Bondyfalat Julien 3IF
Canaple Gabriel

TP C++ n°2 : Héritage - Polymorphisme

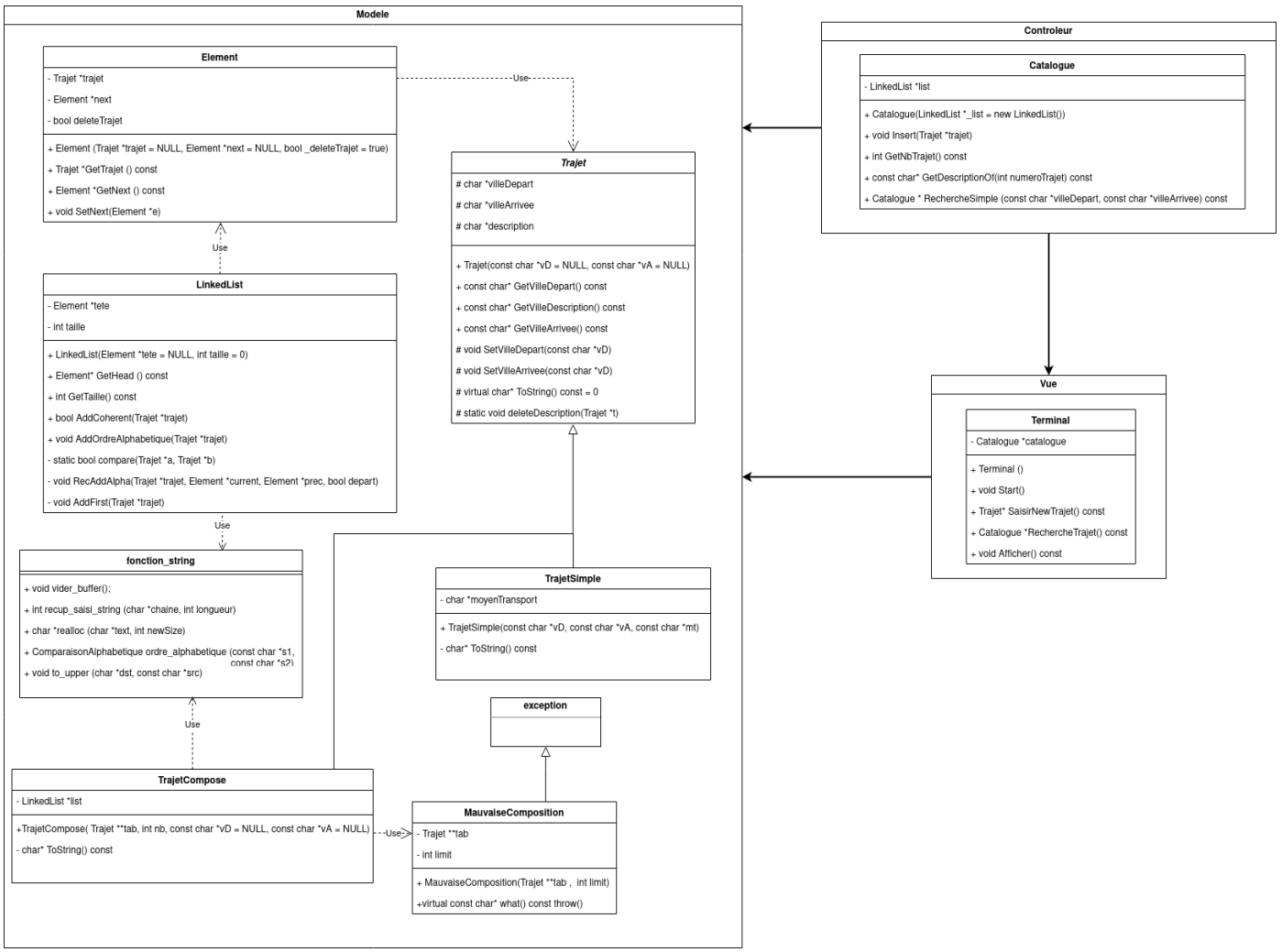

Image 1 : Diagramme UML du projet

I) Description des classes

- **Trajet :**

Trajet est une classe abstraite générique dont la structure est utilisée par les classes TrajetSimple et TrajetCompose. Elle a pour attributs la ville de départ, la ville d'arrivée, et la description du trajet, qui sert à regrouper les caractéristiques d'un trajet pour l'affichage.
L'utilité de la classe Trajet repose dans le fait qu'elle permet de créer une liste chaînée de TrajetSimple et de TrajetCompose en les considérant comme des Trajet grâce au concept de polymorphisme. Elle permet également de limiter la redondance dans TrajetSimple et TrajetCompose.

- **TrajetSimple :**

TrajetSimple est la première implémentation concrète de Trajet. Elle hérite de Trajet, et a un attribut supplémentaire qui est le mode de transport. Du fait de l'héritage, c'est une classe très simple.

- **TrajetCompose :**

TrajetCompose est la deuxième implémentation concrète de Trajet. Elle hérite de Trajet, et a un attribut supplémentaire, qui est une LinkedList contenant des sous-trajets de type Trajet. Avec le polymorphisme, on a ainsi la possibilité d'enregistrer des sous trajets pouvant être des TrajetSimple, mais également des TrajetCompose.

- **MauvaiseComposition :**

Cette classe hérite d'exception, et permet, pour un TrajetCompose, de gérer les exceptions liées à l'ajout d'un trajet ne respectant pas les règles de cohérence entre ville de départ et ville d'arrivée. Elle se charge notamment de la destruction du tableau utilisé dans le constructeur de TrajetCompose.

- **Element :**

Un Element est un chaînon de la LinkedList utilisée dans le projet. Il contient trois attributs, qui sont un trajet, un booléen spécifiant le comportement lors du delete, et un pointeur vers l'élément suivant afin d'assurer le chaînage. La présence du booléen est rendue nécessaire par la méthode RechercheSimple de Catalogue. Cette classe introduit des méthodes donnant accès aux caractéristiques des Element, afin de permettre leur manipulation par d'autres classes, comme LinkedList.

- **LinkedList :**

Cette classe implémente une liste chaînée simple, avec pour chaînons des Element. Elle contient deux attributs, qui sont sa taille, et un pointeur vers le premier Element de la liste. Elle intègre notamment des méthodes facilitant l'ajout de trajets à la liste, avec un ajout par ordre alphabétique, pour le catalogue, et un ajout dit *cohérent*, qui se révèle utile pour les TrajetCompose.

- **Catalogue :**

Un Catalogue a pour attribut une liste chaînée de type LinkedList contenant des trajets. Sa particularité est de toujours devoir être ordonné (tous les ajouts au catalogue se font par ordre alphabétique), et de fournir une méthode permettant la recherche de trajets allant d'un point A à un point B en spécifiant A et B.

- **Terminal :**

Il s'agit de la classe assurant l'interface homme-machine de notre projet. En ce sens, elle intègre une méthode qui permet à l'utilisateur d'interagir avec un menu, qui comprend les fonctionnalités mentionnées dans le cahier des charges, à savoir l'affichage du catalogue courant, l'ajout d'un trajet au catalogue courant, et la recherche de trajet. D'autres méthodes sont incluses, permettant de gérer chacun des choix susmentionnés.

- **fonction_string :**

Ce module propose des fonctions de manipulation de chaînes de caractères, qui sont utilisées notamment dans LinkedList et TrajetCompose. Il facilite la saisie de chaînes de caractères, ainsi que la comparaison par ordre alphabétique de deux chaînes.

II) Description de la structure de données choisie : une liste chaînée

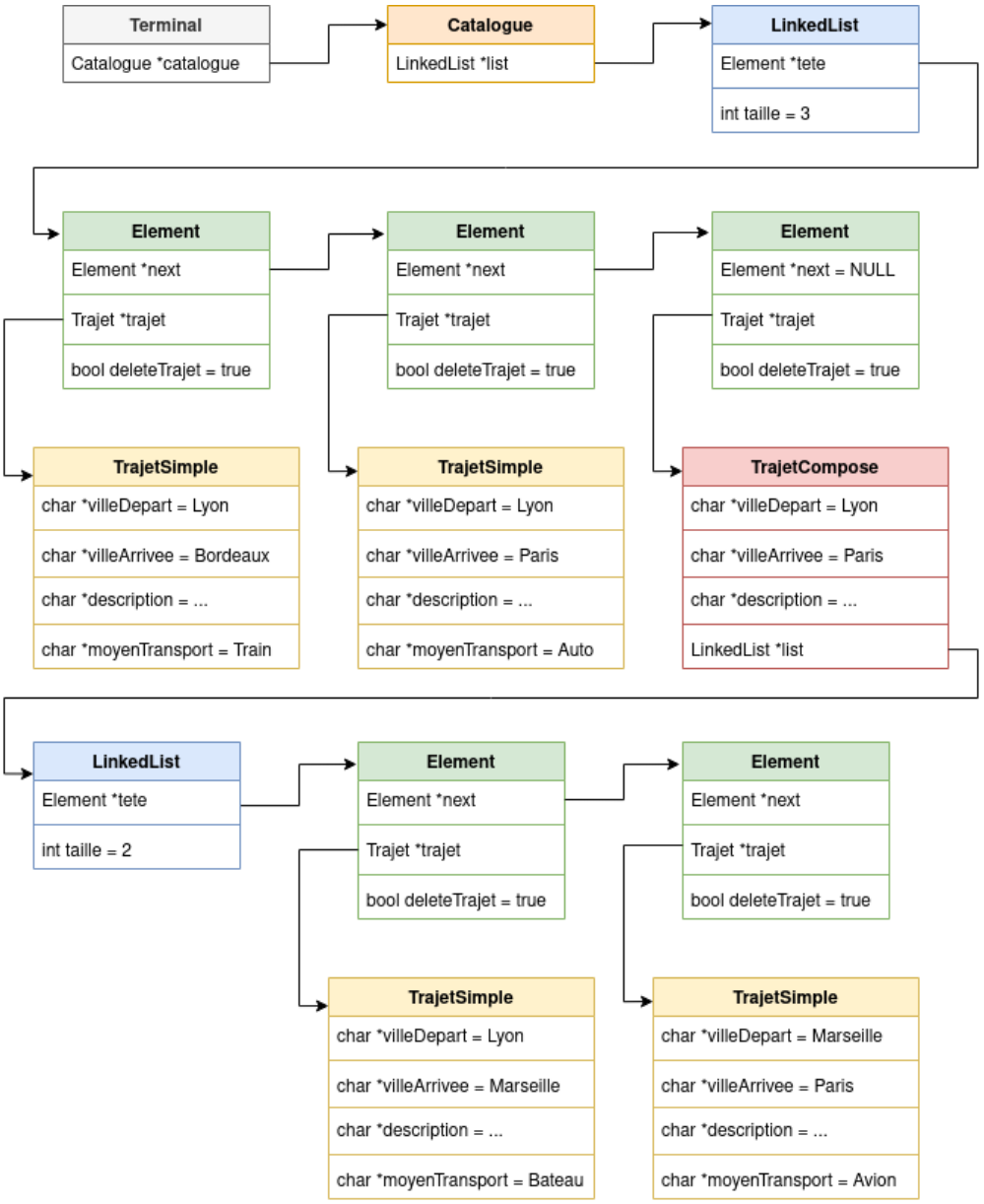

Image 2 : Schéma de la mémoire après la création des trajets décrits dans le TP

Pour des raisons d'espace et de clarté, nous n'avons pas utilisé la représentation en colonnes de la mémoire. La répartition des données dans la mémoire est la suivante :

Dans la pile, dans le contexte du main, se trouve l'objet Terminal. A sa construction, il construit un Catalogue dans le tas, qui, lui-même, construit une LinkedList dans le tas.

Ensuite, à chaque création de trajet simple, le schéma est le même :

- Les chaînes de caractère sont allouées dans le tas, puis demandées à l'utilisateur et récupérées via la fonction fgets, tout cela dans la classe Terminal.

- Le TrajetSimple est créé dans le tas à partir des chaînes, puis l'Element est aussi créé dans le tas, et ajouté à la liste par ordre alphabétique. A chaque nouvel ajout à la liste, l'attribut next sera actualisé.

Puis, à chaque création de trajet composé, on crée autant de trajets simples que demandé par l'utilisateur, en réitérant les étapes décrites ci-dessus. Tout sera encore une fois créé dans le tas.

III) Problèmes rencontrés durant le TP

Un problème rencontré au moment du développement de la méthode RechercheSimple, dans Catalogue, a été de gérer la délétion du Catalogue retourné par la fonction. En effet, le Catalogue retourné ne réallouant pas les trajets trouvés via la recherche, nous avons été face à une double délétion. Deux solutions se sont alors présentées :

- La première, qui nous plaisait davantage, était de copier chacun des trajets (et donc de les réallouer), afin de pouvoir delete le Catalogue retourné sans soucis.
- La deuxième, qui nous a semblé moins adaptée, était d'intégrer dans la classe Element un booléen spécifiant le comportement lors de la délétion de cet Element. Ainsi, les trajets associés aux Element ne seront pas détruits si ce booléen est à false.

C'est au final la deuxième solution que nous avons adoptée, n'ayant pas trouvé de solution qui n'utilisait que des notions abordées dans la première partie du cours de C++ pour la première solution. En effet, la première solution impliquait qu'il fallait retrouver le type dynamique (TrajetSimple ou TrajetCompose) de chacun des Trajet inclus dans les chaînons de la liste chaînée.

Nous avons également rencontré une difficulté lors de l'instanciation d'un TrajetCompose. En effet, lorsque le tableau de Trajet passé en paramètre n'était pas au *cohérent* c'est-à-dire que, pour un sous-trajet, il n'y avait pas une ville de départ qui correspondait avec la ville d'arrivée du sous-trajet précédent, on sortait du constructeur de façon assez brusque ce qui engendrait de nombreuses fuites mémoire et notamment au niveau du tableau en paramètre et du début de LinkedList qui a été créée.

Une solution pour résoudre ce problème a donc été de lancer une exception que nous avons définie et dont le but est de récupérer ce tableau ainsi que l'indice à partir duquel un problème a été détecté afin de détruire tous les trajets restants avant de détruire le tableau en lui-même ainsi que la liste qui a commencé à être remplie. Pour faire cela, on aurait pu se passer de créer une exception mais il aurait fallu trouver un moyen de mettre fin au constructeur subitement (chose pas forcément facile à faire car il n'y a pas de "return" dans un constructeur) et détruire l'objet TrajetCompose créé incomplètement, ce qui aurait posé problème dans le destructeur.

IV) Conclusion

Finalement, ce TP a été l'occasion de mettre en pratique toute la première partie du cours de C++, ce qui s'est révélé à la fois complexe et riche en apprentissages. Si nous aurions parfois aimé utiliser des notions du cours avancé de C++, comme le mot-clé friend ou la surcharge d'opérateurs, nous avons quand même pu aller suffisamment loin pour être satisfaits. Il reste cependant des angles d'amélioration, notre projet étant très perfectible. Entre autres, nous aurions pu intégrer la méthode de recherche avancée décrite dans le TP, ou alors mettre en place une IHM plus sophistiquée intégrant une partie graphique.