TP C++ n°2 : Héritage – Polymorphisme

Intégrants du binôme n°B3403

- AQUINO Diego
- LINARES Sebastian

I. Description des Classes

Tout d'abord, le schéma des classes de notre programme, notamment le graphe d'héritage, est le suivant:

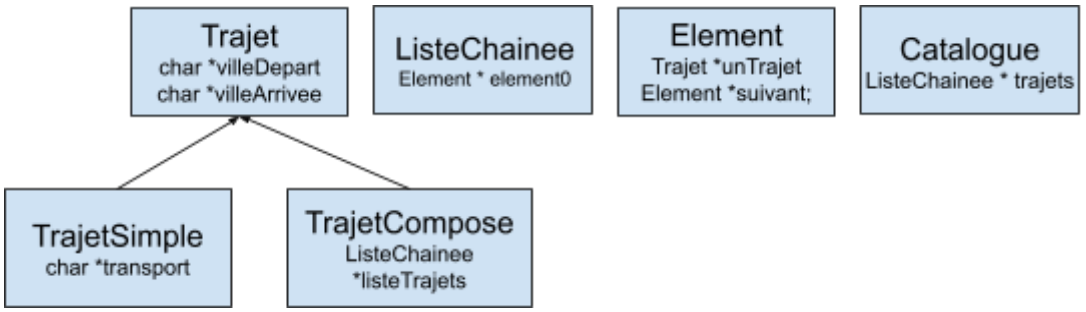

Figure 1: Schéma des classes

Dans un premier temps, le programme distingue deux types de trajets : les trajets simples et les trajets composés (constitués de plusieurs trajets simples). Ces deux types de trajets partagent deux attributs communs : ville de départ et ville d'arrivée. Cependant, chacun possède également des caractéristiques spécifiques. Pour répondre à ces besoins, nous avons conçu une classe de base appelée Trajet, qui contient les attributs villeDepart et villeArrivee. Cette classe sert de fondation pour deux classes dérivées :

1. TrajetSimple : Cette classe inclut un attribut spécifique, transport, qui représente le moyen de transport utilisé pour chaque trajet simple.
2. TrajetCompose : Cette classe possède un attribut unique, listeTrajets, qui contient la liste des trajets simples formant le trajet composé.

Ensuite, nous avons créé deux classes supplémentaires, Element et ListeChainee, qui permettent d'implémenter une structure de liste chaînée pour gérer les collections de trajets de manière flexible et efficace.

Enfin, une classe Catalogue a été développée pour ajouter et rechercher des trajets dans le programme, offrant une interface centralisée et intuitive pour manipuler les données. Une explication détaillée de chaque classe est fournie ci-après.

A. Classe Trajet

La classe Trajet permet de représenter un trajet défini par une ville de départ et une ville d'arrivée. Cette classe est destinée à servir de base pour les classes TrajetSimple et TrajetCompose.

Cette classe contient un constructeur par défaut, un constructeur qui reçoit comme paramètres la ville de départ et la ville d'arrivée, et un destructeur qui libère la mémoire éventuellement allouée pour ces deux attributs. En plus, elle contient deux getters, getVilleDepart et getVilleArrivee, qui permettent d'accéder à la ville de départ et à la ville d'arrivée d'un trajet, et sont utilisés notamment pour la construction d'un trajet composé. Finalement, elle possède une méthode virtuelle Afficher, qui affiche la ville de départ et la ville d'arrivée d'un trajet. Cette méthode a été déclarée comme virtuelle parce que chaque type de trajet pourra définir sa propre méthode Afficher.

B. Classe TrajetSimple

La classe TrajetSimple hérite de la classe Trajet et représente un trajet simple. Ce type de trajet est caractérisé par un unique moyen de transport.

Elle comporte un constructeur qui reçoit comme paramètres la ville de départ, la ville d'arrivée et le moyen de transport, ainsi qu'un destructeur qui libère la mémoire éventuellement allouée pour l'attribut transport. En outre, elle comporte une méthode Afficher, qui fait appel à la méthode Afficher de la classe Trajet et affiche également le moyen de transport du trajet simple.

C. Classe TrajetCompose

La classe TrajetCompose est conçue pour représenter une succession ordonnée de plusieurs trajets simples formant un trajet complexe. Elle hérite de la classe Trajet.

Elle comporte un constructeur qui reçoit comme paramètre une liste chaînée de trajets simples constituant le trajet composé, ainsi qu'un destructeur qui libère la mémoire allouée pour cette liste. Par ailleurs, elle comporte une méthode Afficher, qui fait appel à la méthode Afficher de chaque trajet simple contenu dans la liste de trajets.

D. Classe Element

La classe Element représente un nœud dans la liste chaînée. Chaque instance contient un trajet (simple ou composé) et un pointeur vers l'élément suivant dans la liste.

Elle comporte un constructeur par défaut qui initialise les attributs à NULL, un constructeur qui reçoit comme paramètre un Trajet et un destructeur.

E. Classe ListeChainee

La classe ListeChainee est une structure de données basée sur des éléments chaînés. Une instance est construite à partir d'un élément initial (Element * element0), qui sert de point

d'entrée dans la liste chaînée. Cette structure est utilisée pour assembler des trajets composés, où chaque Element contient un trajet simple. Le catalogue est également basé sur une liste chaînée, où chaque élément correspond à un trajet (simple ou composé). L'élément initial (element0) pointe vers une série d'objets Element, chacun encapsulant un trajet.

Cette classe comporte également un constructeur par défaut qui crée une liste vide, ainsi qu'un destructeur qui itère dans la liste chaînée pour libérer la mémoire. En plus, elle comporte une méthode AjouterElements qui ajoute un trajet à la fin de la liste.

F. Classe Catalogue

La classe Catalogue permet de gérer un ensemble de trajets saisis par l'utilisateur. Un objet Catalogue repose sur une instance de ListeChainee, qui stocke les trajets sous forme de liste chaînée.
Cette classe comporte un constructeur qui reçoit en paramètre une liste chaînée, ainsi qu'un destructeur qui libère la mémoire allouée pour cette liste. De plus, elle inclut une méthode AjouterTrajet, qui permet d'ajouter un trajet simple ou composé au catalogue. Elle dispose également d'une méthode Afficher, qui fait appel à la méthode Afficher de chaque trajet simple ou composé contenu dans la liste de trajets du catalogue. Enfin, elle propose une méthode RechercherParcoursSimple, qui permet de trouver des trajets dans le catalogue en fonction d'une ville de départ et d'une ville d'arrivée.

II. Structure de données utilisée pour gérer la collection ordonnée de trajets

La structure de données adoptée est une liste chaînée, qui permet de gérer efficacement une collection ordonnée de trajets. La liste chaînée commence par un attribut element0, de type Element*, qui représente le premier nœud de la liste.

Chaque objet de type Element contient un pointeur vers un trajet (qui peut être simple ou composé) et un pointeur Element* suivant, qui relie cet élément à l'élément suivant dans la liste. Ainsi, chaque élément contient à la fois les informations relatives au trajet et un lien vers l'élément suivant, permettant une organisation dynamique et flexible des trajets.

La liste se termine lorsqu'un élément a un pointeur suivant égal à null, ce qui marque la fin de la séquence de trajets. Cette structure facilite l'ajout et la suppression d'éléments en tout point de la liste, tout en maintenant l'ordre des trajets.

III. Dessin de la mémoire pour le jeu d'essai

Nous allons maintenant présenter le dessin de la mémoire pour le jeu d'essai donné. Ce dessin représente l'état final de la mémoire, c'est-à-dire lorsque l'utilisateur a déjà saisi tous les trajets. L'objectif de ce schéma est d'illustrer la structure de données adoptée (une liste chaînée) et sa représentation en mémoire pour le trajet composé et le catalogue.

Par ailleurs, les chaînes de caractères sont représentées sur une seule ligne dans ce dessin. Bien que cette représentation ne soit pas conforme (les chaînes devraient normalement être

affichées caractère par caractère), nous avons choisi cette approche afin d'économiser de l'espace et de rendre le dessin plus lisible.

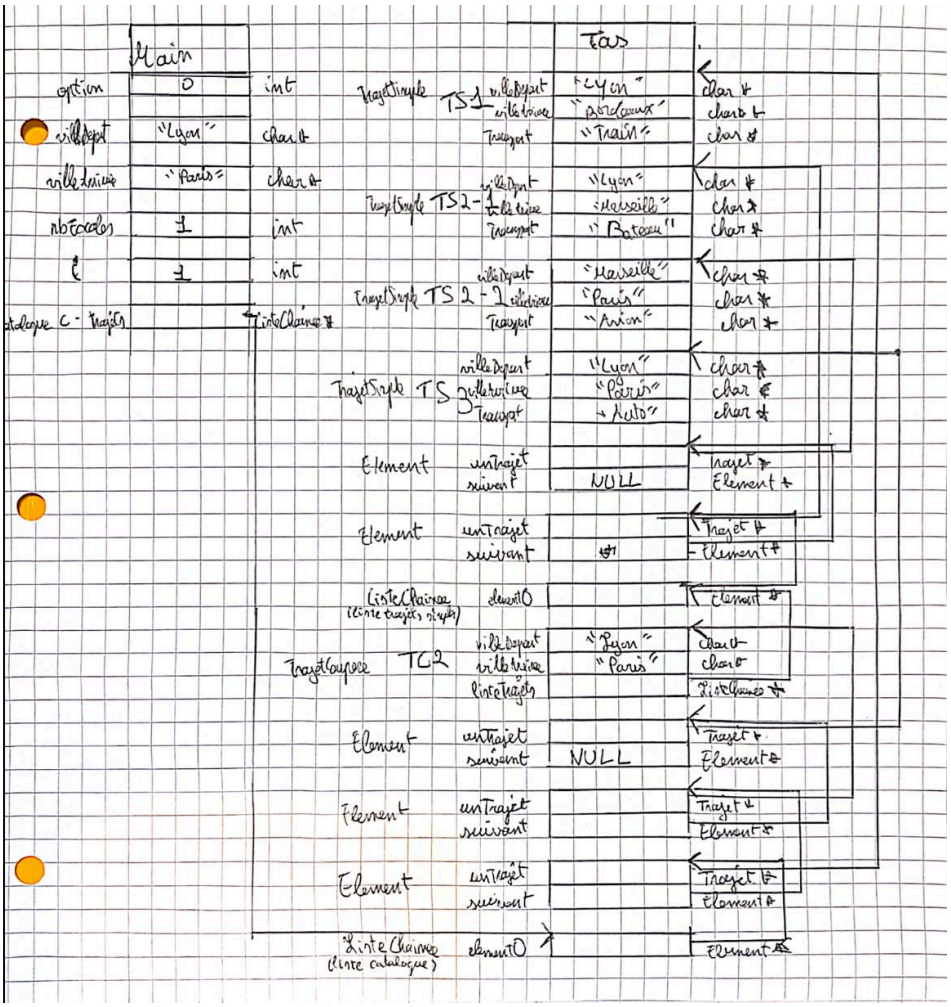

Figure 2: Dessin de mémoire pour le jeu d'essai

IV. Problèmes Rencontrés

Tout d'abord, nous avons rencontré un problème lorsque nous avons tenté d'accéder à la valeur d'un attribut protected en dehors des méthodes de la classe à laquelle cet attribut appartient (par exemple, pour accéder à la ville de départ d'un trajet afin d'initialiser la ville de départ d'un trajet composé). Pour résoudre ce problème, nous avons créé des méthodes appelées getters, qui permettent de récupérer la valeur des attributs d'un objet. On a aussi utilisé un setter pour modifier la valeur des attributs.

Ensuite, nous avons rencontré des difficultés pour libérer correctement la mémoire des objets alloués dynamiquement dans notre programme. Après plusieurs tentatives, nous avons réussi à libérer toute la mémoire allouée. Cependant, notre programme tente parfois de libérer de la mémoire déjà libérée, ce qui constitue un problème. Ce point reste à améliorer dans notre implémentation.