Compte rendu de TP - POO 1

I. Description des différentes classes de l'application

Comme on peut le voir sur le schéma ci-dessous, l'application comporte au total 5 classes : Trajet, TrajetSimple, TrajetCompose, Noeud, et Catalogue.

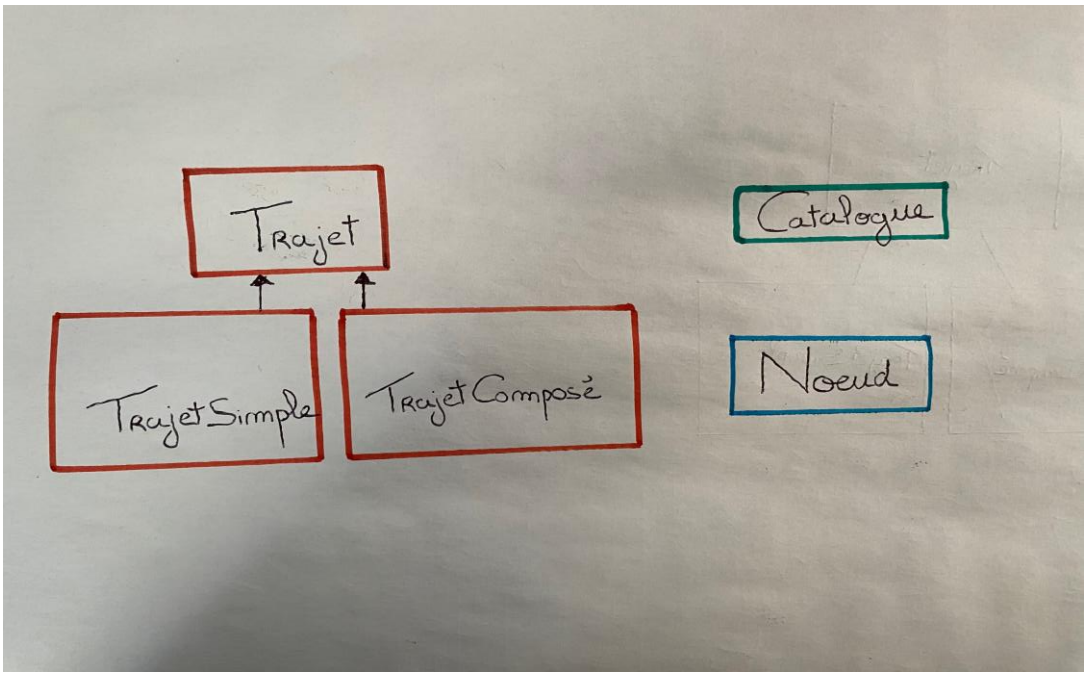

Figure 1 : Graphe d'héritage des classes de l'application

La classe Trajet comporte deux attributs qui sont des chaînes de caractères : un départ et une arrivée. En réalité, un trajet peut être simple ou composé. Ainsi, les deux classes TrajetSimple et TrajetCompose héritent de la classe trajet. De plus, puisqu'un trajet est nécessairement simple ou composé, il ne peut exister d'objet "Trajet" seulement ; la classe Trajet est donc une classe abstraite. Mais qu'est-ce qui différencie alors le trajet simple du trajet composé dans notre implémentation ?

En plus des attributs dont elle hérite, la classe TrajetSimple possède l'attribut qui correspond au mode de transport utilisé lors du voyage. Le type de cet attribut est un MDT, une énumération entre différents types de locomotion disponibles : l'avion, la voiture, le bateau, le train et "inconnu". A l'inverse, la classe TrajetCompose comporte trois nouveaux attributs au-delà de ceux dont elle hérite : une liste ordonnée des différentes escales du voyage, le nombre d'escales total, et une liste ordonnée de mode de transports utilisés lors des différents trajets qui composent le voyage.

En plus de ces trois classes, notre implémentation utilise également la classe Noeud et la classe Catalogue. Cependant nous détaillerons plus amplement l'utilisation de ces deux classes plus bas puisque ce sont elles qui contiennent la structure de données utilisée pour gérer la collection ordonnée de trajets.

II. Description de la structure de données utilisée

Ainsi donc, pour gérer notre collection ordonnée d'objets, nous utilisons la classe Noeud, qui comporte deux attributs : un trajet et le nœud d'après. Le trajet est un pointeur vers un objet de type Trajet est peut donc être aussi bien un trajet simple qu'un trajet composé. Les deux cas sont traités indifféremment. Le nœud d'après est un pointeur sur un nœud, ce qui permet de traiter notre structure de données comme une liste chaînée. Le dernier trajet de la liste aura donc un pointeur nul.

En parallèle, pour gérer cette structure, on fait appel à la classe catalogue, qui contient un attribut Liste (un pointeur vers le premier nœud de la liste chaînée) et un attribut dernier (un pointeur vers le dernier nœud de la liste chaînée). On remarque que si la liste ne contient qu'un seul élément, les deux attributs pointent vers le même nœud. Pour mieux visualiser ce que cette structure implique, voici un schéma de la mémoire sur un jeu de test simple. Ici, la liste chaînée est composée de trois trajets dont les informations sont détaillées dans le TP.

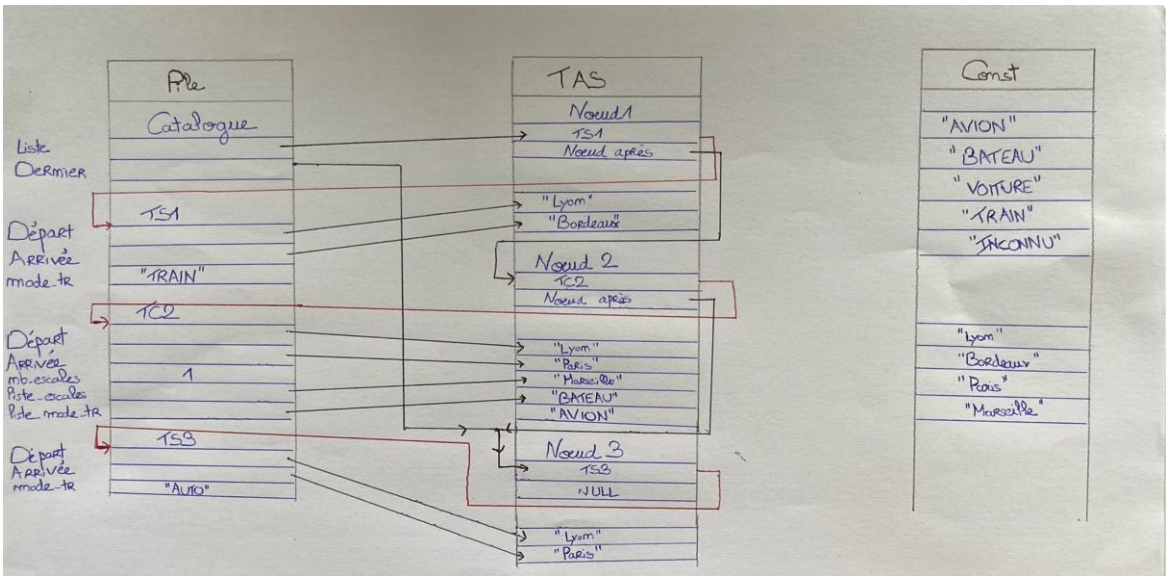

Figure 2 : Schéma de la mémoire

Sur le schéma ci-dessus, on voit que l'objet Catalogue, ainsi que ses deux attributs Liste et Dernier (des pointeurs) sont stockés en mémoire dans la pile. Chaque nœud de la liste est initialisé par un new. Ainsi, les trois nœuds correspondant à TS1, TS3 et TC2 sont contenus dans le tas. Chacun de ces nœuds contient un pointeur vers un trajet, et un pointeur vers le prochain nœud.

Les trajets n'étant pas initialisés par un new, ils sont tous stockés dans la pile. Ils contiennent un pointeur vers la chaîne de caractère départ, et un vers la chaîne de caractère arrivée (toutes deux dans le tas à cause de new). Ces chaînes, pour être initialisées, sont copiées de la zone de mémoire const grâce à strcpy. Remarque : Dans l'ensemble du schéma, on a représenté les chaînes de caractères dans la mémoire sous la forme "chaîne de caractères" par simple souci de simplification. En réalité, chaque case mémoire correspond à une lettre : 'c','h','a','î' etc.

Enfin, il y a une différence entre les trajets simples et le trajet composé. Le trajet simple possède un mode de transport, copié sur la pile à partir des différents modes de transports énumérés et présents dans const. Pour le trajet composé, le nombre d'escales est stocké sur la pile. La liste des modes de transports ainsi que la liste des escales sont deux pointeurs stockés sur la pile qui pointent chacun vers une liste stockée sur le tas (à cause du new).

III. Difficultés rencontrées

Problèmes rencontrés :

- Le mélange des cin >> entier et les cin.getline(…) créait des bugs où le programme sautait des saisies à cause du '\n' non pris par le cin >> entier. Solution : Utilisation de cin.ignore(…).
- La gestion de la liste des trajets déjà pris en compte dans la recherche complexe du catalogue (variable trajets_deja_faits) a été laborieuse car il ne fallait ni l'écraser à chaque appel récursif ni la garder en mémoire pour toujours avec le mot clé *static.* Solution : créer une fonction de façade initialisant cette variable qui ne se détruit qu'à la fin de la recherche complète. Néanmoins la variable peut contenir 100 trajets maximum, il faudrait peut-être ajouter une variable d'instance nb_trajets pour la classe Catalogue et allouer alors dynamiquement la mémoire de trajets_deja_faits avec cette variable d'instance.
- Le constructeur de la classe TrajetCompose nous a posé un petit peu problème pour la bonne gestion du tableau de Trajet* et l'allocation dynamique de la mémoire pour toutes les escales. On aurait également pu réaliser une liste chaînée pour ces escales mais étant donné que l'on ne peut pas modifier un trajet cette solution est peu utile.
- Nous avons eu du mal en général avec l'héritage des classes : méthode à redéfinir, celles à compléter…

Pistes d'améliorations :

- On pourrait garder en mémoire le Catalogue même après la fin du programme en écrivant dans un fichier texte/binaire que l'on pourrait charger au début du programme.

- On pourrait ajouter la possibilité de supprimer un trajet, même si demande un peu de réflexion à cause de la liste chaînée des nœuds et la possibilité d'avoir plusieurs trajets avec les mêmes départs et arrivées.
- Nous pourrions remplacer la méthode Afficher() pour les trajets par la surcharge de l'opérateur <<.
- Nous pourrions empêcher la construction par copie ainsi que l'utilisation de l'opérateur '=' pour les trajets qui n'ont pas d'opérateur de copie ni de surcharge de l'opérateur '=' car inutiles dans le cadre du TP et compliqués à gérer à cause du grand nombre de pointeurs pointant sur le tas pour les trajets composés notamment.