Compte-Rendu TP IFA-3-POO2-1

Application « Voyage Voyage » : Gestion des entrées/sorties



Table des matières

[I. Contexte de l’application 4](#_Toc531711022)

[II. Description détaillée du format de fichier 4](#_Toc531711023)

[II.1. Metadata 4](#_Toc531711024)

[II.2. Trajet Simple 4](#_Toc531711025)

[II.3. Trajet Composé 4](#_Toc531711026)

[II.4. Exemple 4](#_Toc531711027)

[III. Spécifications des nouvelles fonctionnalités 5](#_Toc531711028)

[III.1. Gestion des noms de fichiers 5](#_Toc531711029)

[III.2. Cas limites 5](#_Toc531711030)

[IV. Description détaillée des classes 7](#_Toc531711031)

[V. Conclusion 10](#_Toc531711032)

[V.1. Problèmes rencontrés 10](#_Toc531711033)

[V.1.a. Makefile 10](#_Toc531711034)

[V.1.b. Gestion de la mémoire 10](#_Toc531711035)

[V.2. Améliorations possibles 10](#_Toc531711036)

[V.2.a. Makefile 10](#_Toc531711037)

[V.2.b. Gestion de la mémoire 10](#_Toc531711038)

[V.2.c. Sauvegarde de trajet composé complexe 11](#_Toc531711039)

[V.2.d. Liste de critères 11](#_Toc531711040)

1. 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.

1. Description détaillée du format de fichier
   1. Metadata

La première ligne du fichier continent des infos sur l’ensemble du catalogue. Ces infos sont séparé par des « | ». Vous y trouverez dans l’ordre : Le nombre de trajets simples, le nombre de trajets composées, une liste de toutes les villes de départ séparé par des « ; », une liste de toutes les villes d’arrivées séparé par des « ; ».  
  
Nous avons fait le choix d’ajouter cette ligne de metadata afin de ne pas avoir à lire l’intégralité du fichier lors de l’utilisation de critères.

* 1. Trajet Simple

Les trajets simples sont représentés sur une ligne. Chaque champ est séparé par un « ; ». Les champs sont renseignés dans l’ordre suivant : ville de départ, moyen de transport, ville d’arrivé.

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

* 1. 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 suffixé par la ville d’arrivé du trajet composé et séparé de la première étape par un « : ». Les étapes suivantes sont également 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.

* 1. Exemple

Contenu du fichier demo.txt :

2|1|Lyon;|Bordeaux;Paris;

Lyon;1;Bordeaux

Lyon;2;Paris

Lyon;3;Marseille:Paris

Marseille;4;Paris

1. Spécifications des nouvelles fonctionnalités
   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.

* 1. 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.

1. Description détaillée des classes

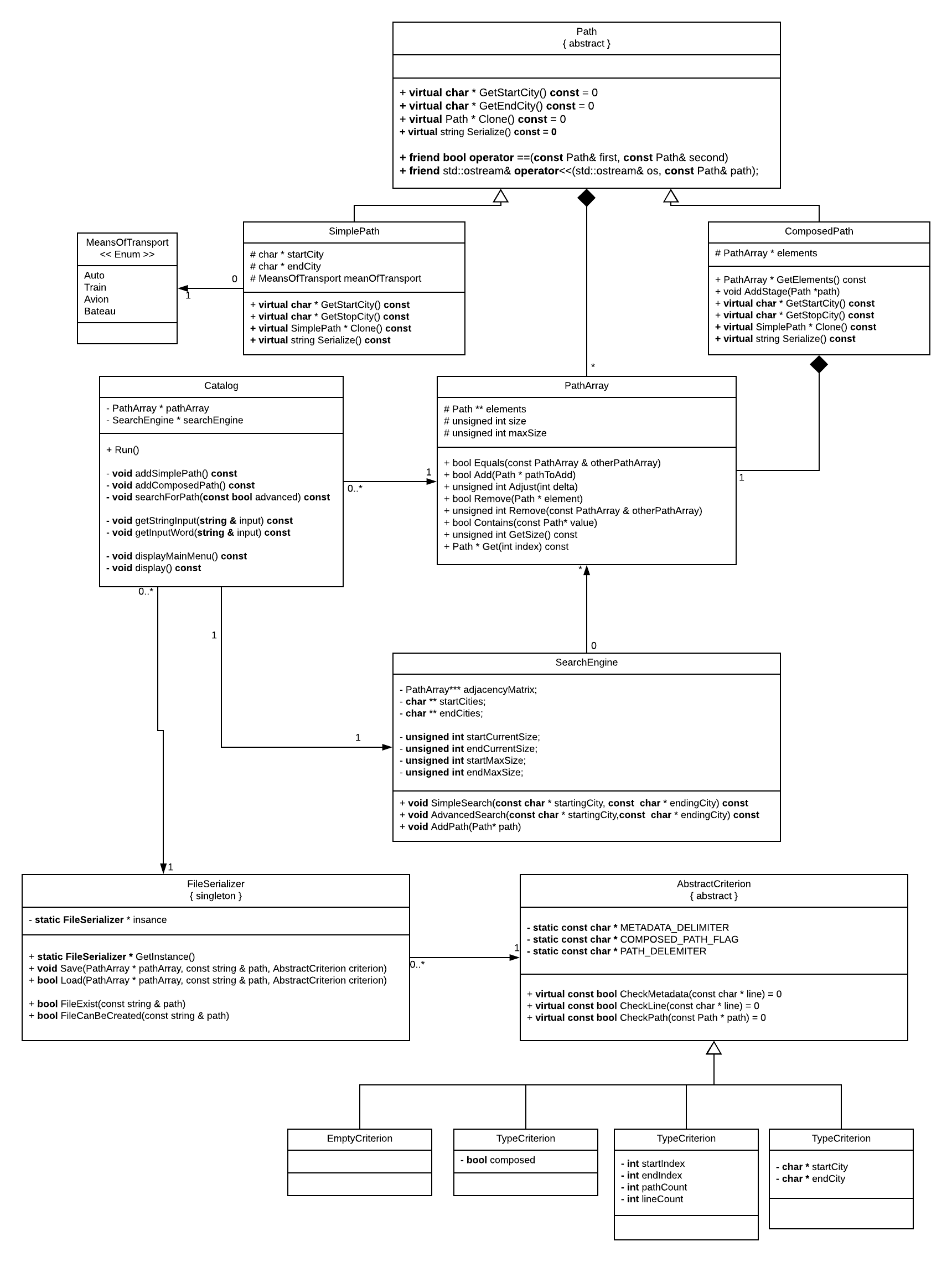
Une vision globale de l’application peut se faire à travers le diagramme de classes suivant. (Note : tous les membres publics y sont renseignés. En revanche, le diagramme ne présente pas de manière exhaustive les membres privés/protégés par soucis de concision.)

Figure 1 – Diagramme de classes de l’application VoyageVoyage

Le point d’entrée de l’application se fait au niveau de la méthode « Run() » de la classe Catalog. Cette classe gère l’interface utilisateur basée sur un menu console. En ce sens, elle possède un ensemble de méthodes permettant de gérer et de formater les entrées utilisateurs ainsi qu’un ensemble de méthodes liées à l’affichage du menu de l’application. Elle encapsule un « PathArray » qui stocke tous les trajets renseignés et un « SearchEngine » permettant d’effectuer des recherches de parcours. Les différents choix utilisateurs provoqueront les différentes manipulations possibles du catalogue courant : ajout d’un trajet, affichage du catalogue, recherche de parcours.

La gestion des trajets est inspirée d’un pattern Composite. En effet, on peut remarquer qu’un trajet composé peut lui-même être composé d’un trajet composé. De plus, la manipulation dans le PathArray devant être transparente entre trajet simple et composé, cette architecture basée sur une classe abstraite Path et deux classes concrètes SimplePath et ComposedPath représentant respectivement un trajet simple et un trajet composé s’est naturellement imposée. Les attributs « startCity » et « endCity » sont placés dans la classe SimplePath plutôt que la classe mère Path. Dans le cas contraire, comme ces informations peuvent être retrouvées dans les différents trajets du trajet composé, cela aurait causé une redondance de ces informations pour les instances de ComposedPath. De manière similaire, l’attribut représentant le moyen de transport n’a de sens que dans un trajet simple, les trajets composés pouvant être définis par plusieurs trajets aux moyens de transport différents.

La classe SearchEngine s’occupe de la recherche, notamment en implémentant deux algorithmes pour la recherche « simple » et la recherche « avancée ». Son rôle étant clairement défini et l’algorithme de recherche relativement lourd, c’est une entité propre détachée du catalogue (à l’inverse par exemple, de l’affichage).

La classe PathArray représente une collection de trajets manipulée en interne sous la forme d’un tableau dynamique. Les détails de l’implémentation peuvent se trouver dans la partie suivante.

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 fichier. Combiné 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 »

1. **Exécution**

* Version de production/debug : Exécuter « .Build/VoyageVoyage »
* Versions de test : Exécuter « .Build/T<Module> »

1. Conclusion
   1. Problèmes rencontrés
      1. Makefile

Lors de la création du Makefile, nous avons rencontré un problème de portée des variables. En effet, il nous a été impossible d’éditer des variables du Makefile depuis le code bash appelé dans les règles.

La variable était considérée comme « une commande » et nous n’avons trouvé aucune solution à ce problème. D’une manière générale, la création du Makefile générique a nécessité une part de travail importante.

* + 1. Gestion de la mémoire

La manipulation intensive des pointeurs et des objets a posé des problèmes de gestion de la mémoire. La rigueur nécessaire à cet exercice est nouvelle et bouscule les habitudes de programmation. A force d’essais/erreurs, nous sommes finalement arrivés à une solution satisfaisante (selon Valgrind). Cependant, avec le recul certains choix de gestion ne sont pas nécessairement très judicieux. Cela est discuté dans la partie suivante.

* 1. 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.

* + 1. Makefile

Deux améliorations pourraient être apportés au Makefile :

* Récupérer le nom des modules de manière dynamique en récupérant la liste des dossiers afin que le Makefile n’ai pas besoin d’être « configuré ».
* Compiler uniquement les fichiers nécessaires aux exécutables de test sans avoir à les spécifier dans les constantes du Makefile.
  + 1. Gestion de la mémoire

La solution finale ne possède pas de fuites mémoires, cependant la gestion de la mémoire peut être amélioré.

* Le PathArray pourrait effectuer une copie du trajet pointé lors de l’ajout afin de le rendre indépendant de la source du trajet. Cela serait primordial si cette classe était destinée à être utilisée en externe.
* Le SearchEngine pourrait ne pas dupliquer les trajets entiers et travailler directement sur les pointeurs. Il ne serait ainsi pas en charge de la destruction des trajets sur lesquels il travaille. Dans le cadre de l’apprentissage de la gestion de la mémoire, il était plus simple qu’il ne soit pas dépendant d’une autre classe pour la gestion de sa mémoire afin de simplifier les tests isolés et le repérage des fuites mémoires.
  + 1. Sauvegarde de trajet composé complexe

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

* + 1. Liste de critères

Actuellement lors de la sauvegarde ou lors du chargement d’un catalogue, il n’est pas possible de sélectionner plusieurs critères pour filtrer les résultats. Cette fonctionnalité pourrait être implémentée par la suite en apportant quelque 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.