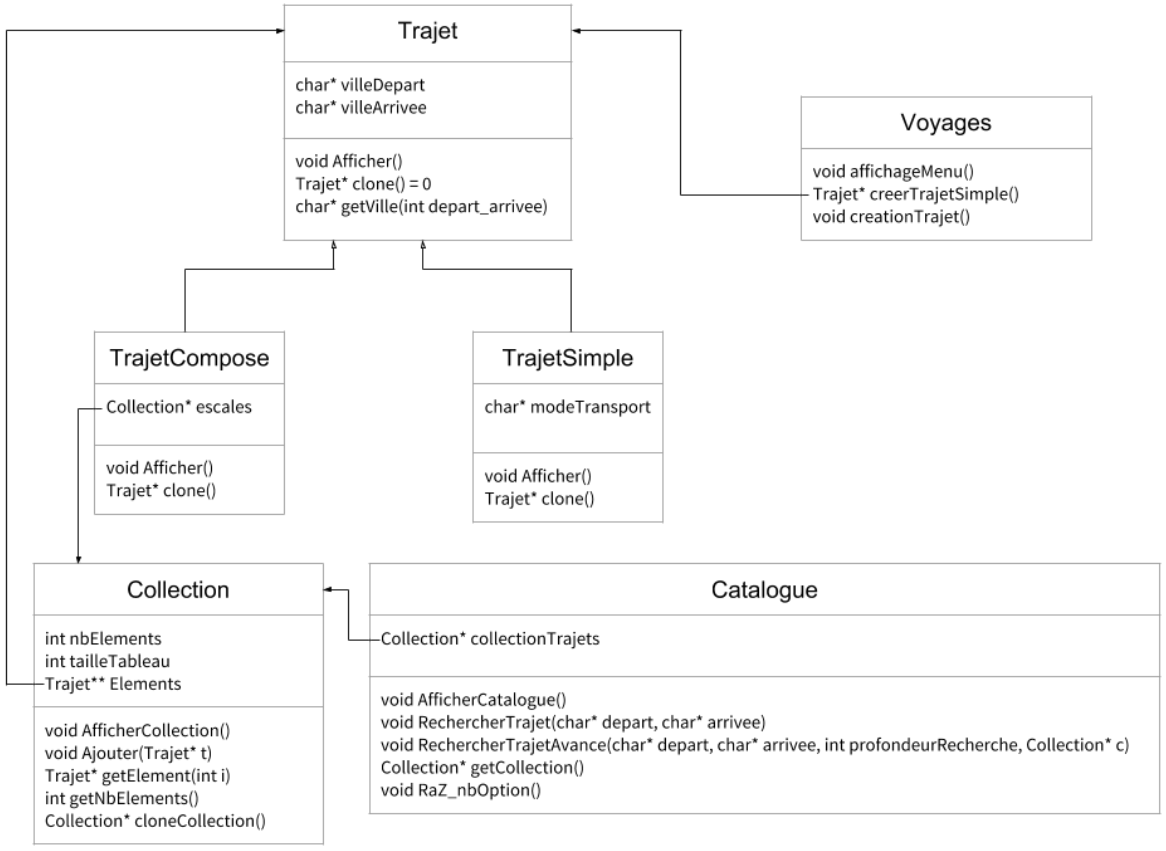
Mathis Guilhin
Téo Bouvard 3IF

TP C++ : Héritage - Polymorphisme

1. Classes

    a. Graphe d'héritage



    b. Description des classes

La classe <Trajet> constitue la brique de base de cette application. C'est une classe abstraite car on ne veut jamais instancier de simple <Trajet>, mais uniquement des <TrajetSimple> ou des <TrajetCompose>. Un <Trajet> possède deux attributs : une ville d'arrivée et une ville de départ.

La classe <TrajetSimple> est une sorte de Trajet, et hérite donc de cette classe. Elle possède néanmoins un attribut spécifique qui est le mode de transport.

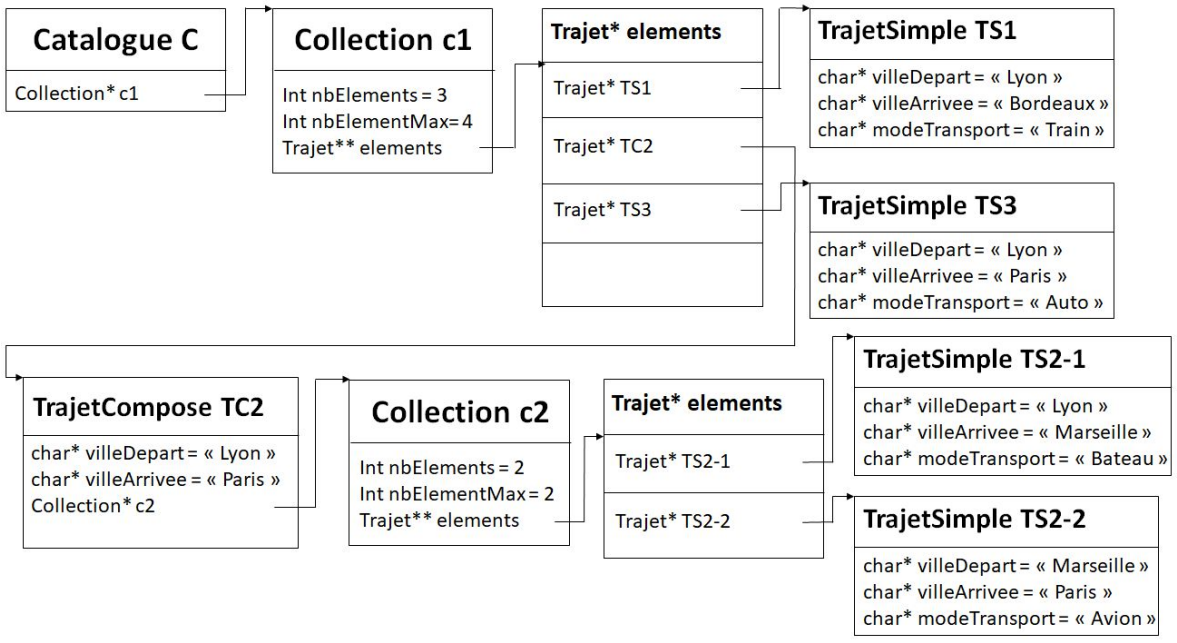
Au même titre que <TrajetSimple>, <TrajetCompose> hérite de <Trajet> et possède ses deux attributs. Cependant, <TrajetCompose> ne partage pas l'attribut "modeTransport" de <TrajetSimple> car il est composé de plusieurs escales qui peuvent être réalisée avec des modes de transport différents. Les différentes escales qui le compose sont stockées dans une <Collection>.

La <Collection> est la structure de donnée utilisée pour qui permet de stocker des pointeurs sur des <Trajet> dans un tableau dynamique. Les attributs de <Collection> sont le tableau dynamique, le nombre d'éléments qu'il contient et la taille actuelle du tableau dynamique. Afin d'éviter la redondance d'information, c'est la seule classe qui compte le nombre de <Trajet> qu'elle contient. On utilisera le service getNbElements() de cette dernière pour accéder au nombre de trajets dans <TrajetCompose> ou dans le <Catalogue>.

La classe <Catalogue> représente un catalogue de trajets et n'est composé que d'une <Collection>. Cette classe contient les fonctions de recherche simple et avancée.

Enfin, le module <Voyage> implémente ces différentes classes et fournit une interface graphique à l'utilisateur. Il contient la fonction main ainsi que deux fonctions qui permettent de confectionner des voyages avant de les ajouter au Catalogue.

2. Structure de données - Exemple



2. Difficultés

a. Conception

Une des phases les plus complexe et déroutante fut la conception. Nous avons eu beaucoup de mal à imaginer les problèmes que l'on allait rencontrer, et également à trouver une structure de donnée adaptée au problème.

b. Gestion de la mémoire

Le tableau dynamique de pointeurs “elements”, attribut de <Collection>, n’a pas été évident à visualiser dans un premier temps. C’est la première fois que nous manipulons des pointeurs de pointeurs, et il nous a fallu effectuer des tests unitaires afin de se familiariser avec son utilisation.

Nous avons aussi eu des difficultés à déterminer où détruire chaque objet créé. L’utilisation de valgrind (avec les options --leak-check=full --show-leak-kinds=all) et le traçage des constructeurs/destructeurs nous a permis d’identifier les sources des fuites de mémoire et les erreurs de lecture.

c. Récursivité

L’utilisation de méthodes récursives pour la recherche avancée et pour la construction du catalogue nous a permis de toucher du doigt la difficulté mais aussi l’efficacité de ce type d’algorithme.

d. Redondance d’information

Lors de notre conception initiale, de nombreux attributs étaient redondants. Par exemple, <TrajetCompose>, <Collection> et <Catalogue> possédaient chacune un attribut qui correspondait au nombre de trajets contenus. Nous avons préféré utiliser un getter dans <Collection> et se débarrasser des attributs redondants.

e. Entrées utilisateur

Pour l’interface, nous avons décidé que la variable qui contenait le choix de l’utilisateur serait de type entier. Cependant, lorsque ce dernier saisit autre chose qu’un entier, le programme plante et il faut le redémarrer.
Ce point constitue l’axe majeur d’évolution de notre application. Afin de ne considérer que les saisies valides de l’utilisateur, on pourrait tester l’état du flux cin à l’aide de la méthode fail() de la classe basic_ios. Ainsi, seuls des entiers seraient lus et le programme ne planterait plus dans ce cas là. Nous avons testé l’application avec cette interface et cela fonctionne, mais ne respecte pas la consigne d’utilisation des librairies.
Afin de se débarrasser complètement du problème, il conviendrait de fournir une interface graphique aboutie qui limiterait les choix du menu à des boutons.