Saad GUESSOUS
Jules DUCANGE

Compte Rendu de TP : C++ n°3 : Gestion des entrées / sorties

Binôme B3307 : Saad GUESSOUS et Jules DUCANGE

Introduction

L'objectif de ce TP est de compléter l'application développée lors du TP précédent, afin de pouvoir assurer la gestion des entrées / sorties. L'utilisateur doit pouvoir sauvegarder le catalogue dans un fichier, mais aussi restituer un catalogue déjà sauvegardé. Ce TP comprend 4 scénarios différents de sauvegarde/restitution, tous les scénarios ont été implémentés.

Format du fichier

Nous avons décidé d'opter pour un format de fichier simple et très intuitif :

- Un fichier est composé d'un ou plusieurs trajets.
- Chaque trajet est identifiable grâce à l'abréviation de son type (Composé : `TC`, Simple : `TS`)
- Ensuite, chaque trajet contient les informations permettant de l'identifier.

Pour un trajet simple :

Sur les 3 lignes successives on retrouve les informations suivantes :

- La ville de départ
- La ville d'arrivée
- Le moyen de transport

Pour un trajet composé :

On retrouve respectivement après le `TC` :

- Le nombre de sous-trajets (simples ou composés) composant le trajet en question.
- Une succession de ces trajets, représentés avec le même format défini.

Exemple du fichier demo.txt

```
TS
Lyon
Bordeaux
Train
TC
2
TS
Lyon
Marseille
Bateau
```

```
TS
Marseille
Paris
Avion
TS
Lyon
Paris
Auto
```

Version commentée

Version annotée, un commentaire est indiqué par un `#`.

```
TS        #    Trajet simple
Lyon      #    Ville de départ
Bordeaux  #    Ville d'arrivée
Train     #    Moyen de transport
TC        #    Trajet Composé
2         #    Nombre de sous-trajets
TS        #    Premier sous-trajet (Simple)
Lyon
Marseille
Bateau
TS        #    Deuxième sous-trajet (Simple)
Marseille
Paris
Avion
TS        #    Trajet simple
Lyon
Paris
Auto
          #    Ligne vide (résidu de l'export)
```

> **Remarque** : Il est possible que le fichier contienne ou non une ligne vide à la fin du fichier. Cela n'affecte pas l'importation.

Scénarios de sauvegarde/restitution :

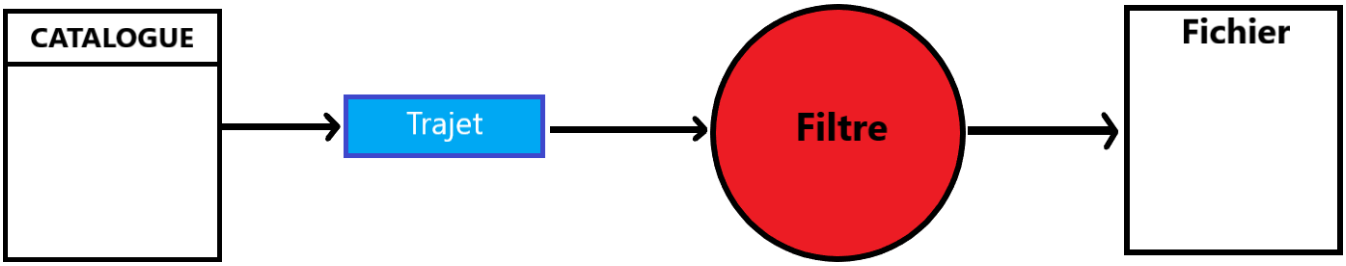
La sauvegarde du Catalogue courant dans un fichier, et le chargement d'un catalogue de trajets pourront se faire selon quatre scénarios. Ces scénarios sont non composables dans l'application actuelle.

- Sans critère de sélection
- Selon le type de trajet (`TS` ou `TC`)
- Selon la ville de départ et / ou d'arrivée du trajet
- Selon une sélection de trajets

Ces scénarios ont été développés sous forme de `Filtres`, qui sont tout simplement des classes.
Chaque filtre est donc implémenté par une classe, contenant une méthode `bool Valide(Trajet*)` permettant d'envoyer un booléen valant `True` si le trajet satisfait ce filtre.

Il suffit donc avec cette implémentation de parcourir le Catalogue de trajets (ou le fichier) et de soumettre chaque trajet au filtre choisi par l'utilisateur, et selon le résultat obtenu, on le sauvegarde (ou restitue) ou non.

Schéma expliquant le fonctionnement des filtres



Fonctionnement des filtres pour une sauvegarde

Explication détaillée du fonctionnement des filtres

Le fonctionnement des filtres repose sur le principe d'héritage. Il existe une classe mère `Filtre` qui implémente 2 méthodes : `SaisieUtilisateur` et `Valide`. Pour implémenter un filtre il suffit d'implémenter une classe qui hérite publiquement de `Filtre` et implémente les deux méthodes.

> La classe mère n'est pas virtuelle est peut-être utilisée comme un filtre qui ne retire aucun élément.

L'utilisation d'un filtre commence par son instanciation. Une fois instancié, celui-ci est passé à la fonction d'import ou d'export via un pointeur de type `Filtre`. Dans un premier temps c'est la méthode `SaisieUtilisateur` qui est appelée pour initialiser le filtre avec les données fournies par l'utilisateur.

Ensuite la méthode `Valide` est appelée pour chaque trajet qu'elle prend en paramètre. Elle retourne un booléen qui détermine si le trajet valide ou non le filtre.

> Cette méthode de filtrage permet de faciliter l'implémentation de nouveaux filtres ou la modification de filtres existant. Cependant cette méthode n'est pas parfaite d'un point de vue performance puisque lors de l'import chaque trajet doit-être reconstruit avant d'être filtré.

Gestion des erreurs

- Pour la restitution : si l'utilisateur rentre un nom de fichier n'existant pas ou ne pouvant être ouvert, une erreur sera affichée sur le flux de sortie standard.
- Pour le 4ème scénario : si l'utilisateur rentre un intervalle vide au sens mathématique, exemple : début de l'intervalle à 3 et fin de l'intervalle à 2, aucun trajet ne sera sauvegardé ou restitué. Dans tous les autres cas, tous les trajets inclus dans l'intervalle de l'utilisateur seront sauvegardés/restitués.

Conclusion

Problèmes rencontrés : Fuite de mémoire et filtres

Pour la restitution : Avant l'implémentation des filtres, chaque trajet lu dans le fichier était ajouté au catalogue. Avec l'ajout des filtres, certains trajets étaient lus, mais n'étaient toutefois pas ajoutés au catalogue. Cela causait donc des fuites de mémoires !

Dans la version finale, chaque trajet est instancié pour l'application du filtre. Si le trajet en question vérifie le filtre il est ajouté, sinon il est directement détruit.

Cette méthode n'est pas la plus économe car certains filtres pourraient être vérifiés facilement sans créer de nouvelles instances.

Cependant elle permet de créer de nouveaux filtres de manière complètement indépendante du mécanisme d'import ou d'export.

Axes d'évolution et d'amélioration

De par sa nature générique, notre système de filtres manque de flexibilité pour les opérations de masse. Même si le filtre utilisé ne sélectionne qu'un nombre précis (pouvant être très réduit) de voyages, l'application du filtre nécessite un passage à travers tout le catalogue, donc tous les trajets.

Cette limitation pourrait être corrigée via l'implémentation d'un système permettant d'arrêter le parcours duCatalogue, une piste serait d'utiliser le mécanisme d'exception du C++.