Rapport de Refactoring et Améliorations du Code

**Introduction**

Dans le cadre de ce projet, j'ai entrepris un important travail de nettoyage et d'amélioration du code existant. L'objectif était d'optimiser la structure du code, d'ajouter des tests et d'introduire une nouvelle fonctionnalité permettant de stocker les informations soit en JSON soit en CSV. Ce rapport détaille les modifications et améliorations effectuées.

**Problèmes Identifiés dans le Code Initial**

Le code initial était monolithique, avec toutes les fonctionnalités regroupées dans une seule classe Main. Cela rendait le code difficile à maintenir et à tester. Parmi les principaux problèmes relevés :

* **Absence de modularité** : Une seule classe gérait à la fois la lecture des fichiers, la gestion des commandes et la manipulation des données.
* **Difficultés de testabilité** : Il était difficile de tester isolément chaque fonctionnalité.
* **Gestion du format de fichier rigide** : Seul le format JSON était pris en charge, et la structure des données était rudimentaire ([Milk :10]).
* **Suppression des éléments imprécise** : L'approche initiale pour la suppression des éléments utilisait contains() au lieu de vérifier l'égalité stricte des noms.

**Refactoring et Nouvelles Améliorations**

J'ai décomposé le code en plusieurs classes, chacune ayant une responsabilité claire.

**1. Classe Main**

* La classe Main est maintenant uniquement responsable du lancement de l'application.
* Elle délègue l'exécution des commandes à CommandExecutor.

**2. Classe Item**

* Introduction d'une classe Item pour modéliser les éléments de la liste de courses.
* Ajout d'un constructeur avec annotations @JsonCreator et @JsonProperty pour faciliter la conversion JSON.
* Ajout de la méthode toCSV() pour convertir un objet Item en ligne CSV.

**3. Classe GroceryListManager**

* Cette classe gère le chargement et l'enregistrement des éléments.
* Prise en charge des formats JSON et CSV.
* En CSV :
  + Les colonnes sont séparées par ,.
  + La première ligne contient les noms des colonnes.
* En JSON :
  + Structure des données améliorée sous forme d'un tableau d'objets { "name": "Milk", "quantity": 10 }.

**4. Classe CommandParser**

* Responsable de la gestion des arguments en ligne de commande.
* Ajout de la gestion du paramètre --format pour déterminer si l'on doit utiliser JSON ou CSV.

**5. Classe CommandExecutor**

* Cette classe exécute les commandes add, remove et list.
* **Modification de la suppression** : Avant, la suppression était effectuée en vérifiant si la chaîne contenait le nom de l'élément (contains()). Maintenant, on vérifie si le nom correspond exactement (equalsIgnoreCase()).
* **Ajout d'un feedback utilisateur** : Indique si l'élément à supprimer était présent ou non.

**Tests et Fiabilité**

J'ai ajouté des tests unitaires pour vérifier les principales fonctionnalités :

* **Tests de parsing des arguments** (CommandParser)
* **Tests de chargement et sauvegarde JSON/CSV** (GroceryListManager)
* **Tests des commandes add, remove et list** (CommandExecutor)
* **Tests sur la cohérence des données**

Difficultés

Lors de l’utilisation de Git, nous avons été confrontés à un problème majeur : certains fichiers que nous avions push sur le dépôt apparaissaient vides ou incomplets. Cette anomalie s’est produite des deux côtés, et malgré notre maîtrise de Git, nous n’avons pas encore identifié la cause exacte du problème.

Malgré cette difficulté, nous avons réussi à finaliser notre travail sans utiliser Git. Nous espérons que cette situation ne se reproduira pas à l’avenir.

**Conclusion**

Le code a été réorganisé pour être plus modulaire, testable et extensible. Le format CSV a été ajouté et la suppression des éléments a été améliorée. Ce travail de refactoring garantit une meilleure maintenabilité et une plus grande flexibilité pour d'éventuelles évolutions futures.