Auteurs : Frolin Balthazar – Wallyn Valentin

Compte-Rendu TP IFA-3-POO2-1

Application « Voyage Voyage » : Gestion des entrées/sorties

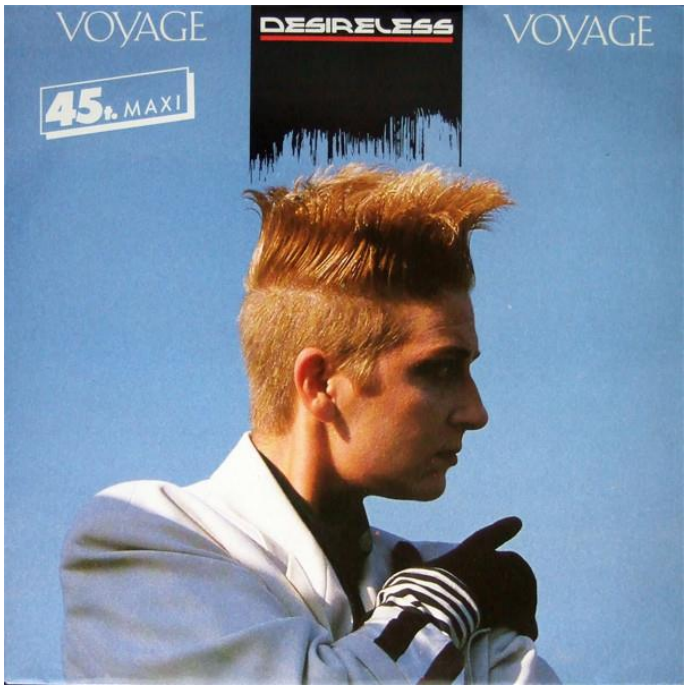

Table des matières

I. Contexte de l'application

Ce compte-rendu détaille la réalisation du TP POO2-1 en classe de 3IFA INSA de Lyon. Ce TP s'inscrit dans l'initiation des notions abordées en cours : gestion des entrées/sorties par l'utilisation de la STL en C++.

Dans ce cadre, l'application se propose de mettre en place la sauvegarde et le chargement de catalogue de trajets selon plusieurs critères, dans la continuité du TP POO1-2.

II. Description détaillée du format de fichier

II.1. Metadonnées

La première ligne du fichier continent des infos sur l'ensemble du catalogue. Ces infos sont séparées par un caractère séparateur arbitrairement choisi « | ». On y trouve dans l'ordre : le nombre de trajets simples, le nombre de trajets composées, une liste de toutes les villes de départ séparées par des « ; », une liste de toutes les villes d'arrivées séparées par des « ; ».

Exemple : `2|1|Lyon;|Bordeaux;Paris;`

Nous avons fait le choix d'ajouter cette ligne de métadonnées afin d'améliorer la performance de certains critères. Par exemple, il n'y pas a pas besoin de lire l'intégralité du fichier si le nombre de trajet simple vaut 0 et que l'on souhaite récupérer tous les trajets simples.

II.2. Trajet Simple

Les trajets simples sont représentés sur une ligne sous la forme suivante :

`Départ;Transport;Arrivée;`

Chaque champ est séparé par un « ; ». Les champs sont renseignés dans l'ordre suivant : ville de départ, moyen de transport, ville d'arrivée.

Nous avons choisi de représenter un moyen de transport par sa valeur entière dans l'énumération « MeansOfTransport ».

II.3. Trajet Composé

Les trajets composés n'étant qu'une imbrication de trajets simples, nous avons gardé le même formalisme que celui des trajets simples pour chaque étape du trajet composé.

La seule différence étant pour la première étape qui est suffixée par la ville d'arrivée du trajet composé séparée de la première étape par un « : ». Les étapes suivantes sont indentées d'une tabulation afin de pouvoir différencier un trajet simple d'une étape de trajet composé.

Nous avons conçu cette notation pour pouvoir par la suite créer des trajets composés composant d'autre trajets composés (trajets composés complexes). Ainsi il suffira de conserver le même formalisme avec une indentation supplémentaire.

Exemple :

```
Lyon;3;Marseille;:Paris
        Marseille;4;Paris;
```

II.4. Exemple complet de démonstration

Ainsi le fichier de sauvegarde correspondant au jeu d'essai de démonstration prend la forme suivante (demo.txt) :

```
2|1|Lyon;|Bordeaux;Paris;
Lyon;0;Bordeaux;
Lyon;1;Paris;
Lyon;2;Marseille;:Paris
        Marseille;3;Paris;
```

III. Spécifications des nouvelles fonctionnalités

III.1. Gestion des noms de fichiers

Les noms de fichiers sont choisis par l'utilisateur. L'utilisateur peut sauvegarder et charger n'importe quel fichier tant que celui-ci possède un chemin valide sous le système d'exploitation OpenSuse.

III.2. Cas limites

- Chargement
  - Fichier qui commence par « 0|0 » : pas de chargement
  - Fichier invalide ou inexistant : retour au menu principal
  - Les trajets déjà existants dans le catalogue ne sont pas ajoutés une seconde fois.
- Sauvegarde
  - Catalogue vide : pas de sauvegarde
  - Fichier déjà existant : demander confirmation à l'utilisateur pour écraser ou non le fichier
- Critères
  - Critère de type :
    - se base sur le type « racine » du trajet (ie. on n'extrait pas les trajets simples à l'intérieur d'un trajet composé par exemple)
  - Critère de ville :
    - non sensible à la casse (uppercase/lowercase). Les accents sont eux pris en compte et forment des noms de villes différents.
    - On peut filtrer selon une ville de départ, une ville d'arrivée, les deux ou rien.
      - Dans le cas d'une ville d'arrivée ou de départ non utilisée en filtre, sa valeur doit être '-'
      - Si les deux villes ne sont pas utilisées en filtre (les deux ont la valeur '-'), un critère vide est utilisé à la place (pas de filtre)
  - Critère d'intervalle [m,n] :
    - on commence à compter à 1
    - les nombres utilisés comme position/indice sont respectivement :
      - sauvegarde : le nombre affiché à l'interface lors du listing (correspondant à l'index du trajet dans la structure PathArray + 1)
      - chargement : le numéro de la ligne. Une nouvelle ligne est comptée lorsqu'elle ne commence pas par une tabulation (ie. la ligne représente un trajet. Un trajet composé est sauvegardé sur plusieurs lignes mais ne compte que pour une ligne dans notre définition). La première ligne de metadata ne compte pas.
    - Intervalle complétement en dehors (m et n supérieure à la taille du catalogue) : chargement = aucun trajet chargé / sauvegarde = sauvegarde vide

- Borne supérieure (n) en dehors : pas de limite de fin.
- m > n : On redemande à l'utilisateur. On doit avoir m <= n.
- m = n : sélection d'une seule ligne/trajet d'indice m/n.
- Borne inférieure (m) < 0. Equivalent à m = 1.
- m < 0 et n < 0. On redemande à l'utilisateur : l'intervalle n'est pas valide dans ce système.

IV. Description des classes

Une vision globale de l'application peut se faire à travers le diagramme de classes suivant. (Note : tous les membres publics y sont renseignés. En revanche, le diagramme ne présente pas de manière exhaustive les membres privés/protégés par soucis de concision.)

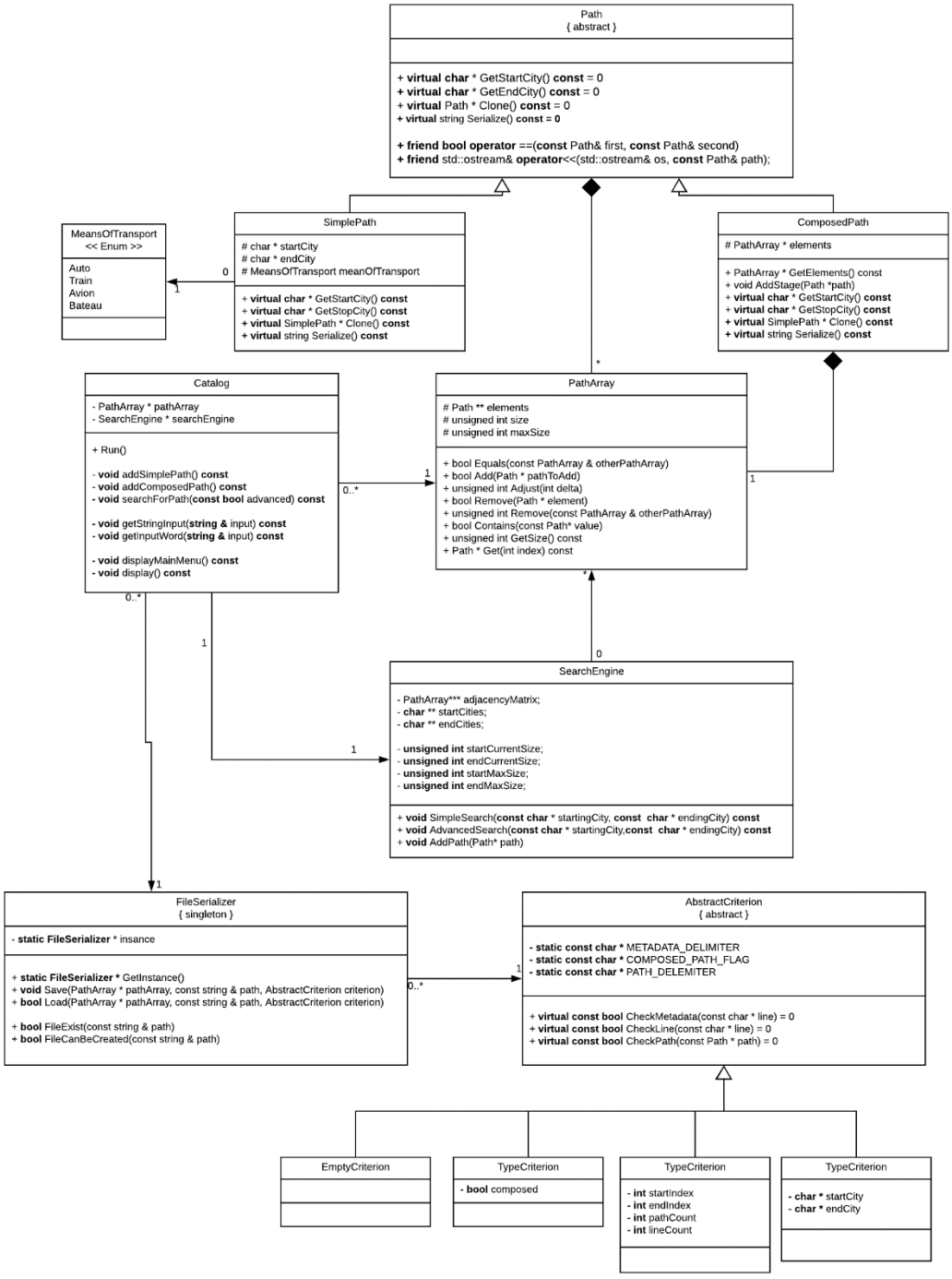

Figure 1 – Diagramme de classes de l'application VoyageVoyage

La classe FileSerializer est un Singleton permettant de gérer l'écriture et la lecture de fichiers de sauvegarde et offrant quelques fonctionnalités de gestion de fichiers. Combinée avec les classes dérivées d'AbstractCriterion, elle permet de sauvegarder ou charger tout ou partie d'un catalogue.

Code Source :

Le code source de VoyageVoyage se trouve dans le dossier « Project » .
Il est également accessible ici : https://github.com/Balthov60/TP2-CPP-Maranzana

Architecture du projet :

```
Project
    |
    | ---------- src
    |             | ---------- <Module>
    |             |                 | ---------- <Module>.h
    |             |                 | ---------- <Module>.cpp
    |             | ---------- Main.cpp
    |
    | ---------- test
    |             | ---------- <Module>
    |             |                 | ---------- T<Module>.h
    |             |                 | ---------- T<Module>.cpp
    |
    | ---------- .Build
    |             | ---------- <Module>.o
    |             | ---------- T<Module>                    << Test Executable
    |             | ---------- VoyageVoyage                 << Main Executable
    |
    | ---------- Makefile
```

Instructions d'utilisation

Les instructions d'utilisations sont également disponibles dans le fichier README.md

1. **Compilation**
   - Se placer dans le dossier « Project »
   - Exécuter « make init »
   - Compiler un exécutable :
     - Version de production : Exécuter « make »
     - Version de debug : Exécuter « make debug »
     - Version de test : Exécuter « make test T<Module> »
   - Nettoyer les fichiers de build :
     - Fichiers .o : Exécuter « make clean »
     - Fichiers .o et exécutable : Exécuter « make clean-all »
2. **Exécution**
   - Version de production/debug : Exécuter « .Build/VoyageVoyage »
   - Versions de test : Exécuter « .Build/T<Module> »

V. Conclusion

V.1. Problèmes rencontrés

Aucune difficulté particulière n'a été rencontrée. Avec l'expérience du TP précèdent l'application est mieux appropriée ce qui facilite le développement.

V.2. Améliorations possibles

Cette application est un exercice académique – il pourrait donc bien entendu être beaucoup plus développé dans ses fonctionnalités. Toutefois, nous avons repéré des points d'amélioration par rapport aux notions abordées qui auraient été intéressants à mettre en place si du temps supplémentaire était accordé à ce TP.

V.2.a. Combinaison des critères

Dans la poursuite de l'objectif de pratique des notions vues en cours, il serait intéressant de combiner les critères entre eux et de manipuler à ce titre des conteneurs de la STL (list, vector...). Exemple : sélectionner tous les trajets simples qui ont pour ville de départ X.

Cette fonctionnalité pourrait être implémentée en apportant quelques modifications à la classe Catalog gérant les interactions avec l'utilisateur ainsi qu'en prenant en compte n'ont plus un seul objet Criterion mais une liste d'objets dans la classe FileSerializer.

V.2.b. Critères supplémentaires

Il serait bien entendu envisageable de créer des critères supplémentaires afin d'étoffer l'application.

Exemples : Critère sur le nombre d'étapes d'un trajet composé, critère sur le moyen de transport utilisé...

L'architecture de notre application rend l'ajout d'un nouveau critère très rapide : il suffit d'implémenter une nouvelle classé héritant d'AbstractCriterion.

V.2.c. Sauvegarde de trajet composé complexe

Le format de fichier supporte les trajets composés complexes (des trajets composés, eux-mêmes composés de trajets composés). Mais le système de sauvegarde et de chargement des données ne supporte pas ce cas d'utilisation. Il serait donc intéressant de développer cette fonctionnalité probablement en utilisant des fonctions récursives.