Explication détaillée du code source

A. Description des classes et graphe d'héritage

- **La classe Trajet**

Un trajet est une classe seulement caractérisée par un départ et une arrivée.

- **La classe TajetS (pour les trajets simples)**

Les trajets simples sont des trajets mais avec un seul moyen de transport. Donc la classe TrajetS est un descendant de la classe Trajet avec pour caractéristique propre le moyen de transport.

- **La classe TrajetC (pour les trajets composés)**

Un trajetC est un trajet composé de plusieurs trajets simples et/ou composés. Donc trajetC est est un descendant de trajet constitué d'une liste chaînée "liste".

- **La classe Maillon**

La classe maillon permet d'implémenter une liste chaînée de trajets :
-Trajet* elem: pointeur vers un trajet simple/composé;
-Maillon* suivant: pointeur vers le prochain maillon de la liste chaînée

- **La classe LChaine**

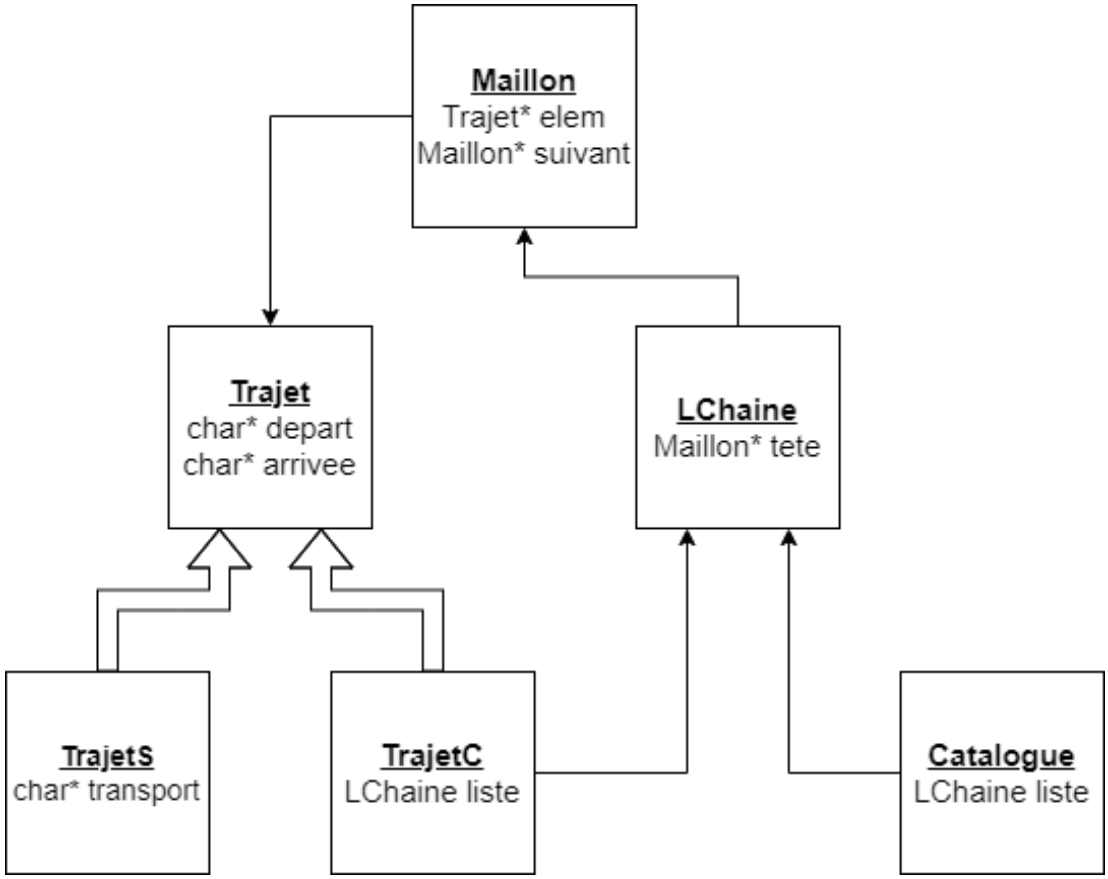
LChaine est une classe "gestionnaire" dont l'attribut pointe vers le premier maillon d'une liste chaînée de trajets.

- **La classe Catalogue**

La classe catalogue permet d'implémenter la liste chainée de tous les trajets. Donc l'attribut "LChaine liste" pointe vers le premier maillon de la liste chaînée de tous les trajets.

- **La classe Pile**

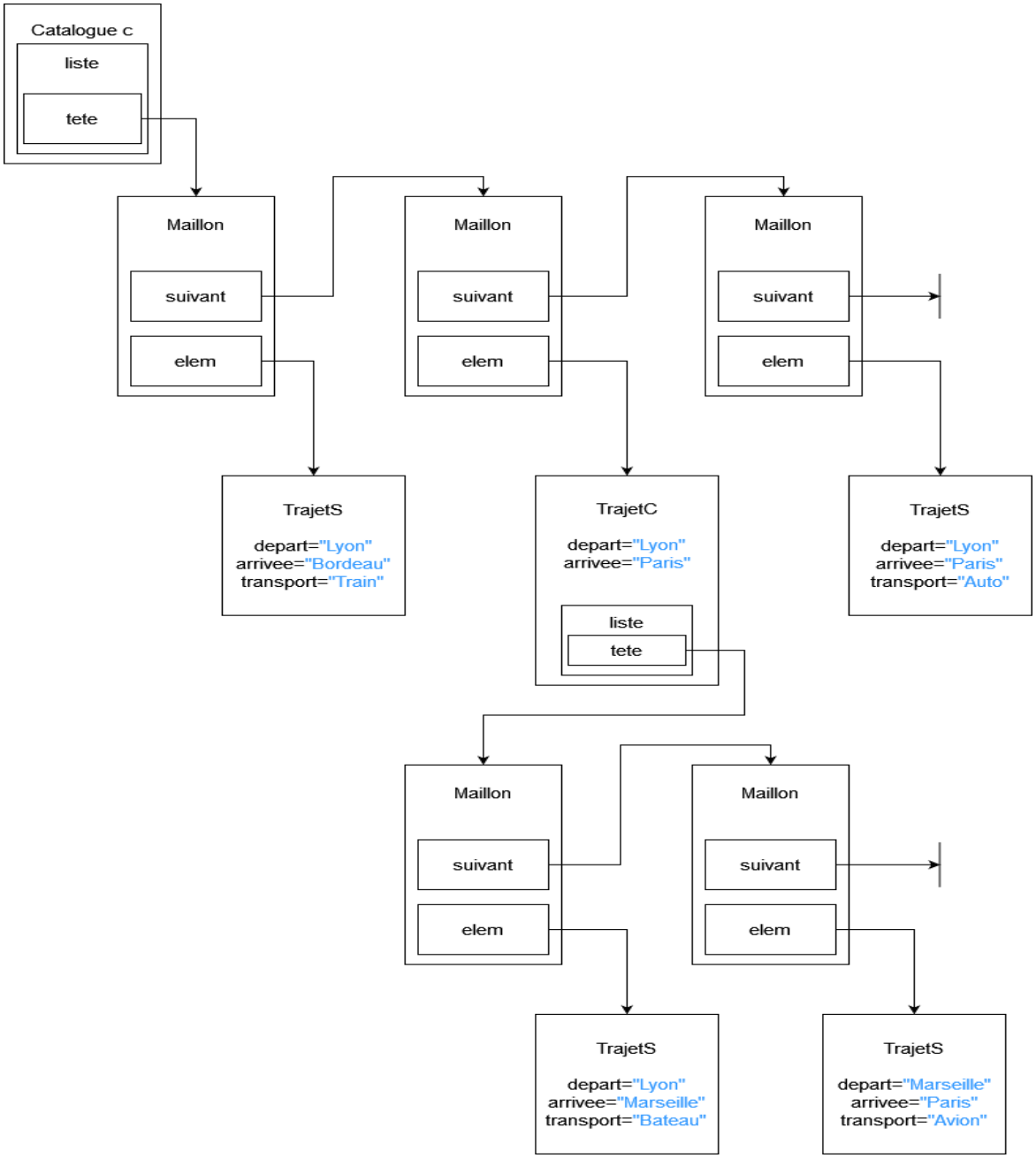
Cette classe permet d'implémenter la recherche avancée. On peut empiler ou dépiler des trajets. Cette pile est comme une liste chaînée mais nous ajoutons et enlevons les éléments à la fin car on veut pouvoir afficher la pile dans l'ordre d'arrivée.
Pour ne pas avoir une grande complexité, nous avons placé un pointeur en fin de liste comme attribut.



Titre : Graphe d'héritage

B. Description détaillée de la collection de donnée

On utilise une liste chaînée qui utilise la classe Maillon pour stocker les données. On a donc dans la classe LChaine, un pointeur vers un Maillon (la tête). Chaque Maillon a un pointeur vers un Trajet (un élément) et un pointeur vers le prochain Maillon. L'élément est un pointeur de trajet car il peut être un pointeur vers un trajet simple ou composé.



Titre : Jeu d'essai

C. Conclusion

- **Comment construire un Maillon avec un élément ?**

=> Quand on construit un Maillon, avec un pointeur de Trajet comme paramètre, on ne copie pas ce trajet (on évite de faire trop de copie). Il ne faut donc pas delete les Trajets pointés après qu'on les ait mit dans la liste.

- **Certains attributs doivent pouvoir être récupérés comme par exemple "suivant" dans un Maillon pour parcourir la liste. On doit même parfois modifier cet attribut lors de l'ajout.**

=> Implémentation de méthodes du type GetSuivant et SetSuivant.

- **L’affichage d’une liste chaînée doit se faire différemment en fonction de si on veut afficher le catalogue ou si on affiche un trajet composé :**

=> On a donc ajouté un paramètre sep pour “séparateur” qui nous permet d’afficher des sep entre chaque trajet affiché

- **L’ajout d’un élément dans une liste chaînée se fait différemment dans un catalogue et dans un trajet composé. On peut rechercher un élément dans un catalogue mais pas dans un trajet composé.**

=> Implémentation des fonctions d’ajout et de recherche dans la classe LChaine mais : on ne peut pas faire un appel à Rechercher avec une instance de trajet composé (encapsulation), on fait une fonction Rechercher dans catalogue qui appelle la fonction Rechercher de la liste chaînée. De même avec l’ajout.

- **La recherche avancée nécessite l’usage d’une pile :**

=> Ajout d’une classe Pile qui contient un pointeur vers un maillon, on peut empiler, dépiler des trajets et on peut tester si elle contient un trajet d’un départ à une arrivée.

- **On a besoin de comparer le départ et l’arrivée des trajets dans la recherche avancée :**

=> Implémentation des fonctions ArrIs et DepIs.

- **Pour afficher la pile, on a besoin de le faire dans l’ordre où les éléments sont arrivés.**

=> On empile et dépile à la fin car l’affichage doit se faire en un parcours de la tête à la fin.
=> On utilise un pointeur de fin pour toujours empiler et dépiler en temps constant.

Pour aller plus loin…

Il reste encore pas mal de choses à faire dans cette application. Par exemple, on pourrait vérifier que les trajets composés sont licites (ne pas s’arrêter plusieurs fois au même endroit), ajouter une fonctionnalité de recherche encore plus avancée qui considérerait que l’on peut changer de trajet à chaque escale d’un trajet composé, on pourrait donner un temps, un prix, une empreinte carbone aux trajets et déterminer le plus court chemin d’un point A à un point B, le moins coûteux en prix, en carbone,... On pourrait stocker le catalogue dans un fichier afin de ne pas avoir un catalogue vide au début...