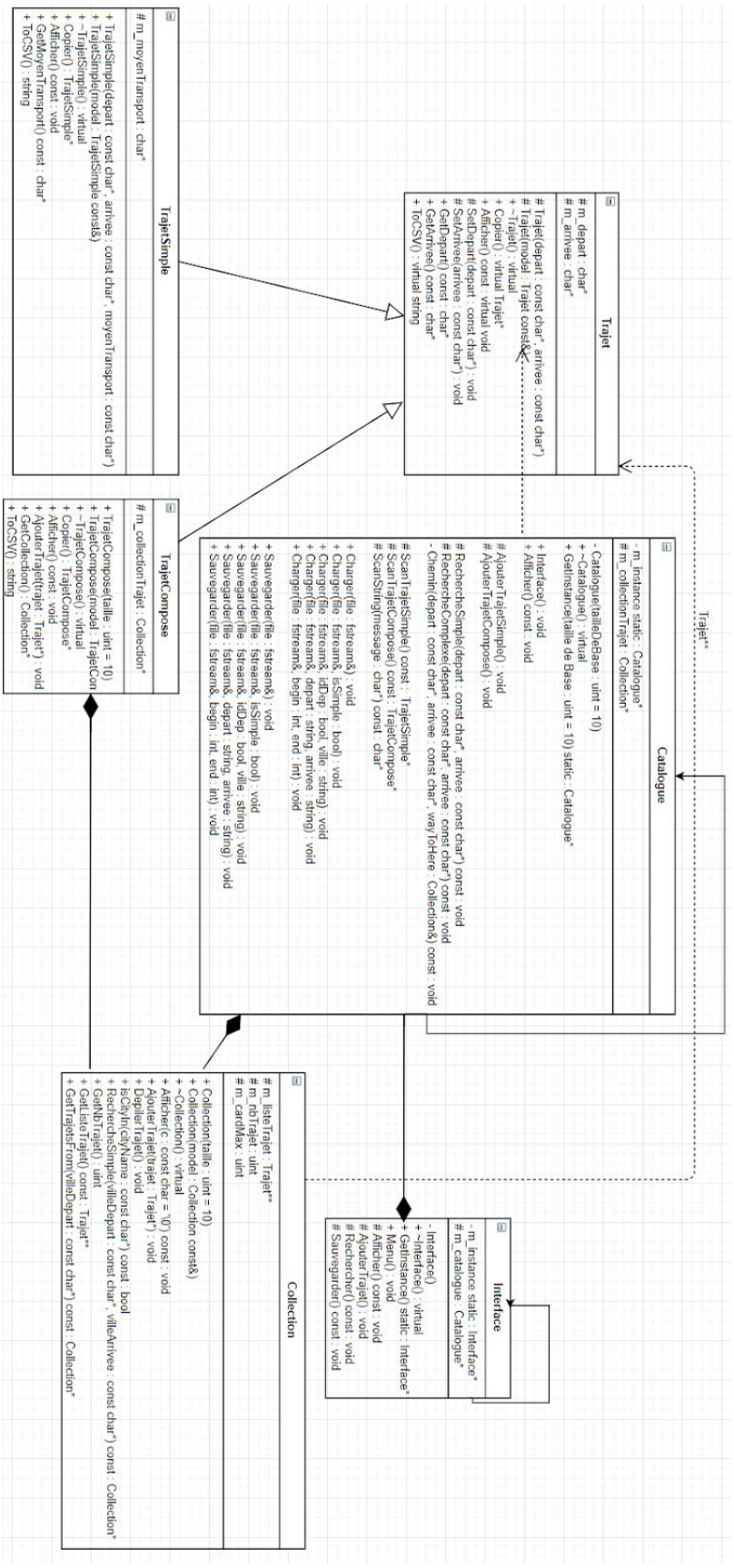
TP C++ n°3 : Gestion des entrées / sorties : Compte-Rendu :

I. Introduction

L'objectif de ce TP était de nous familiariser avec l'utilisation des classes de gestion des entrées / sorties en se basant sur le programme réalisé lors du précédent TP concernant les concepts d'héritage et de polymorphisme et avec leur expression en langage C++.
l'objectif principal de ce TP est l'utilisation des fichiers en écriture et en lecture pour sauvegarder et charger des collections d'objets hétérogènes.

Dans ce compte-rendu, on aura l'occasion de présenter la réponse que nous avons apporté au problème posé, ainsi que les problèmes que nous avons rencontré et les améliorations possibles.

II. UML


Trajet
m_depart : char*
m_arrivee : char*
Trajet(depart : const char*, arrivee : const char*)
Trajet(model : Trajet const&)
+ ~Trajet() : virtual
+ Copier() : virtual Trajet*
+ Afficher() const : virtual void
# SetDepart(depart : const char*) : void
# SetArrivee(arrivee : const char*) : void
+ GetDepart() const : char*
+ GetArrivee() const : char*
+ ToCSV() : virtual string
TrajetSimple
m_moyenTransport : char*
+ TrajetSimple(depart : const char*, arrivee : const char*, moyenTransport : const char*)
+ TrajetSimple(model : TrajetSimple const&)
+ ~TrajetSimple() : virtual
+ Copier() : TrajetSimple*
+ Afficher() const : void
+ GetMoyenTransport() const : char*
+ ToCSV() : string
TrajetCompose
m_collectionTrajet : Collection*
+ TrajetCompose(taille : uint = 10)
+ TrajetCompose(model : TrajetCon
+ ~TrajetCompose() : virtual
+ Copier() : TrajetCompose*
+ Afficher() const : void
+ AjouterTrajet(trajet : Trajet*) : void
+ GetCollection() : Collection*
+ ToCSV() : string
Catalogue
- m_instance static : Catalogue*
m_collectionTrajet : Collection*
- Catalogue(tailleDeBase : uint = 10)
+ ~Catalogue() : virtual
+ GetInstance(taille de Base : uint = 10) static : Catalogue*
+ Interface() : void
+ Afficher() const : void
# AjouterTrajetSimple() : void
# AjouterTrajetCompose() : void
# RechercheSimple(depart : const char*, arrivee : const char*) const : void
# RechercheComplexe(depart : const char*, arrivee : const char*) const : void
- Chemin(depart : const char*, arrivee : const char*, wayToHere : Collection&) const : void
ScanTrajetSimple() const : TrajetSimple*
ScanTrajetCompose() const : TrajetCompose*
ScanString(message : char*) const : char*
+ Charger(flle : fstream&) : void
+ Charger(flle : fstream&, isSimple : bool) : void
+ Charger(flle : fstream&, isSimple : bool, idDep : bool, ville : string) : void
+ Charger(flle : fstream&, depart : string, arrivee : string) : void
+ Charger(flle : fstream&, begin : int, end : int) : void
+ Sauvegarder(flle : fstream&) : void
+ Sauvegarder(flle : fstream&, isSimple : bool) : void
+ Sauvegarder(flle : fstream&, isSimple : bool, idDep : bool, ville : string) : void
+ Sauvegarder(flle : fstream&, depart : string, arrivee : string) : void
+ Sauvegarder(flle : fstream&, begin : int, end : int) : void
Interface
- m_instance static : Interface*
m_catalogue : Catalogue*
- Interface()
+ ~Interface() : virtual
+ GetInstance() static : Interface*
+ Menu() : void
# Afficher() const : void
# AjouterTrajet() : void
# Rechercher() const : void
# Sauvegarder() const : void
Collection
m_listeTrajet : Trajet**
m_nbTrajet : uint
m_cardMax : uint
+ Collection(taille : uint = 10)
+ Collection(model : Collection const&)
+ ~Collection() : virtual
+ Afficher(c : const char = 10) const : void
+ Ajouter(trajet : Trajet*) : void
+ Depiler(trajet) : void
+ isCityIn(cityName : const char*) const : bool
+ RechercheSimple(villeDepart : const char*, villeArrivee : const char*) const : Collection*
+ GetNbTrajet() : uint
+ GetListeTrajet() const : Trajet**
+ GetTrajetsFrom(villeDepart : const char*) const : Collection*
Trajet**

III. Description du fichier de sauvegarde / restitution

1. Contenu

Le fichier de sauvegarde / restitution contient plusieurs métadonnées permettant de décrire totalement les différents trajets en prenant en compte leur type (simple ou composé) :

Tout d'abord, sur la première ligne nous inscrivons / trouvons le nombre total de trajets contenus dans le catalogue. Cela facilite l'implémentation de différentes fonctionnalités comme par exemple la sauvegarde / restitution d'un intervalle de trajets.

Ensuite sur le reste des lignes, nous avons tout d'abord le nombre de sous-trajets composant le trajet. Ainsi, si ce nombre est égal à 1, il s'agit d'un trajet simple nous avons donc ensuite la ville de départ la ville d'arrivée et le moyen de transport, séparés par des virgules.
Si le nombre de sous-trajets est supérieur à un, c'est donc un trajet composé, on a donc la ville de départ puis la ville d'arrivée. Puis, sur le bon nombre de lignes suivantes, nous avons la description des sous-trajets composant le trajet total (identique à celle des trajets simples puisque c'est ainsi qu'ils sont modélisés dans notre application) .

Cette description est assez simple comme nous pouvons le voir sur l'exemple suivant.

2. Exemple

Cet exemple nous permet de montrer le contenu du fichier de sauvegarde / restitution en utilisant les trajets donnés dans le sujet de TP :

3
1,Lyon,Bordeaux,Train
1,Lyon,Paris,Auto
2,Lyon,Paris
1,Lyon,Marseille,Bateau
1,Marseille,Paris,Avion

3. Nouvelles fonctionnalités

Nous avons décidé d'intégrer l'architecture **MVC** (Modèle Vue Controlleur) à notre programme pour une meilleure clarté. De plus, nous avons fait en sorte que les classes Interface et Catalogue soit des **singletons**. Il ne peut pas y avoir plusieurs instances de ces classes.
L'utilisateur peut sauvegarder en format *.csv* son catalogue avec le nom choisi par ses soins. Par exemple : *demo*. Ce fichier sera alors sauvegardé dans le dossier *"/saves"* lui même se trouvant dans le dossier *"/src"* de notre solution.
Pour le chargement de fichier, l'utilisateur doit entrer le nom d'un fichier existant dans le dossier *saves* (sans son extension).

L'utilisateur peut appliquer quatre critères à la sauvegarde ou à la restitution de trajets : sans critère de sélection, selon le type des trajets, selon la ville de départ et/ou la ville d'arrivée et selon une sélection de trajets.

IV. Conclusion

Le principal problème que nous avons rencontré a été lors du remplacement des fonctions "*cin* >>" par des fonctions "*getline()*", pour pouvoir récupérer des espaces dans les noms de ville. En effet, dans notre menu nous avons des fonctions *fscanf* qui nous permettent de récupérer les commandes de l'utilisateur. Ces dernières conduisaient à un saut de certaines commandes *getline()* permettant par exemple la saisie d'une ville de départ à cause d'un retour à la ligne (caractère de fin de la fonction *getline()*) gardé en mémoire sur le buffer (zone mémoire où les caractères saisies par l'utilisateur sont stockés). Nous avons donc utilisé la fonction *cin.ignore()* pour "vider" le buffer avant de faire des saisies de villes ou de moyens de transport.