**TP C++ n°2 : Héritage – Polymorphisme**

Constitution de voyages à partir d’un catalogue de trajets

1. Description des classes de l’application et du graphe d’héritage

Les différentes classes de notre application peuvent être catégorisées en deux sous-groupes. Il y a d’une part les objets que nous utilisons afin de créer le catalogue. Cela inclut la classe Catalogue, la classe Liste qui est utilisée par le catalogue, et la classe Maillon qui est utilisée par la liste. Et d’autre part nous avons les classes qui permettent la gestion des trajets, c’est-à-dire les classes Trajet, TrajetSimple et TrajetComposé. Les classes TrajetSimple et TrajetComposé héritent de Trajet. Nous aurions pu nous en passer, mais ce choix était assez logique d’un point de vue sémantique et il facilite grandement la compréhension de l’architecture de notre application. Enfin, on peut noter que pour ne pas créer une nouvelle structure de données, les trajets composés utilisent la classe Liste.

Le graphe d’héritage décrit dans le paragraphe ci-dessus est à retrouver en annexe.

**Catalogue**

Le catalogue contient une liste triée de trajets. C’est l’objet avec lequel l’utilisateur interagit directement lorsqu’il rentre ses instructions dans l’interface.

**Liste**

Une liste est une collection ordonnée de maillons, structurés sous la forme d’une liste simplement chaînée. La liste contient un pointeur sur le premier maillon ainsi qu’un pointeur sur le dernier maillon.

Nous utilisons des listes pour stocker des trajets : pour les trajets du catalogue mais aussi pour les trajets dont est constitué un trajet composé. Enfin, la recherche avancée de trajet est, elle aussi, réalisée avec des listes de trajets comme structure de données.

**Maillon**

Les maillons sont les objets dont sont constituées nos listes simplement chainées de trajets. Chaque maillon contient un pointeur sur un trajet, qui peut être simple ou composé, et un pointeur sur le prochain maillon. Le dernier maillon de la liste pointe sur le pointeur nul.

**Trajet**

Un trajet est constitué d’une ville de départ et d’une ville d’arrivée. C’est la classe dont héritent les trajets simples et composés.

**TrajetSimple**

La classe TrajetSimple hérite de la classe trajet. Néanmoins, un trajet simple stocke aussi le moyen de transport utilisé, en plus de la ville de départ et de celle d’arrivée.

**TrajetCompose**

La classe TrajetCompose hérite de la classe trajet. Un trajet composé et contient un pointeur sur une liste de trajets ainsi que la ville de départ et celle d’arrivée de cette liste de trajets.

C’est à l’utilisateur de faire en sorte que la ville de départ et celle d’arrivée soient cohérentes avec la liste de trajets fournis. C’est aussi à lui de faire en sorte que la liste elle-même soit cohérente, c’est-à-dire que l’arrivée d’un trajet soit le départ du suivant.

1. Description de la structure de données utilisée pour gérer les trajets

Pour gérer les trajets, nous avons utilisés une liste simplement chaînée. C’est une liste composée d’un pointeur sur le premier maillon. Ensuite, chaque maillon contient un pointeur sur un trajet et un pointeur sur le prochain maillon. Le pointeur sur le prochain maillon du dernier maillon de la liste est le pointeur nul.

Pour parcourir cette liste, il faut commencer au premier maillon puis récupérer le pointeur sur le prochain maillon, récupérer ce maillon, et ainsi de suite jusqu’à ce que le pointeur sur le prochain maillon soit le pointeur nul.

La liste comporte aussi un pointeur sur le dernier maillon. En effet, cela permet d’ajouter des maillons en queue de liste plus facilement et plus rapidement. Cela nous est très utile, notamment dans la recherche avancée où cette fonction est grandement utilisée.

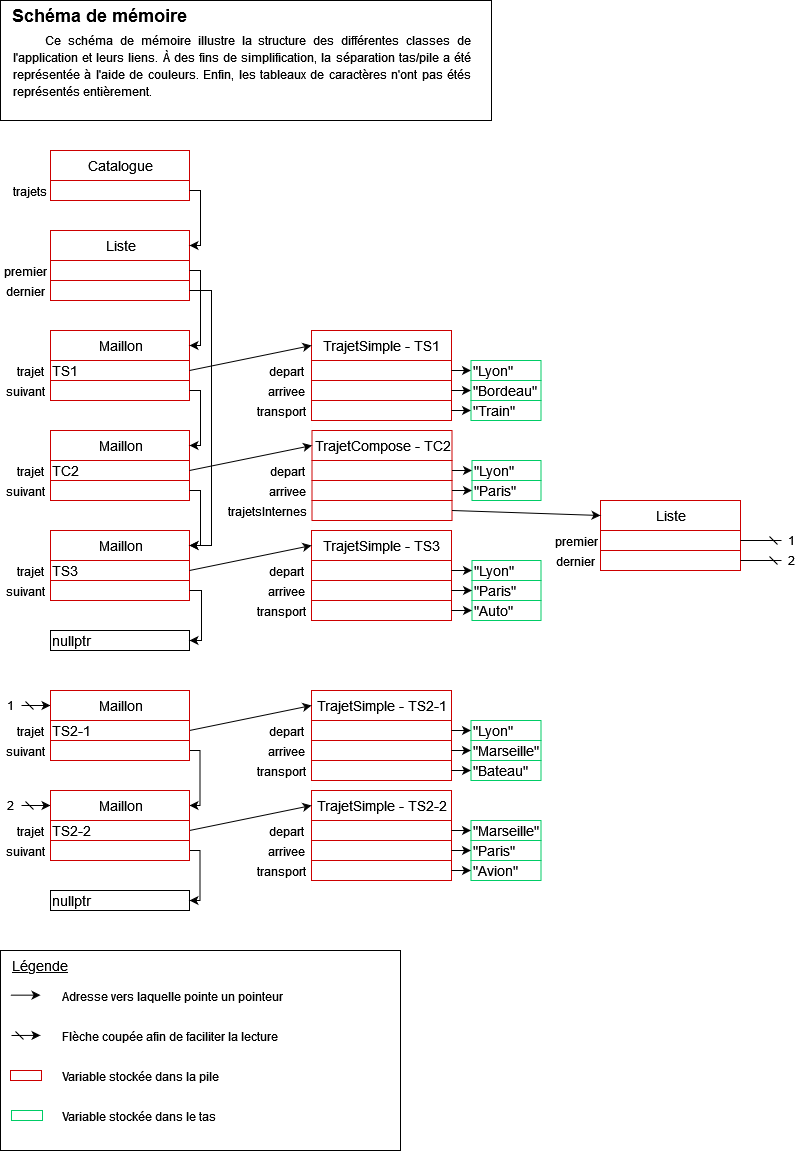
Les maillons sont ordonnés par ordre alphabétique des villes de départ des trajets puis des villes d’arrivée et enfin par ordre d’insertion si la ville de départ ainsi que la ville d’arrivée sont identiques.

**Exemple**

En annexe nous avons représenté la mémoire pour un jeu d’essaie fourni. On peut voir les différentes classes de notre application dans un contexte réel.

1. Conclusion : problèmes marquants rencontrés et axes d’évolution

* Choix de la structure de données pour la liste (tableau / liste chainée / etc…)
* Fonction « recherche de trajets » retourne une liste de trajets ou affiche juste les trajets
* Départs différents des arrivées : trajets touristiques ?
* Recherche avancée faite sans utiliser de nouvelle structure de donnée : utilisation de la liste simplement chaînée, elle ne prend pas non plus en compte les boucles

Dessin de la mémoire pour le jeu d’essai fourni