RAPPORT DE PROJET

CONSTITUTION DE VOYAGES A PARTIR D'UN CATALOGUE DE TRAJETS

14 DECEMBRE 2020

JIN FRANCINE, TRAN QUANG HUY
INSA LYON 2020/2021 – 3IF210

Table des matières

INTRODUCTION

Le TP consiste à réaliser une application proposant à l'utilisateur une interface avec différentes actions possibles. Il doit taper le numéro correspondant à une des actions proposées dans le menu afin de lancer la fonctionnalité correspondante.

Le but principal de cette application est de permettre à l'utilisateur de chercher les trajets disponibles dans le catalogue lui permettant d'aller d'une ville A à une ville B qu'il aura rentrées sur le terminal.

Pour ce faire, plusieurs entités et fonctionnalités doivent être implémentées.

Le **Catalogue** est la classe manipulée dans le main(), point d'entrée de l'application. Toutes les fonctionnalités proposées dans l'interface le prennent en paramètre. C'est une liste chainée de Trajet. Cette classe stocke ainsi tous les trajets crées et permet à l'utilisateur de faire des recherches d'itinéraires, d'ajouter de nouveaux trajets dans l'ordre alphabétique selon la ville de départ et de tous les afficher.

La classe abstraite **Trajet** est la base de l'application. En effet, le catalogue contient des trajets qui pourront être simples ou composés, Trajet fait donc office de classe mère, exploitable par les autres classes sans qu'elles aient besoin de se soucier des spécificités de TrajetSimple et TrajetCompose. Elle est constituée de deux pointeurs de chaines de caractères passés en paramètre dans le constructeur, permettant de stocker un trajet d'une ville de *depart* A à une ville d'*arrivee* B. On peut obtenir avec getDepart() et getArrivee, son départ et son arrivée mais pas les modifier. Sa méthode afficher() est en visual et sera défini par ses descendants.

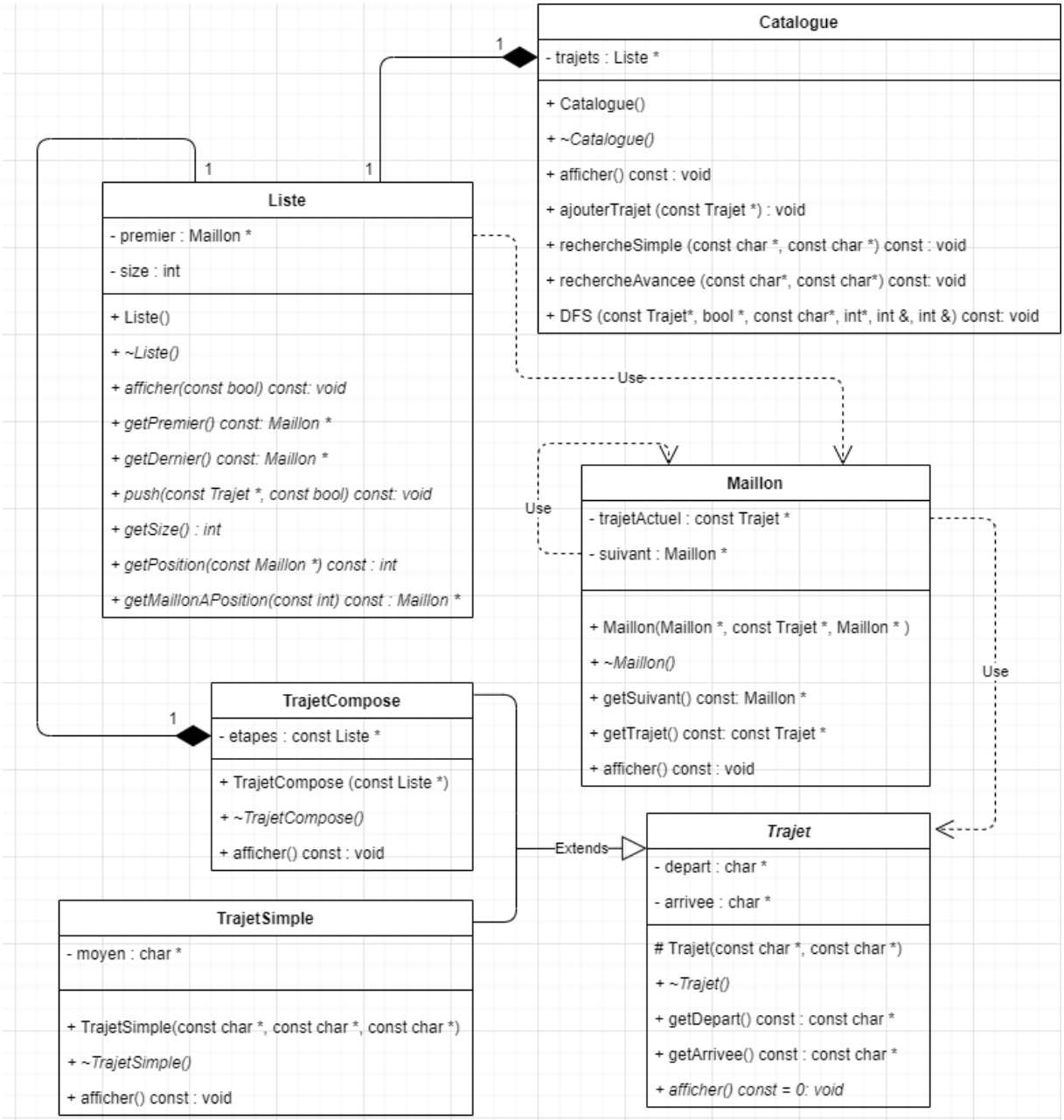
La classe **TrajetSimple** hérite de l'interface de la classe Trajet et donc de toutes ses fonctions et attributs. De plus, elle a l'attribut *moyen*, un pointeur de chaine de caractères. Sa méthode afficher() va afficher *depart*, *arrivee* (hérités de Trajet) mais aussi de *moyen*. Cet attribut lui est spécifique, et son afficher() sera réutilisée dans afficher() de TrajetCompose.

La classe **TrajetCompose** hérite de l'interface de la classe Trajet et donc de toutes ses fonctions et attributs. Contrairement à TrajetSimple, elle n'a pas l'attribut moyen mais elle contient à la place une Liste de Trajet. Un TrajetCompose peut être une suite de TrajetSimple, de TrajetCompose ou un mélange. Elle permet de stocker un itinéraire d'une ville A à une ville B en faisant des escales par d'autres villes.

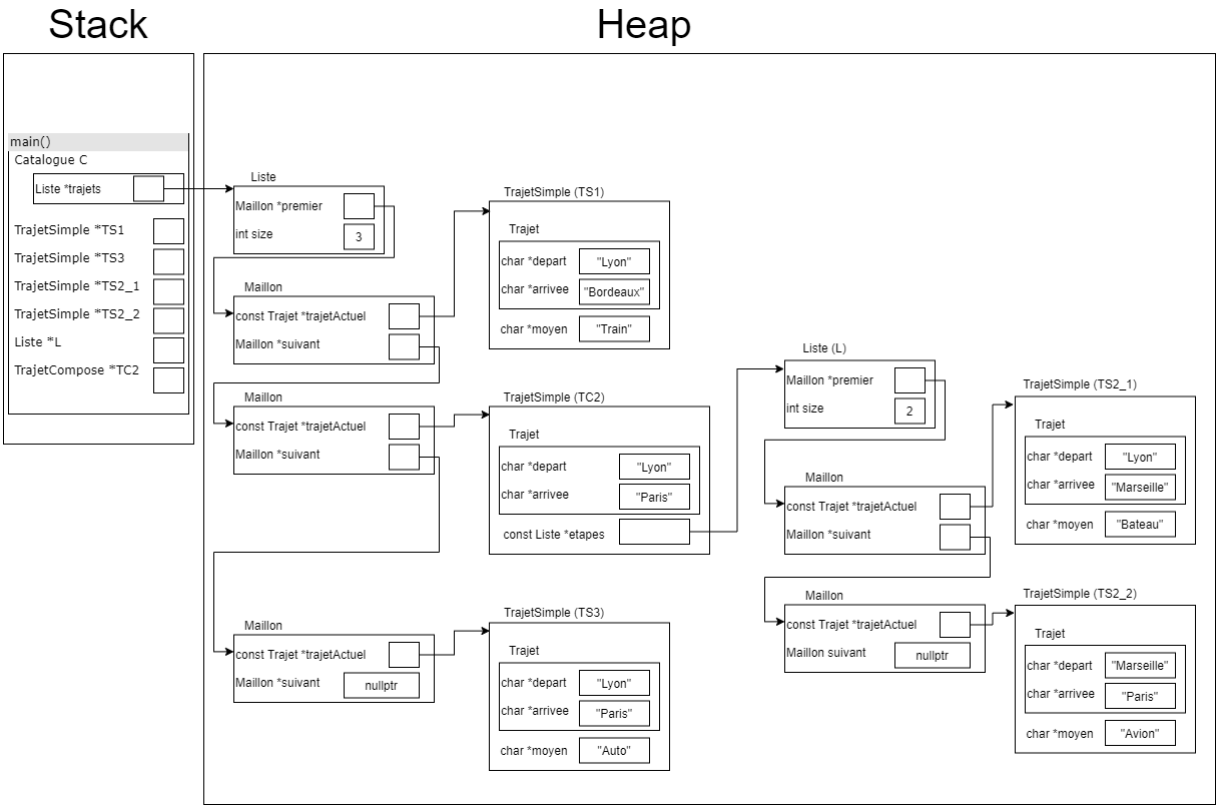
La classe **Maillon** existe à la place de struct Maillon qu'on ne peut pas utiliser. Elle est composée de d'un pointeur sur Maillon et d'un pointeur sur Trajet. C'est ce qui compose Liste. *suivant* pointe vers l'emplacement du prochain maillon et s'il n'existe pas, il est un pointeur null. *trajetActuel* permet d'obtenir des informations sur le trajet en cours.

La classe ***Liste*** est une classe équivalente à la classe List en C++ que nous avons réalisée. Nous en avons fait une liste chainée car c'est plus flexible au niveau du stockage, Catalogue et TrajetCompose n'ayant pas une limite de trajets qu'elles peuvent ajouter, un tableau avec une taille fixe ne convenait pas. Liste est composée de deux Maillon, *premier* pointant vers le début de la liste. La classe a aussi un int pour stocker la taille de la liste. On peut ajouter en plein milieu d'une liste un trajet via un nouveau maillon A juste en changeant le pointeur *suivant* du maillon Z se trouvant à l'emplacement cible, et ce qui rend cette classe très pratique.

DIAGRAMME DE CLASSE



Structure de données



Ressentis et améliorations possibles

Le projet en soi n'était pas compliqué mais nous avons pris du temps à démarrer. En effet, il fallait faire valider la structure de données qu'on allait utiliser avant de pouvoir coder, et notre diagramme UML a été refusé plusieurs fois. On ne pensait pas assez loin sur la manière dont allait être stocké les trajets. On pensait à une liste chainée mais nous n'avons pas montré comment elle allait être réalisée.

Il y a eu beaucoup de contraintes imposées par le sujet, notamment l'interdiction d'utiliser *struct*, la classe *String* et les librairies autre que *iostream* et *cstring*. Nous avons dû nous adapter et prendre du temps pour créer les classes et les méthodes qui de base existaient déjà en C++, par exemple la classe *List* (dans notre cas Liste) ou la classe *Etape* à la place d'un struct *Node* dans *Liste*. Cela nous a compliqué la tâche sur Etape où on a trouvé au dernier moment un moyen d'enlever Get et Set sur un attribut privé pour mettre à jour l'attribut *suivant*.

Enfin, il fallait penser à l'intégrité des données stockées et passées en paramètre. On a dû jouer sur les *const* : quand on en ajoutait un quelque part, il fallait vérifier que ça ne créé pas d'erreur ailleurs.

À la suite d'un manque de temps, nous n'avons pas pu améliorer le projet notamment sur certains points :

- Ajouter des vérifications sur les trajets simples tel qu'il n'existe pas deux trajets ayant la même ville de départ, d'arrivée et le même moyen de transport (même avec une casse différente) enregistrés dans le catalogue.
- Transformer les moyens de transport en Enum pour limiter les choix de l'utilisateur et ainsi réduire le risque de se retrouver avec deux trajets identiques ayant juste la casse du moyen de transport différente (par ex : 'Avion' et 'avion').
- Etendre la fonction Recherche en ajoutant la possibilité que la ville d'arrivée B souhaitée soit une escale d'un autre trajet et non obligatoirement la ville d'arrivée. Ainsi le nombre de trajets possibles entre la ville A et B augmenterait.

Le projet fut très enrichissant et instructif, notamment sur la manipulation des pointeurs et de la mémoire.