TP : Atelier de construction d'une commande de café avec le pattern Builder en Java

Objectif : Mettre en pratique le pattern Builder en Java en construisant un système de gestion de commandes de café.

Description : Vous avez été chargé de développer un système de gestion de commandes de café pour un café en ligne. Le système doit permettre aux clients de créer des commandes de café personnalisées en spécifiant les types de café, les options de lait, de sucre, et d'autres extras. Utilisez le pattern Builder pour implémenter la création de commandes de café de manière flexible et maintenable.

Étape 1 : Création de la classe CommandeCafe

1. Créer une **classe CommandeCafe** qui représente une commande de café.
2. Ajouter les **attributs** nécessaires pour stocker les options d'une commande :
    - Type de café
    - Type de lait (facultatif)
    - Quantité de sucre
    - Options supplémentaires (ex : chantilly, sirop caramel…)
3. Rendre le **constructeur privé** pour forcer l'utilisation du Builder.
4. Ajouter une **méthode afficherCommande()** qui affiche les détails d'une commande de café.

Étape 2 : Implémentation du Pattern Builder

1. Ajouter une **classe interne statique ConstructeurCommande** à CommandeCafe.
2. Définir les **mêmes attributs** que dans CommandeCafe, en ajoutant des valeurs par défaut si nécessaire.
3. Ajouter un **constructeur prenant en paramètre le type de café** (obligatoire).
4. Ajouter des **méthodes de configuration** pour :
    - Définir le type de lait
    - Définir la quantité de sucre
    - Ajouter des options supplémentaires (chantilly, sirop caramel…)
5. Implémenter une **méthode construire()** qui retourne une instance de CommandeCafe.

Étape 3 : Création et utilisation du Builder

1. Dans une **classe Main**, créer plusieurs **instances de CommandeCafe** en utilisant le Builder.
2. Construire plusieurs **exemples de commandes** :
    - Une commande de café noir **sans sucre ni lait**.
    - Une commande de café avec **lait et sucre**.
    - Une commande de café **gourmand avec chantilly et sirop caramel**.
3. Appeler la **méthode afficherCommande()** pour chaque commande créée.

Étape 4 : Affichage des commandes

1. Vérifier que chaque **commande est correctement affichée** avec toutes les options définies.
2. Tester l'affichage d'une **commande sans options** (ex : café noir simple).
3. Vérifier que les options facultatives **n'apparaissent pas** si elles ne sont pas définies.

Étape 5 : Ajout de tests supplémentaires

1. Vérifier que l'on peut créer une **commande avec uniquement un type de café**.
2. Tester une **commande avec toutes les options activées**.
3. Tester une **commande avec seulement certaines options activées**.

4. Ajouter une **validation** pour interdire une commande incohérente (ex : chantilly sans lait).
5. Ajouter un **message d'erreur** dans construire() si une option invalide est demandée.

Étape 6 : Implémentation d'un système de commande de thé

Ajoutez une nouvelle classe CommandeThe et son **Builder** pour permettre aux clients de commander du thé avec différentes options.

1. **Créer une nouvelle classe CommandeThe** qui représente une commande de thé avec les attributs suivants :

   - **Type de thé** (Thé vert, noir, blanc, rooibos…)
   - **Type de lait** (optionnel)
   - **Quantité de sucre**
   - **Ajout de miel** (oui/non)
   - **Ajout de citron** (oui/non)

2. **Créer une classe interne ConstructeurCommandeThe** dans CommandeThe :

   - Ajouter un constructeur prenant en paramètre le **type de thé** (obligatoire).
   - Ajouter des **méthodes fluentes** pour configurer les options (ex: definirLait(), ajouterMiel(), etc.).
   - Ajouter une **méthode construire()** pour retourner une instance de CommandeThe.

3. **Dans la classe Main**, créer plusieurs **exemples de commandes de thé** :

   - Un thé simple **sans sucre ni lait**.
   - Un thé au lait avec **miel** et **sucre**.
   - Un thé citronné **sans lait mais avec du sucre**.

4. **Afficher les commandes de thé** avec une méthode afficherCommande().
5. **Tester les configurations possibles** et ajouter des **validations** pour éviter les erreurs (ex: interdire miel + citron dans la même boisson si nécessaire).