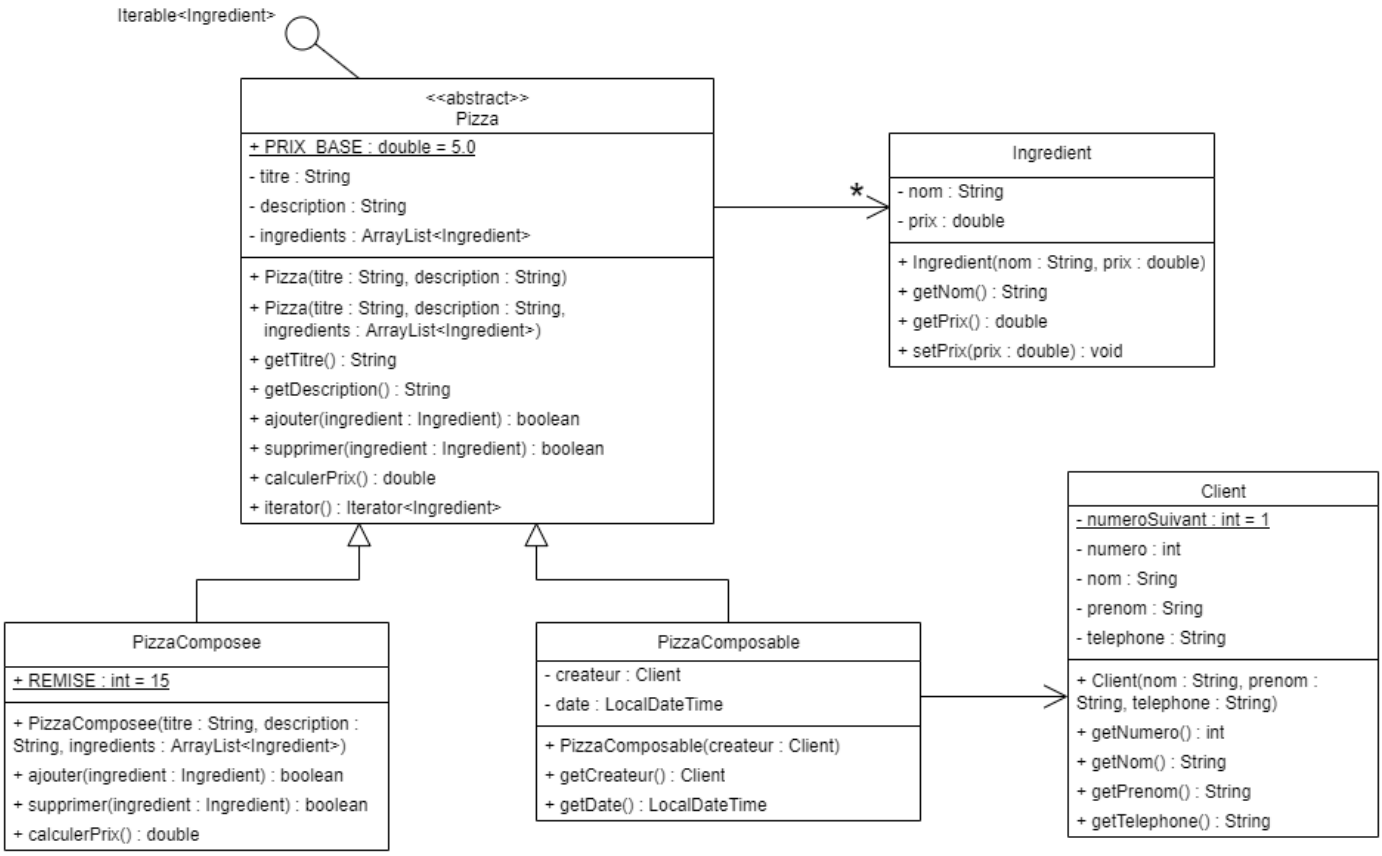
# 3 Exercice

## 3.1 Introduction

Il s'agit d'implémenter une application de gestion de commandes de pizzas : la pizzeria en ligne.

L'application doit permettre de gérer les commandes de pizzas effectuées par des clients. Les pizzas sont composées soit par la pizzeria (ces pizzas sont appelées pizzas composés), soit par le client selon ses goûts (ces dernières sont appelées pizzas composables).

Ci-dessous, se trouve une ébauche du diagramme de classes de l'application. Les classes qui s'y trouvent sont volontairement incomplètes et il y manque encore des classes.



## 3.2 Consignes

Vous allez devoir implémenter les classes présentées ci-dessus en tenant compte des commentaires ainsi que répondre à diverses questions.

Dans IntelliJ, créez un projet intitulé **AJ_atelier01**. Vous pouvez mettre vos classes directement dans le répertoire src de ce projet.

Votre code doit exploiter au mieux les richesses d'une découpe orientée Objet :

- Les copier-coller sont formellement interdits.
- Chaque attribut est encapsulé.
- Chaque objet est responsable de ses propriétés (état et comportement).
- Le code est réutilisable, lisible et bien structuré (indenté).

Remarque : dans un premier temps, traitez uniquement les cas d'exception demandés.

## 3.3 Les classes Ingredient et Client

**Question 1** : Pour rappel, un objet est une instance d'une classe. La classe correspond au type de l'objet. En UML, la représentation d'une classe est divisée en trois parties. À quoi correspondent ces parties ?

**Question 2** : Il est important qu'une classe veille à l'encapsulation. En quoi consiste l'encapsulation ?

**Question 3** : Que signifie le fait que la variable `numeroSuivant` soit soulignée ? Comment traduit-on cela en java ?

Implémentez les classes `Ingredient` et `Client` conformément au diagramme de classes. Le numéro du client doit lui être attribué automatiquement par classe (le premier créé recevant le numéro 1, le deuxième le numéro 2, …). Pour ce faire, utilisez la variable `numeroSuivant`.

## 3.4 Égalités référentielles et structurelles

L'égalité référentielle consiste à comparer les références mémoires : deux objets sont considérés comme égaux s'ils ont la même référence mémoire. En java, on utilise le == pour voir si deux objets sont égaux.

Dans les faits, on a souvent envie de considérer que deux objets sont égaux s'ils possèdent les mêmes valeurs pour certains de leurs attributs. Par exemple, si deux clients ont le même numéro, on veut qu'ils soient considérés comme égaux. C'est ce qu'on appelle l'égalité structurelle.

**Question 4** : en Java, que faut-il faire pour définir l'égalité structurelle pour une classe ? Comment teste-t-on l'égalité structurelle entre deux objets.

Complétez votre implémentation des classes `Ingredient` et `Client` pour faire en sorte qu'un ingrédient soit identifié par son nom et un client par son numéro.

## 3.5 La classe Pizza

Comme indiquée sur le diagramme, la classe Pizza doit être abstraite.

**Question 5** : quel est l'intérêt d'une classe abstraite ?

**Question 6** : La classe pizza possède deux constructeurs. Comment s'appelle le fait d'avoir deux constructeurs ou d'avoir deux méthodes de même nom ?

**Question 7** : pour éviter de recopier le code commun, on peut, dans un constructeur d'une classe, invoquer un autre constructeur de la classe. Comment fait-on cela en java ?

Implémentez la classe `Pizza` en tenant compte des remarques suivantes :

- Le deuxième constructeur doit invoquer le premier.
- Une pizza ne peut pas contenir deux fois le même ingrédient. Par conséquent :
    - Dans le constructeur recevant une ArrayList en paramètre, il faudra créer une nouvelle ArrayList pour l'attribut et copier tous les ingrédients de l'ArrayList en paramètre dans la nouvelle. Si un ingrédient se trouve deux fois dans l'ArrayList en paramètre, il faut lancer une `IllegalArgumentException` avec comme message "Il ne peut pas y avoir deux fois le même ingrédient dans une pizza"
    - La méthode ajouter doit échouer si on essaie d'ajouter un ingrédient qui se trouve déjà dans la pizza.
- La méthode supprimer échoue si l'ingrédient n'est pas dans la pizza.
- Le prix total est égal au prix de base auquel on ajoute la somme des prix des ingrédients.

## 3.6 La sous-classe PizzaComposee

À sa création, une pizza composée reçoit son titre, sa description et sa liste d'ingrédients. La première chose à faire dans son constructeur est d'invoquer le constructeur de sa classe parent.

**Question 8** : Comment invoque-t-on le constructeur de la classe parent dans le constructeur d'une sous-classe.

Pour une pizza composée, il n'est pas autorisé de lui ajouter ou de lui supprimer un ingrédient. Afin d'empêcher ces opérations, les méthodes ont été remises dans la sous-classe `PizzaComposee`. Dans la sous-classe, ces méthodes se contenteront de lancer une `UnsupportedOperationException` (c'est une classe java héritant de `RuntimeException`) avec comme message "Les ingrédients d'une pizza composée ne peuvent pas être modifiés".

**Question 9** : Comment appelle-t-on le fait de remettre dans une sous-classe une méthode afin d'en changer le comportement ?

**Question 10** : `UnsupportedOperationException` est-elle une checked ou une unchecked exception ?

**Question 11** : Quelles sont les différences entre les checked et les unchecked exceptions ?

Le prix d'une pizza composée est donné par le prix comme calculé précédemment sur lequel une remise de 15% est octroyée (arrondi à l'entier supérieur). C'est pourquoi il faut réécrire la méthode `calculerPrix` dans la classe `PizzaComposee`. Il faudra, dans cette méthode, invoquer la méthode `calculerPrix` de sa classe parent (classe `Pizza`).

**Question 12** : en java, comment fait-on pour invoquer une méthode de la classe parent dans une sous-classe ?

Implémentez la classe `PizzaComposee` en tenant compte de ce qui a été dit ci-dessus.

## 3.7 La sous-classe PizzaComposable

À sa création, une pizza composable ne reçoit que le client qui la crée. Son titre doit être initialisé à la valeur « Pizza composable du client **XX**» (où **XX** doit être remplacé par le numéro du client), sa description à la valeur « Pizza de » suivi du prénom et du nom du client), sa date à la date courante et ses ingrédients à une lise vide.

Pour obtenir la date courante, il existe, dans la classe `LocalDateTime`, une méthode statique `now()` qui la renvoie.

**Question 13** : en java, comme invoque-t-on une méthode statique ?

Implémentez la classe `PizzaComposable` en tenant compte de ce qui a été dit ci-dessus.

## 3.8 Main

Récupérez, sur moodle, le fichier toSring_partie1.txt. Ce fichier contient les méthodes `toString` des classes `Pizza` et `PizzaComposable`. Copiez les `toString` dans les bonnes classes.

Récupérez, sur moodle, les fichiers Ingredients.java et MainPizza.java et copiez-les dans le répertoire src de votre projet.

L'interface `Ingredients` contient uniquement des constantes de type `Ingredient` pouvant être utilisées pour les pizzas.

Exécutez la classe `MainPizza`. Vérifiez que vous avez le bon affichage en comparant avec ce qui est mis dans le fichier affichage_MainPizza.txt.