Trajets

Marcos Bauch Mira
(Je n'ai pas un numéro binôme à cause d'être en Erasmus)

Structure du code

Le code est divisé en plusieurs classes, chacune avec son propre fichier d'en-tête (classe.h) et de source (classe.cpp), ils sont dans un dossier appelé « classes » :

1. **Trajet** (trajet.h, trajet.cpp) : Une classe abstraite représentant un trajet. Elle contient des méthodes virtuelles pures pour afficher le trajet et obtenir les points de départ et d'arrivée.
2. **TrajetSimple** (trajetsimple.h, trajetsimple.cpp) : Une classe dérivée de Trajet représentant un trajet simple avec un moyen de transport spécifié. Elle implémente les méthodes de Trajet et inclut des membres **(char et pas char* !)** pour stocker les points de départ et d'arrivée, ainsi que le moyen de transport.
3. **TrajetCompose** (trajetcompose.h, trajetcompose.cpp) : Une classe dérivée de Trajet représentant un trajet composé de plusieurs trajets simples. Elle stocke un tableau de pointeurs vers des trajets simples, ainsi que le nombre de trajets.
4. **Catalogue** (catalogue.h, catalogue.cpp) : Une classe pour gérer un catalogue de trajets. Elle peut ajouter des trajets, les afficher et effectuer des recherches avancées.

La décision de rendre **Trajet** virtuel est de pouvoir utiliser **TrajetSimple** ou **TrajetCompose** (qui héritent de Trajet bien sûr) dans **Catalog** sans avoir à vérifier le type de la classe avant le calcul.

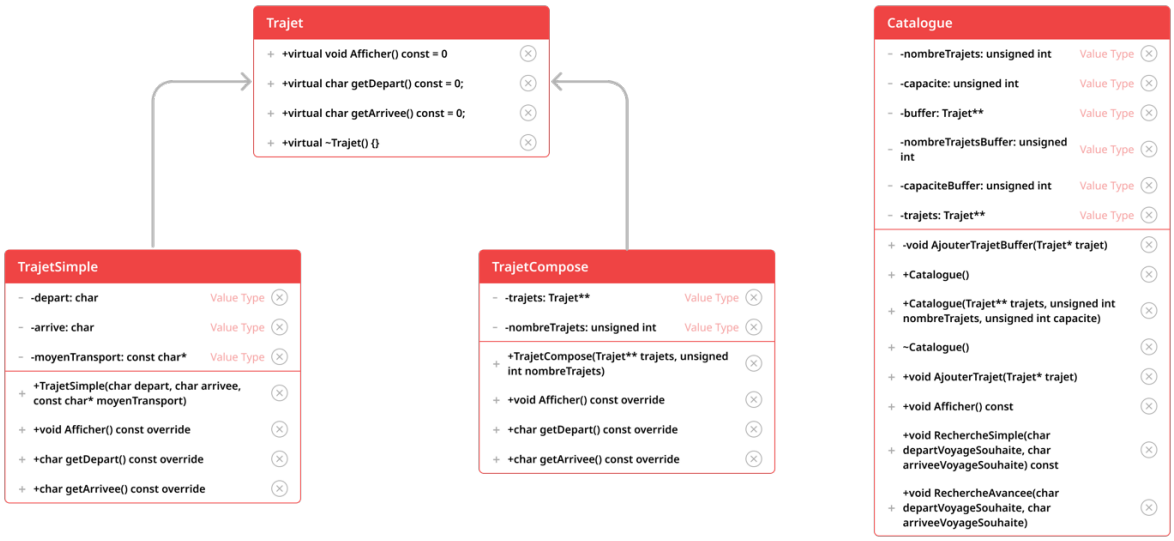
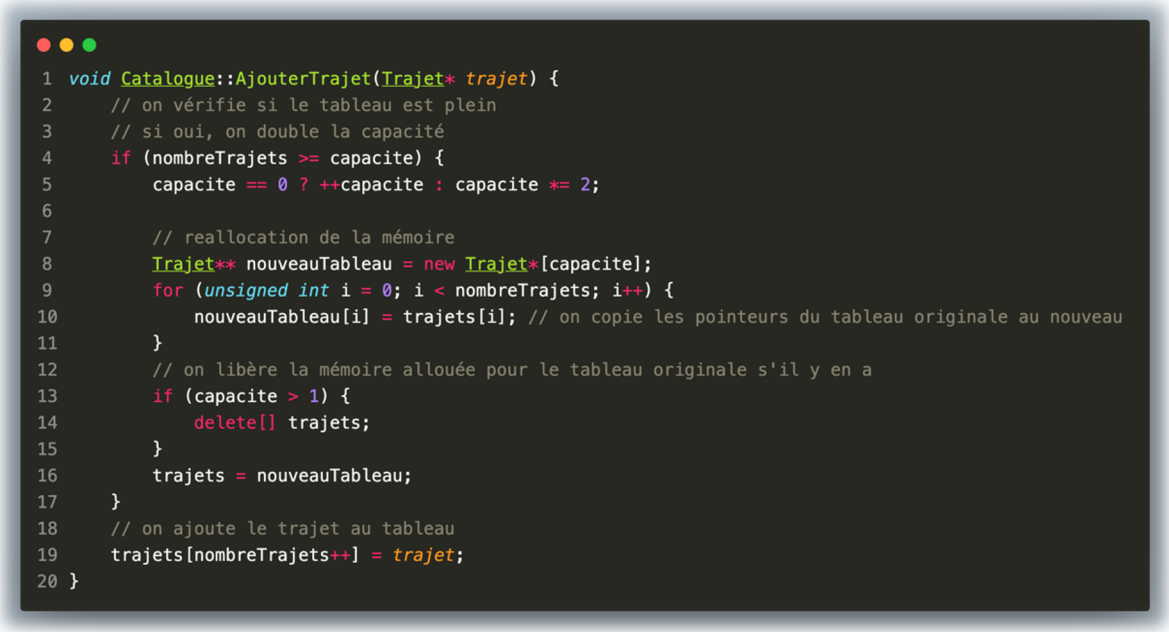


Diagram UML des Classes

Défis

« Memory Leaks » (Fuites de mémoire) : Grâce à l'outil valgrind, les fuites de mémoire ont été corrigées. Bien sûr, à chaque fois qu'on a utilisé l'opérateur **new** pour allouer de la mémoire, on a dû éliminer les traces avec l'opérateur **delete**. Aussi, les tableaux dynamiques sont redimensionnés lorsque nécessaire pour éviter les débordements de mémoire. Un exemple est présenté ci-dessus :



```
void Catalogue::AjouterTrajet(Trajet* trajet) {
    // on vérifie si le tableau est plein
    // si oui, on double la capacité
    if (nombreTrajets >= capacite) {
        capacite == 0 ? ++capacite : capacite *= 2;

        // reallocation de la mémoire
        Trajet** nouveauTableau = new Trajet*[capacite];
        for (unsigned int i = 0; i < nombreTrajets; i++) {
            nouveauTableau[i] = trajets[i]; // on copie les pointeurs du tableau originale au nouveau
        }
        // on libère la mémoire allouée pour le tableau originale s'il y en a
        if (capacite > 1) {
            delete[] trajets;
        }
        trajets = nouveauTableau;
    }
    // on ajoute le trajet au tableau
    trajets[nombreTrajets++] = trajet;
}
```

Utilisation du Programme

Suivant la consigne, le programme peut se compiler et exécuter avec les commandes « make » et « ./trajets » respectivement. Après vous aurez un menu interactif simple qui vous permet de créer des trajets simples et composés, d'afficher le catalogue et d'effectuer une recherche simple et/ou avancée de ces trajets. Ce sont les options permis à l'utilisateur, ainsi que la sortie du programme bien sûr.