Rapport TP - Polymorphisme

Sommaire :

I) Description des classes

- La classe Trajet, est crée pour manipuler des trajets et a pour vocation à engendrer les deux classes : TrajetSimple et TrajetCompose. C'est une classe abstraire puisqu'elle contient des méthodes virtuelles pures. En effet l'on sera mené à afficher des trajets simples et des trajets composés, qui n'ont pas la même structure et donc les deux méthodes effectueront des opérations différentes en fonction du type de trajet dont il est question.

- Un trajet composé, et un catalogue de trajets étant tous les deux une collection ordonnée de trajets quelconques, nous avons opté pour la création d'une classe ListeChainee dans un souci de réemploi.

- On crée ainsi la classe CelluleTrajet, qui possède comme attributs un pointeur sur un trajet, et un pointeur sur CelluleTrajet ; pointeur qui pointera vers la prochaine cellule si elle existe et à NULL sinon.

- Une liste chainée étant entièrement caractérisée par un pointeur vers le premier élément, et un pointeur vers la dernière cellule, la classe ListeChainee possède deux pointeurs sur CelluleTrajet, appelés premier et dernier.

- Les classes TrajetCompose et Catalogue possèdent ainsi chacune comme argument un pointeur vers une ListeChainee, mais chacune possède des méthodes bien distinctes puisqu'elles n'ont pas la même utilité.

- La classe TrajetSimple quant à elle, possède comme argument 3 pointeurs sur une chaine de caractères, un pour la ville de départ, un pour la ville d'arrivée, et un pour le moyen de transport.

Ci dessous, le diagramme UML représentant les classes et leur dépendances :

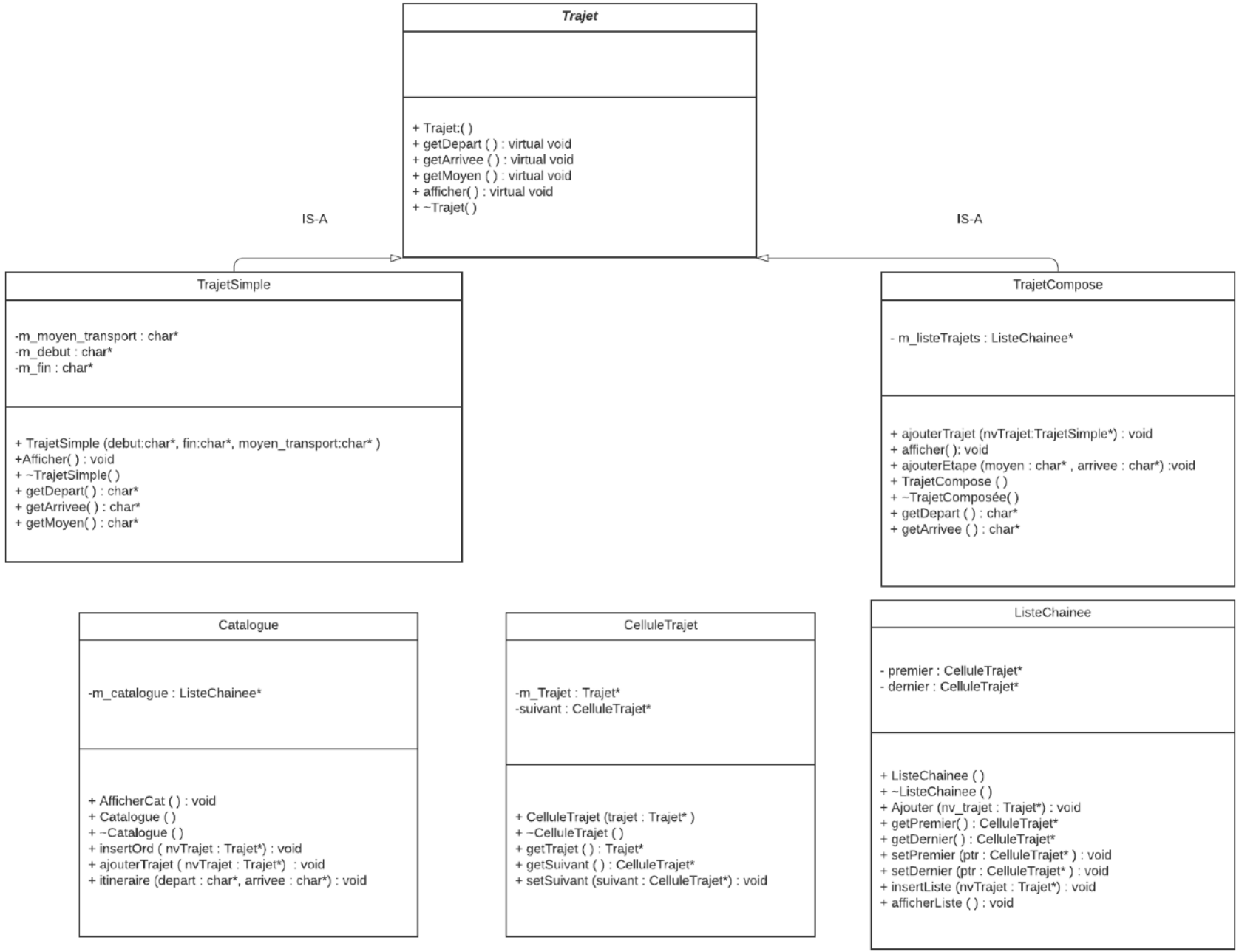
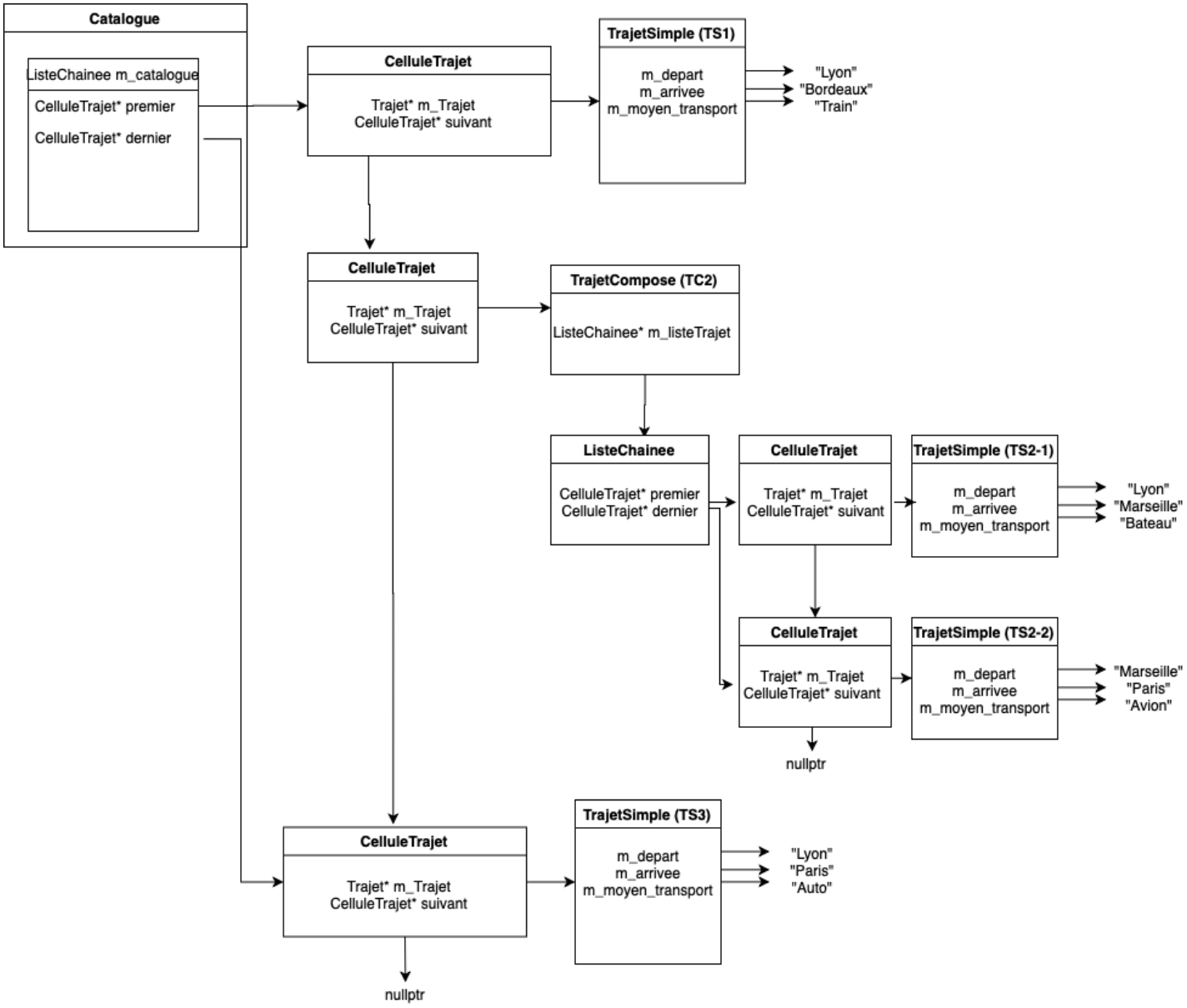


Diagramme UML du Programme « Trajet+ »

II) Description de la structure de données

Voici le fonctionnement de notre structure de données sur un exemple simple, avec un catalogue composé de 2 trajets Simples et 1 trajet composé :



III) Difficultés rencontrées

- Le premier souci auquel on a été confrontés s'est présenté au moment de la création de notre fichier principal main.cpp. En effet nous a fallu du temps pour comprendre comment faire en sorte que l'utilisateur puisse entrer ses mots en tant que pointeurs sur chaîne de caractères. On s'est rendu compte que le souci venait du constructeur TrajetSimple, puisqu'il ne faisait qu'affecter un pointeur à un autre sans allouer de mémoire au niveau du constructeur. Une fois ce souci réglé, tout se passait mieux.

- Ensuite la plus grande difficulté a été de détecter les messages et avertissements adressés par le programme Valgrind. Bien que le programme fonctionne correctement, et fasse tout ce qu'on demande de lui, l'outil Valgrind relève des erreurs qui n'ont pas été faciles à débusquer et qui concernaient la gestion de la mémoire dynamique. Il a fallu plus que le double du temps alloué à toute la création du Programme (qui fonctionnait déjà) pour annuler toutes les erreurs, et comprendre ce qu'elles signifiaient sur des forums. Une des raisons principales d'erreurs était que les attributs de TrajetCompose et Catalogue étaient des ListeChainee, et non pas des pointeurs sur ListeChainee, ce qui causait des problèmes pour leur suppression via leur destructeur.

- Pour le tri du Catalogue, il semblait préférable pour l'utilisateur d'avoir un tri par ordre alphabétique de la ville de départ des trajets. Cette opération, bien qu'en apparence simple n'a pas été de tout repos. Néanmoins , étant donnée que l'utilisateur entre un à un ses trajets dans le catalogue, cela facilite grandement le tri du catalogue puisqu'on a un contrôle sur les données entrés pas à pas.

IV) Axes d'amélioration

Bien que notre programme fonctionne correctement et puisse s'avérer assez utile à des utilisateurs en tout genre, quelques fonctionnalités élever encore l'intérêt de notre application.

- Premièrement, la recherche de Trajets pourrait gagner encore en efficacité. En effet, en partant du principe qu'un trajet composé pourrait être emprunté qu'en partie, on pourrait avoir de plus nombreux itinéraires disponibles.

- De plus, il serait beaucoup plus intéressant d'ajouter une mesure aux différents trajets, par exemple une mesure de durée. Ainsi, les itinéraires affichés seraient plus parlants, et l'utilisateur pourra les comparer avec plus de pertinence. Pour cela il faudrait ajouter un attribut entier nommé duree à la classe TrajetSimple. Puis, l'on pourrait ajouter une méthode itineraireOptimal à la classe Catalogue, qui permettait de trouver parmi les itinéraires possibles entre les différents trajets, celui le plus court : algorithme de Disjkrtat .

- Par ailleurs, on pourrait faire en sorte que les trajets ajoutés au catalogue soient unique, en implémentant une méthode qui renverrait un booléen en fonction de l'existence du trajet dans le Catalogue. Cette fonctionnalité serait assez simple à ajouter, mais nous ne l'avons pas fait dans un souci de rapidité du programme, et car cela reste secondaire.

- La recherche d'itinéraires avancée a été abordée, et a mené avec succès à une premiere version satisfaisante. Néanmoins elle s'arrêtait quand elle trouvait Un itinéraire « composé », et renvoyer une ListeChainee en sens inverse. Il aurait encore fallu implémenter une méthode récursive pour renverser la liste chainée.

Auteurs : Rim BENZEKRI, Adam CHELLAOUI

TP1 C++ (POO2) : Gestion des entrées / sorties

I. Fichier démo

I.1. Description détaillée du format

Chaque ligne commence par un **$** pour un **trajet simple** ou un **@** pour un **trajet composé**. Aussi, chaque ligne se termine par un **%**.

Pour un **trajet simple**, nous avons choisi le format suivant :

$(ville de départ),(ville d'arrivée),(nombre de trajets):(ville de départ),(moyen de transport),(ville d'arrivée),%

Nous pouvons noter que le **nombre de trajets** est **toujours égal à 1** pour un **trajet simple**.

Pour un **trajet composé**, nous avons choisi le format suivant :

@(ville de départ),(ville d'arrivée),(nombre de trajets) :(ville de départ),(moyen de transport),(ville de 1ère escale),(ville de 1ère escale),(moyen de transport),(ville de 2ème escale),(ville de 2ème escale),(moyen de transport),…..,(ville d'arrivée),%

N.B. Quand un catalogue est sauvegardé dans un fichier, la syntaxe est exactement la même que dans le fichier démo, ainsi, un catalogue sauvegardé peut également être inséré comme nouveau catalogue.

I.2. Fichier demo.txt

$Lyon,Bordeaux,1:Lyon,Train,Bordeaux,%

@Lyon,Paris,2:Lyon,Bateaux,Marseille,Marseille,Avion,Paris,%

$Lyon,Paris,1:Lyon,Auto,Paris,%

II. Conclusion

II.1. Problèmes marquants

Nous avons plusieurs fois changé le format du fichier, nous avons d'abord pensé à commencer chaque ligne par un S ou un C pour différencier les trajets simples et complexes, cependant, nous avons eu des bugs puisque ce serait plus difficile de différencier ces lettres de celles qui constituent une ville. Nous avons donc jugé que ce serait plus facile d'utiliser des symbole puisqu'une ville n'en contient pas.

Nous avons finalement opté pour un $ pour les trajets simples puisque ce symbole ressemble au S, et @ pour les trajets composés, nous aurions voulu utiliser un € pour les trajets composés mais ce n'est pas un caractère. Nous avons conscience que ça aurait été plus claire d'utiliser des chaines de caractères pour différencier les trajets, comme par exemple trajet_simple, trajet_composé, mais l'utilisation d'un seul caractère était beaucoup plus simple à coder.

Nous n'avons pas eu de gros problème, mais nous avons passé un peu plus de temps sur la méthode chargerVoyage qui permet d'insérer dans le catalogue des trajets en fonctions des villes de départ et d'arrivée. Finalement, nous avons simplement dû remplacer un sauter par un getline pour régler le problème.

II.2. Axes d'évolution

Nous avons réussi à produire un programme fonctionnel qui réponde aux demandes du cahier des charges. Cependant, certaines fonctions, bien que non obligatoires, le rendrait peut-être plus intéressant.

Nous n'avons par exemple pas intégré de méthode qui vérifie l'unicité d'un trajet, ainsi, si l'utilisateur rentre deux fois le même trajet, il apparaitra bien deux fois dans le catalogue. Nous avons jugé que cette méthode reste secondaire, et que le programme serait plus rapide sans.