Auteurs : Clément BOITTIN, Sami SHAAR

TP C++ n°2 : Héritage – Polymorphisme

I. Introduction

Le logiciel réalisé au cours de ce TP consiste en une implémentation d'un catalogue de trajets permettant d'effectuer des recherches de voyage. Nous avons donc manipulé des *Trajets* recensés dans un *Catalogue* afin que l'utilisateur puisse planifier ses prochaines vacances en fonction des *Parcours* proposés.

II. Description générale

Pour réaliser ce logiciel, il faut dans un premier temps, en définir ses limites.

Tout d'abord, le problème principal est la distinction de deux types de *Trajet* : les *TrajetSimple* et les *TrajetComposé*. Un *TrajetSimple* est composé d'une ville de départ, d'une ville d'arrivée ainsi que d'un moyen de transport tandis qu'un *TrajetComposé* n'est rien d'autre qu'une collection ordonnée de *TrajetSimple*. Le *Catalogue* recense donc des *Trajet*, sans faire de distinction entre un *TrajetSimple* et un *TrajetComposé*. Ensuite, lorsque l'utilisateur s'adonne à la recherche d'un voyage, une liste de parcours s'offre à lui.

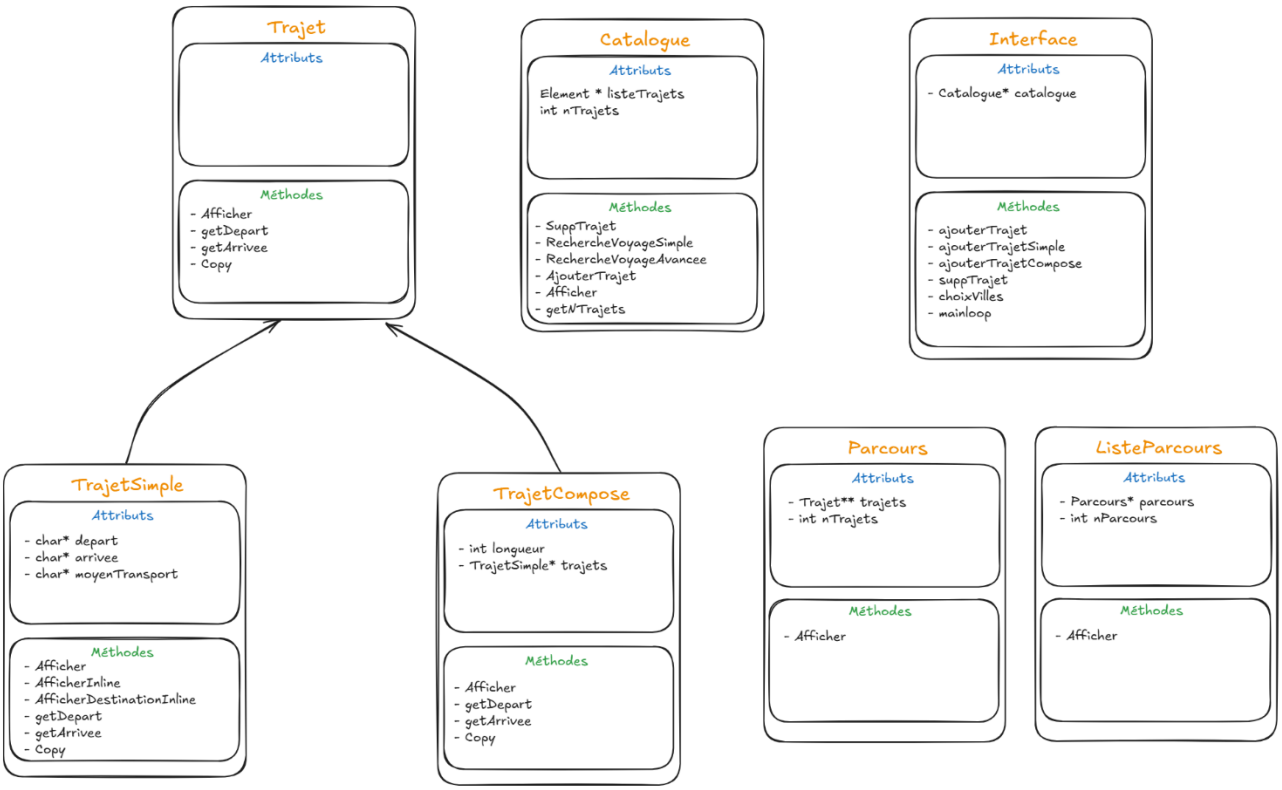

Figure 1 : Schéma symbolique des classes implémentées

Dans ce schéma, nous constatons bien les relations d'héritage, représentées par des flèches, entre *Trajet* et *TrajetSimple* ou *TrajetComposé*, ce qui nous permettra de les traiter sans distinction. La classe *Trajet* est une classe abstraite.

De plus, la classe *Catalogue* permet la recherche de voyage (simple ou avancée) qui constitue le cœur du logiciel. Cette classe dispose aussi d'une structure Element qui est un composant de la liste chaînée, utilisée pour la recherche de voyage.

Ensuite, l'Interface gère les interactions avec l'utilisateur via des lignes de commande.

En outre, nous avons choisi de définir un objet de type *Parcours*, utile notamment pour la recherche avancée, qui combine les *Trajet* du *Catalogue* afin d'aller d'une ville A à une ville B, choisies par l'utilisateur.

Pour finir, l'objet *ListeParcours* est, comme son nom l'indique, une liste de *Parcours* allant d'une ville A à une ville B. Cette classe nous permet un code plus propre, notamment pour son affichage. En effet, les algorithmes de recherche étant déjà assez compliqué et volumineux, nous avons préféré séparer tout ce qui concerne les affichages dans des méthodes de classe ou dans l'Interface.

III. Description de la mémoire

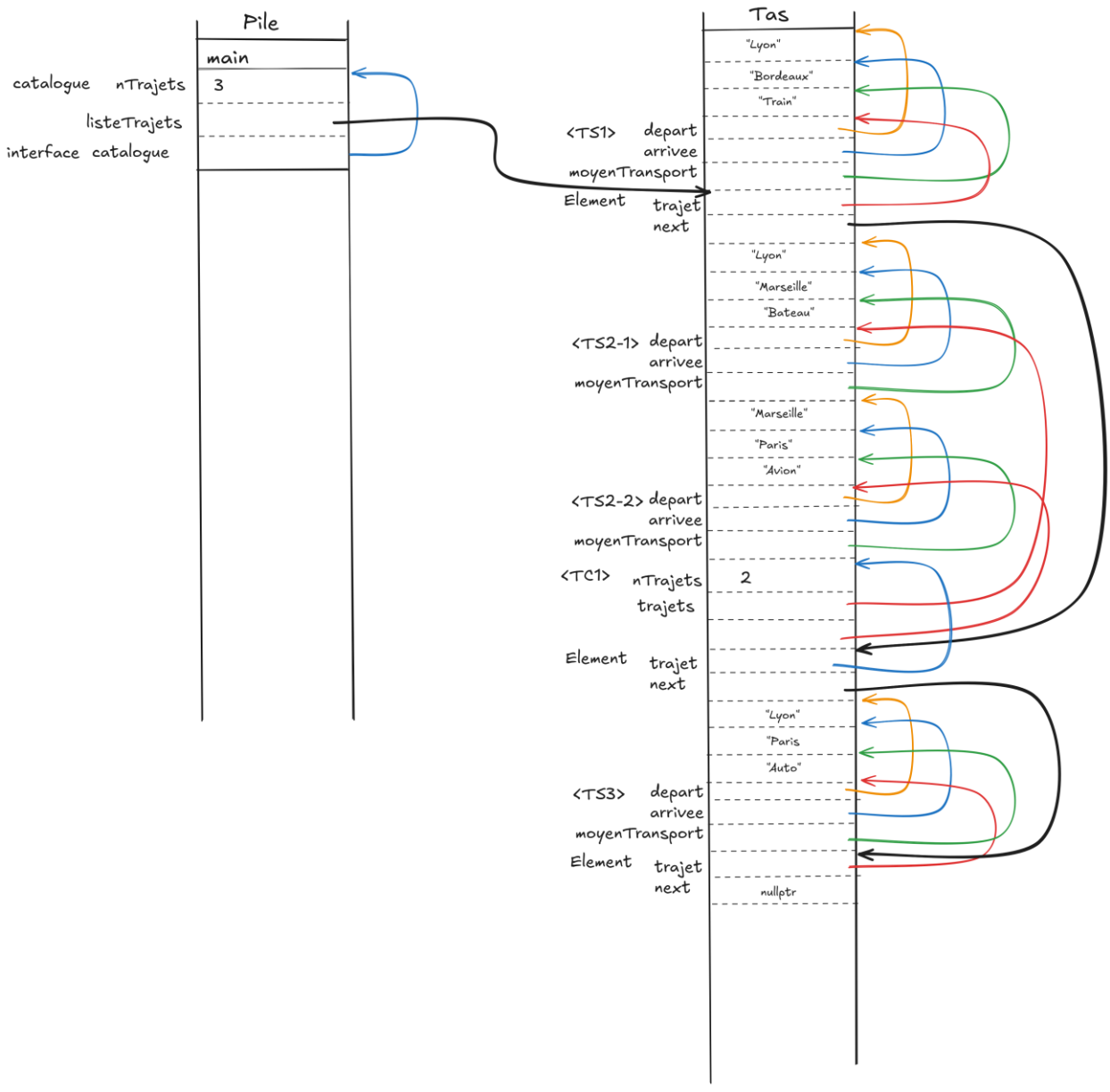

Figure 2 : Schéma de la mémoire avec les données de l'énoncé

IV. Problèmes/Améliorations

IV.A. Gestion des entrées/sorties

Tout d'abord, un problème rencontré a été la gestion des entrées notamment. En effet, ne pouvant seulement utiliser *cin, cout* et *cerr* de la bibliothèque *iostream*, les possibilités quant à la gestion du flux de données entrant sont limitées. Par exemple, si un utilisateur farfelu veut se rendre à « Le Havre », *cin* ne prendra en compte que le « Le » comme ville d'arrivée et « Havre » constituera la prochaine entrée. C'est pourquoi, nous demandons toujours à l'utilisateur la confirmation du trajet écrit avant de l'ajouter ou non au catalogue.

IV.B. Modification du catalogue

Ensuite, le cahier des charges ne demandait que l'ajout de trajet au catalogue. Cela signifie qu'une fois le trajet ajouté, il devient impossible de le modifier ou de le supprimer. L'utilisateur doit donc être sûr de ce qu'il insère dans le catalogue, d'où l'étape de confirmation évoquée plus tôt. Cependant, nous avons pris la décision de proposer à l'utilisateur la suppression d'un trajet afin qu'il puisse plus facilement ajuster son catalogue de trajets ainsi que sa planification de vacances.

IV.C. Gestion des chaînes de caractères

En programmation C et C++, le principal défi est la gestion de la mémoire nécessaire aux chaînes de caractères. En effet, l'objectif du TP était de nous faire manipuler tout un tas de pointeurs dans le but de maîtriser cette gestion de la mémoire. Or, cette décision d'implémentation n'est clairement pas la plus optimale. Il aurait été plus judicieux d'implémenter des *strings* de caractères qui gèrent automatiquement la mémoire.

IV.D. Interface

Le « logiciel » réalisé est conçu pour fonctionner dans un terminal et interagit avec l'utilisateur via des lignes de commandes. L'interface est minimaliste et aurait pu nécessiter une implémentation d'une IHM (Interface Homme Machine) plus conviviale. Notre temps et nos compétences étant limités, nous avons dû faire sans mais il suffirait de modifier la classe Interface pour réaliser une interface digne de ce nom.