Rapport - Application de Liste d'Épicerie

Ce que fait le projet

Notre projet est une application simple de liste d'épicerie en ligne de commande qui permet de gérer les articles à acheter. L'utilisateur peut ajouter des articles, consulter sa liste et supprimer des articles qu'il ne souhaite plus acheter.

L'application stocke la liste d'épicerie dans un fichier JSON ou CSV, ce qui permet de conserver les informations même après avoir fermé le programme. C'est pratique pour maintenir une liste permanente qui peut être mise à jour au fil du temps.

Fonctionnalités principales

Ajouter un article

Voir la liste

Supprimer un article

Réduire la quantité

Comment nous avons structuré notre code

Pour organiser notre code, nous avons créé plusieurs classes qui ont chacune un rôle spécifique :

GroceryItem

Cette classe représente un article dans notre liste d'épicerie. Chaque article a :

- Un nom (comme "Lait" ou "Pain")
- Une quantité (comme 2 bouteilles ou 1 baguette)

C'est la base de notre application, car tout tourne autour des articles qu'on veut acheter.

Gestion des fichiers

Pour stocker notre liste, nous avons créé :

- Une interface **FileFormat** qui définit comment lire et écrire la liste
- **JsonFormat** pour sauvegarder en JSON (le format principal)
- **CsvFormat** pour sauvegarder en CSV (au cas où nous voudrions exporter la liste)
- Une classe **File** qui gère les opérations sur les fichiers

Cette structure nous permet de facilement ajouter d'autres formats si besoin.

Actions sur la liste

Pour manipuler la liste, nous avons créé :

- **AddElement** pour gérer l'ajout d'articles
- **RemoveElement** pour gérer la suppression d'articles
- **GererListe** qui coordonne toutes les opérations et fait le lien entre l'interface utilisateur et le stockage

Interface utilisateur

La classe **Main** s'occupe d'interpréter les commandes de l'utilisateur et d'appeler les bonnes méthodes dans GererListe.

Les défis que nous avons rencontrés

1. Gestion des doublons

Au début, quand nous ajoutions deux fois le même article (par exemple "Lait"), nous obtenions deux entrées séparées dans notre liste. Nous avons résolu ce problème en utilisant une Map avec le nom de l'article comme clé, ce qui garantit l'unicité.

2. Formats de fichier

Nous voulions que l'application soit flexible concernant le format de stockage. Créer une interface FileFormat nous a permis d'implémenter facilement différents formats sans changer le reste du code.

3. Tests unitaires

Écrire des tests pour les opérations sur les fichiers était compliqué car il fallait simuler un système de fichiers. Nous avons utilisé @TempDir de JUnit pour créer des fichiers temporaires pendant les tests.

Ce que nous avons appris

1. **L'importance de la séparation des responsabilités** : Avoir des classes distinctes pour chaque aspect du programme rend le code plus clair et plus facile à modifier.

2. **Les avantages des interfaces** : Utiliser des interfaces comme FileFormat nous a permis de changer facilement l'implémentation sans toucher au reste du code.

3. **Tests unitaires** : Nous avons appris à tester chaque composant séparément, ce qui nous a aidés à trouver des bugs que nous n'aurions pas vus autrement.

Améliorations futures ?

Pour vérifier les doublons et prévenir des erreurs volontaires (ou involontaires) des utilisateurs, on a pensé à rajouter un système de comparaison avec les noms qui utilisera convertira la chaîne d'entrée en majuscules ou en minuscules et compareras avec les clés (en majuscule ou en minuscule aussi) afin de mieux prévenir des doublons.

Des catégories pour organiser les articles (produits laitiers, fruits, etc.).

Étendre la lecture de fichier a d’autre type de document comme un tableur excel par exemple.