Java Avancé

Atelier 1 : Rappel orienté objet – partie 2

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

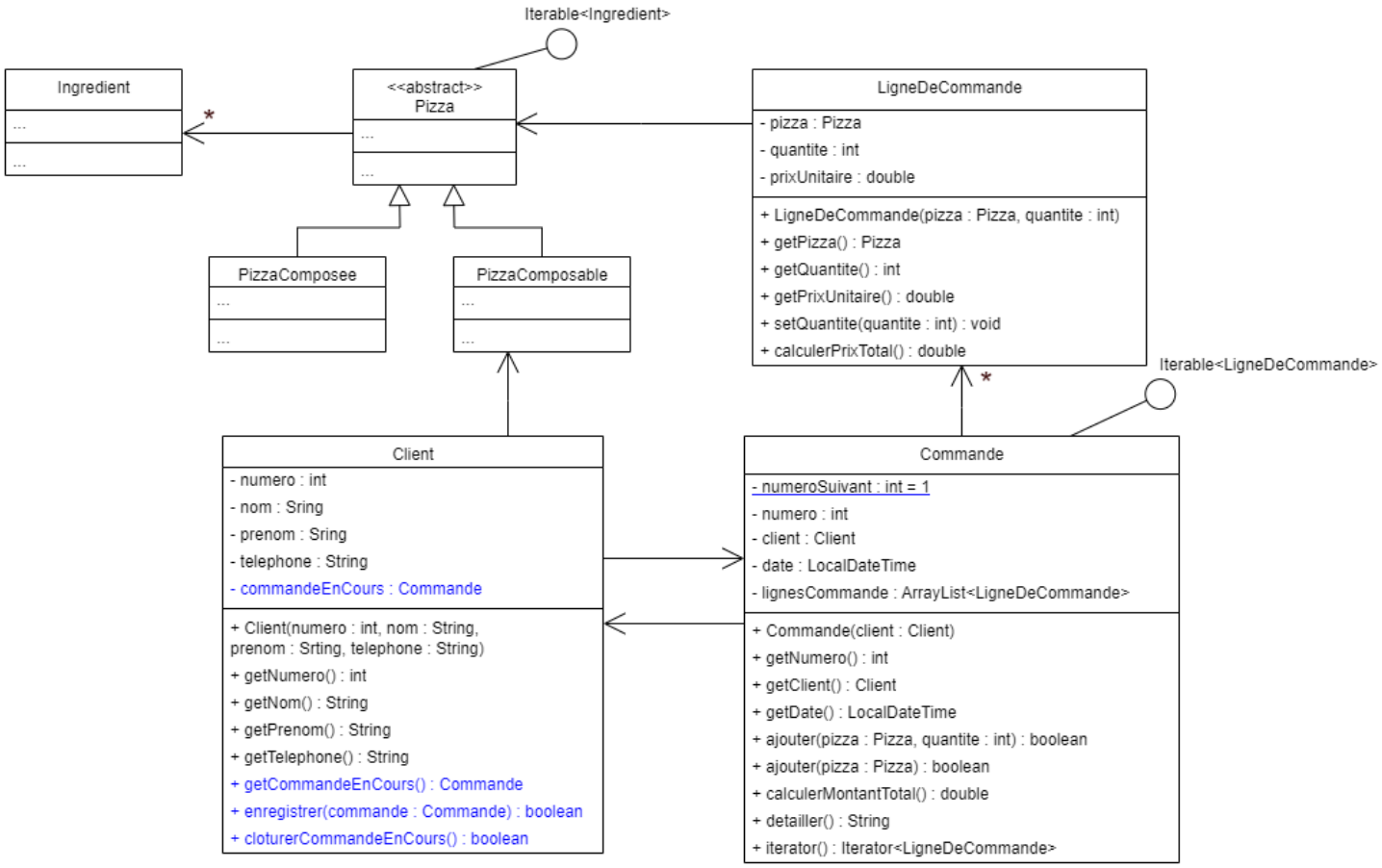
Les thèmes abordés sont : attributs de classe et d'instance, `ArrayList`, égalité, constructeurs (chaîne des constructeurs), encapsulation, exceptions (checked et unchecked), héritage, interface, dates, `Iterator`, dynamic binding, polymorphisme, overriding, …

3 Exercice

3.1 Introduction

Il s'agit de compléter l'application de gestion de commandes de pizzas entamée lors de la séance précédente. Si vous n'avez pas terminé la première partie, vous pouvez récupérer la solution qui se trouve sur moodle.

Ci-dessous, se trouve le diagramme de classes complété de l'application. Les classes qui s'y trouvent sont volontairement incomplètes. Dans la classe Client, ce qui a été mis en bleu est ce qu'il faut ajouter à celle-ci par rapport à la séance précédente.



3.2 Consignes

Comme à la séance précédente, vous allez devoir implémenter les classes présentées ci-dessus.

Repartez de votre solution ou de celle fournie sur moodle. Si vous utilisez la solution disponible sur moodle, vous devez l'ouvrir dans IntelliJ et éventuellement renseigner la SDK (pour cela, il faut aller dans File → Project Structure et choisir une SDK dans la liste déroulante dans la section Project SDK)

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 La classe LigneDeCommande

Implémentez la classe `LigneDeCommande` selon le diagramme de classes ci-dessus. Le prix unitaire représente le prix unitaire de la pizza. Il est gardé car, comme les prix peuvent varier au cours du temps, il est important de garder le prix au moment de la commande.

3.4 Associations entre Commande et Client

Quand on crée une commande pour un client, celle-ci doit automatiquement être enregistrée comme commande en cours de ce client. De ce fait, on ne peut pas créer une commande pour un client s'il a encore une commande en cours, ce qui donne des contraintes supplémentaires au niveau des associations.

Commencez par créer la classe `Commande`. Dans un premier temps, ne mettez que les attributs, les getters demandés et la méthode `iterator`.

Complétez ensuite la classe `Client` en ajoutant ce qui est mis en bleu dans le diagramme de classes et en tenant compte des remarques suivantes :

- La méthode `enregistrer` échoue s'il y a déjà une commande en cours ou si la commande passée en paramètre n'est pas une commande du client. Sinon, elle enregistre la commande passée en paramètre comme commande en cours.
- La méthode `cloturerCommandeEnCours` échoue s'il n'y a pas de commande en cours. Sinon, elle supprime la commande en cours.

Ajoutez maintenant le constructeur de la classe `Commande` en tenant compte des remarques suivantes :

- Le constructeur lance une `IllegalArgumentException` (avec comme message « impossible de créer une commande pour un client ayant encore une commande en cours») s'il n'est pas possible d'enregistrer une commande pour le client en paramètre. Sinon, il ne faut pas oublier d'enregistrer la commande du côté du client.
- Le numéro de la commande est attribué automatiquement par la classe (ka première commande créée doit avoir le numéro 1, la deuxième le numéro 2, …)
- La date doit être initialisée à celle de l'instant de la création.

3.5 Suite de la classe Commande

Ajoutez les méthodes manquantes de la classe Commande en tenant compte des remarques suivantes :

- Les méthodes ajouter échouent si la commande n'est pas la commande en cours du client. Sinon, s'il existe déjà une ligne de commande pour cette pizza, elle modifie cette ligne pour lui ajouter la quantité voulue. Sinon elle crée une nouvelle ligne de commande pour la pizza et l'ajoute.
- Pour la méthode ajouter ne recevant pas de quantité en paramètre, il faut ajouter une seule pizza.
- La méthode caculerMontantTotal calcule le montant total de la commande et le renvoie.
- La méthode détailler renvoie, sous forme de chaîne de caractères, toutes les lignes de la commande (une ligne de commande par ligne)

3.6 Égalité structurelle sur les pizzas

On veut que, par défaut, le titre de la pizza permette de l'identifier. Comme deux pizzas composables du même client ont le même titre, c'est, dans ce cas, le titre associé à la date de création qui permet de l'identifier. Ajoutez, dans les classes adéquates, les méthodes nécessaires pour définir ces égalités structurelles.

3.7 Main

Récupérez, sur moodle, le fichier toSring_partie2.txt. Ce fichier contient les méthodes `toString` des classes `LigneDeCommande` et `Commande`. Copiez les `toString` dans les bonnes classes.

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

L'interface MenuPïzzeria contient uniquement des constantes de type `PizzaComposee` correspondant aux différentes pizzas composées du menu de la pizzeria.

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

3.8 Parties optionnelles

3.8.1 Historique des commandes d'un client

On veut maintenant garder, pour un client, toutes ces commandes passées. Une commande va être ajoutée aux commandes passées d'un client au moment de sa clôture. De plus, il faut empêcher l'enregistrement d'une commande passée à un client. Il faut aussi pouvoir parcourir toutes les commandes passées du client (au moyen d'un foreach).

3.8.2 Possibilité de supprimer des pizzas d'une commande

Il faut maintenant donner la possibilité de retirer des pizzas de la commande. Pour cela, ajoutez, dans la classe `Commande` les méthodes suivantes :

- `public boolean retirer(Pizza pizza, int quantite)`
  Cette méthode échoue si la commande n'est pas en cours du client, s'il n'y a pas de ligne de commande concernant la pizza en paramètre dans la commande ou si la quantité de pizza à retirer est strictement supérieur à la quantité commandée. Sinon, si la quantité à retirer est strictement inférieure à la quantité commandée, il faut mettre à jour la quantité de la ligne de commande et, dans le cas où elle est égale, il faut supprimer la ligne de commande.
- `public boolean retirer(Pizza pizza)`
  Cette méthode fait la même chose que la méthode précédente avec une quantité à retirer de 1.
- `public boolean supprimer(Pizza pizza)`

Cette méthode échoue si la commande n'est pas en cours du client ou s'il n'y a pas de ligne de commande concernant la pizza en paramètre dans la commande. Sinon, elle retire la ligne de commande concernant la pizza passée en paramètre de la commande.

3.8.3 Test des paramètres des méthodes/constructeurs

Il faut ajouter les tests pour la validité des paramètres des méthodes/constructeur et lancer une `IllegalaArgumentException` en cas de paramètres invalides. Il faut refuser comme paramètre :

- La valeur `null` lorsqu'il s'agit d'un objet ;
- Une chaîne de caractères constituées uniquement de « blanc » ;
- Une liste d'ingrédients vide dans le constructeur de Pizza ayant une liste en paramètre ;
- Un prix inférieur ou égal à 0.
- Une quantité inférieure ou égale à 0

Vous pouvez utiliser l'interface `Util` qui offre des méthodes de vérifications pour les cas les plus courants.