

Compte-rendu TP n°2 C++

I. Description générale des classes / Héritage

I.1. Classe Trajet

Un trajet est composé d'un point de départ (start) et d'une arrivée (end).

Cette classe est abstraite dans la mesure où un trajet devra être soit simple soit composé (mais pas juste un trajet tout court). Les attributs et méthodes implémentés ici seront utilisés par toutes les classes héritières.

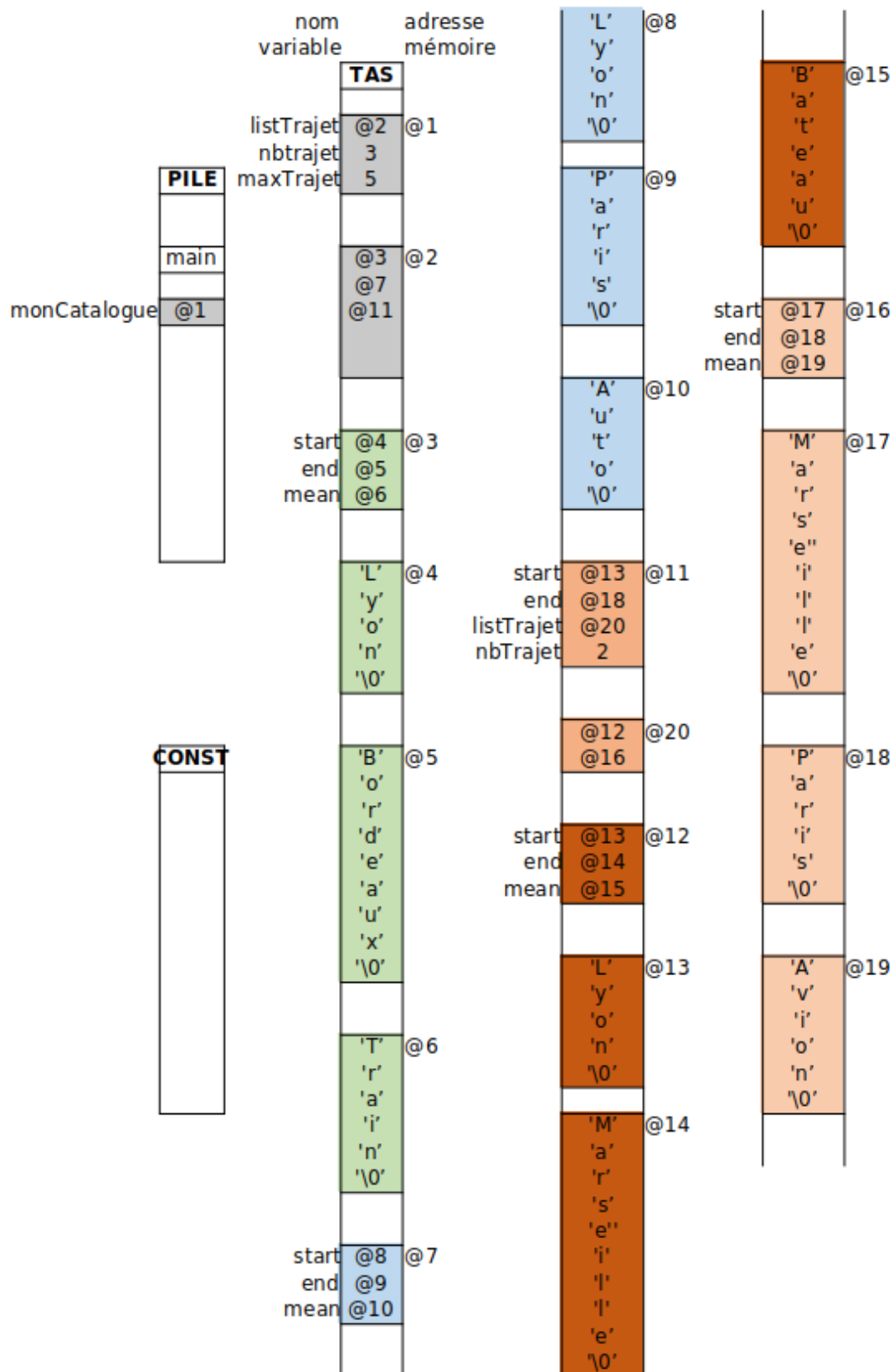
I.2. Classe TrajetSimple

Un trajet simple hérite de Trajet en spécifiant en plus le moyen de transport. En effet un trajet simple ne sera composé que de deux lieux (départ et arrivé, contrairement à un trajet composé).

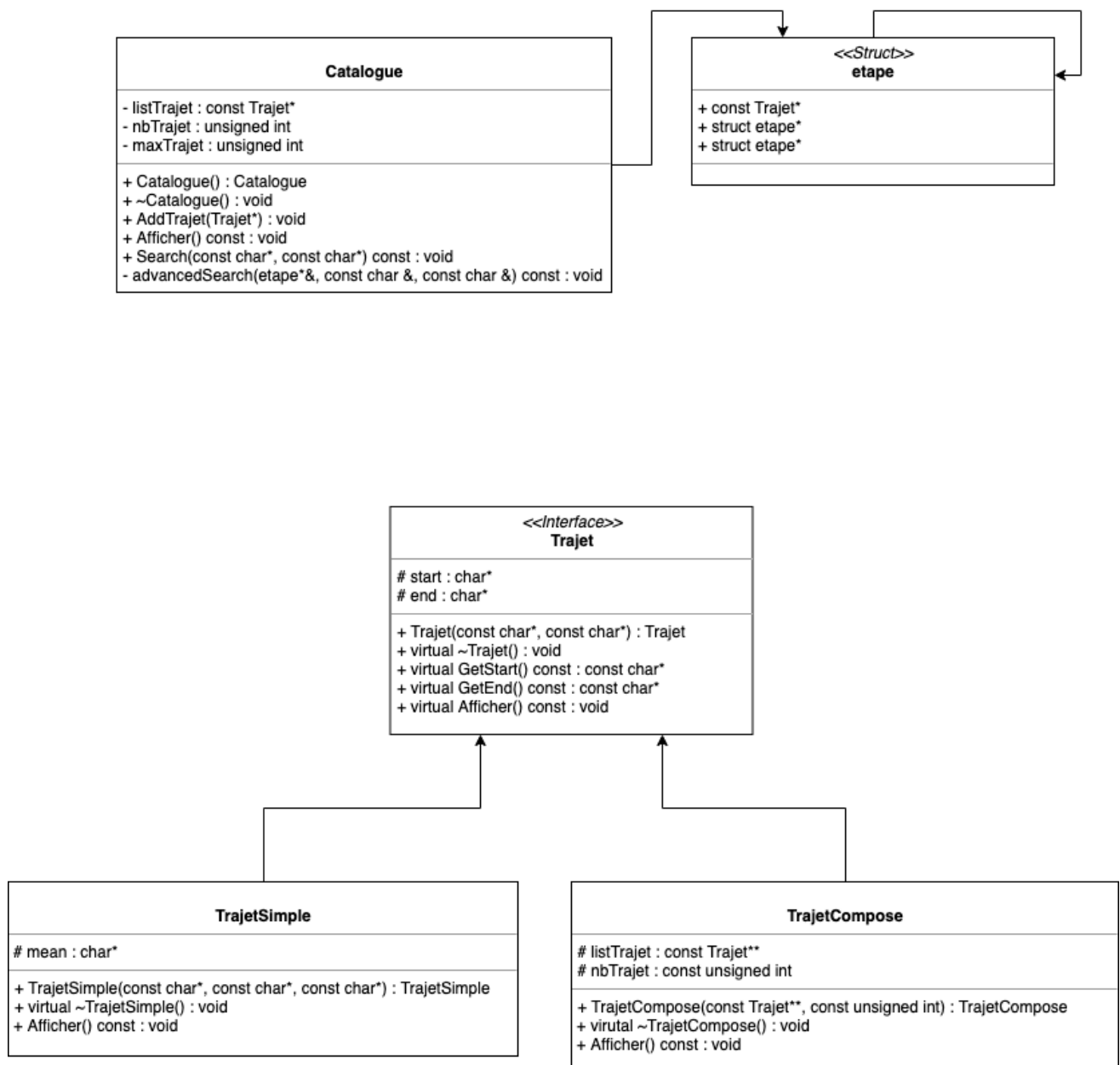
I.3. Classe TrajetComposé

Un trajet composé hérite de Trajet. Il s'agit en fait d'une liste de trajet simples. Le fait d'avoir une telle organisation des classes permet à chaque trajet simple d'un trajet composé de disposer de son propre moyen de transport.

II. Dessin de la structure de donnée (exemple concret)



III. UML



IV. Listing des interfaces et réalisations

- IU : demo.cpp
- Catalogue : Catalogue.h / Catalogue.cpp
- Trajet : Trajet.h / Trajet.cpp
- Trajet simple : TrajetSimple.h / TrajetSimple.cpp
- Trajet composé : TrajetCompose.h / TrajetCompose.cpp

V. Problèmes rencontrés

Lors de l'écriture de l'UML nous avons déclaré des const partout où il nous le semblait nécessaire. Cependant, nous nous sommes rabattus sur l'idée d'en mettre moins car nous pensions que certaines variables ne devaient pas l'être. Mais finalement, l'UML était correct : il a donc fallu remplacer certains const. Nous avons donc appris que des variables dynamiques pouvaient être déclarés const.

Concernant la fuite de mémoire, l'utilisation de Valgrind nous a permis de remplacer les delete par delete[] pour désallouer un tableau ainsi que de mettre en évidence que nous avions oublié d'implémenter le destructeur de trajet composé en entier.

V. Axes d'évolution

Une première possible serait d'ajouter un service au sein de notre classe Catalogue qui permettrait de supprimer un trajet.

Deuxièmement, nous pourrions effectuer une vérification de jointure entre deux trajets consécutifs d'un trajet composé.

En troisième lieu, nous n'avons pas eu besoin d'implémenter les constructeurs de copie ainsi que l'opérateur d'affectation pour les classes ayant des attributs alloués dynamiquement (Catalogue, Trajet, TrajetSimple et TrajetCompose) : il serait préférable de les coder avant de distribuer le produit.

Aussi, nous avons implémenté la listTrajet dans la classe Catalogue sous forme de Trajet**. Alors qu'une implémentation un peu plus poussée en table de hachage aurait permis de bien meilleures performances pour l'exécution de l'algorithme de recherche avancée.

En dernier lieu, il serait utile de pouvoir créer un nouveau trajet composé à partir d'autres trajets composés.