

RAPPORT TP2 - C++

Héritage et Polymorphisme : Gestionnaire de Trajets

Binôme n° : [B3403] Membres : [BENCHEIKH Hamza] & [XUE Yuchen] Date : Décembre 2025

A. Description des Classes de l'Application

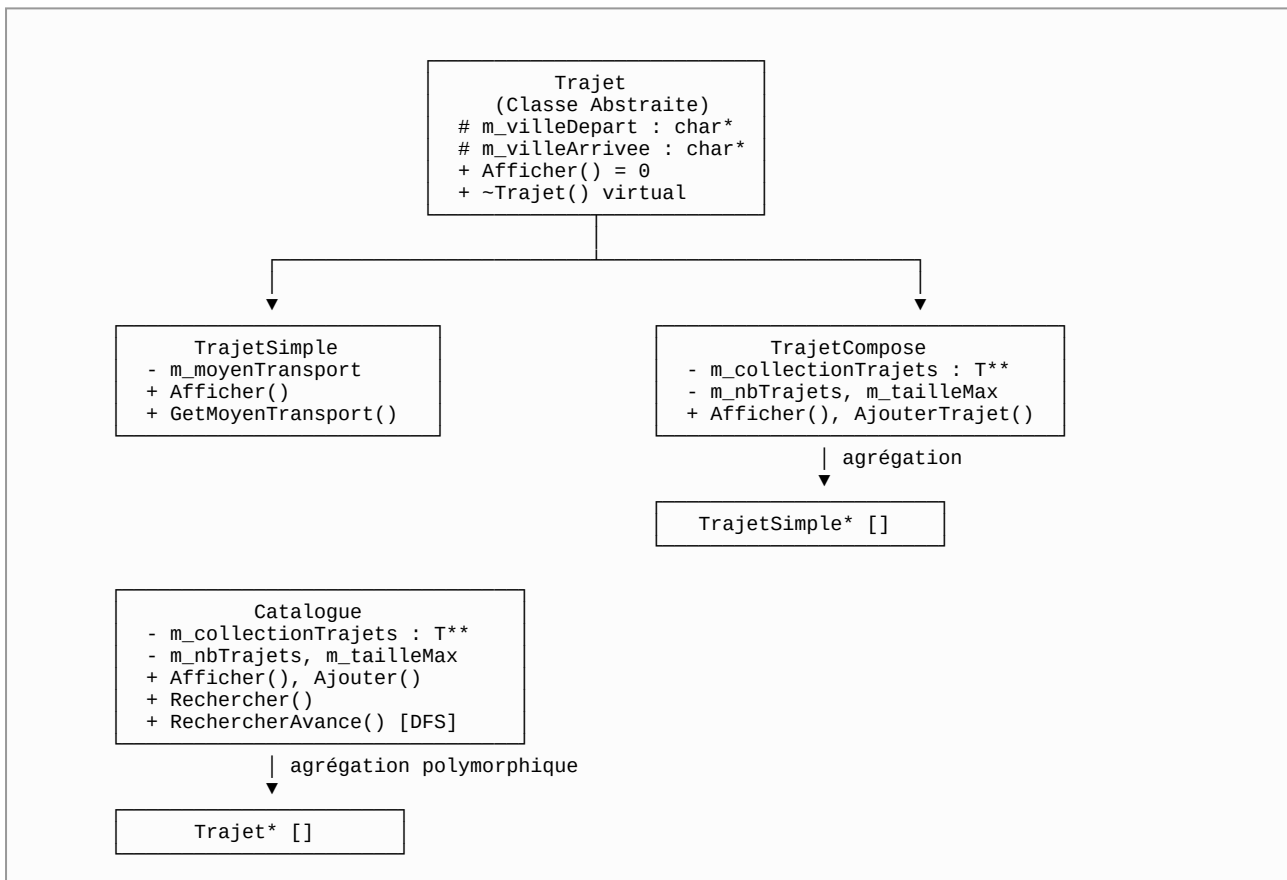
1. Architecture des Classes

- **Trajet (Abstraite)** : Interface commune avec villes départ/arrivée, méthode virtuelle pure `Afficher()`, destructeur virtuel.
- **TrajetSimple** : Hérite de `Trajet`. Ajoute un moyen de transport (`char*`).
- **TrajetCompose** : Hérite de `Trajet`. Agrège une collection de `TrajetSimple`.
- **Catalogue** : Gestionnaire avec tableau dynamique polymorphe de `Trajet*`.

2. Méthodes Clés

- `Catalogue::Ajouter(Trajet* t)` : Ajoute un trajet, gère la réallocation dynamique.
- `TrajetCompose::Ajouter(TrajetSimple* t)` : Ajoute une étape, vérifie la continuité.
- `Afficher()` : Méthode polymorphe pour lister le contenu.

3. Graphe d'Héritage



4. Justification de la Conception

Le choix de l'héritage permet au **Catalogue** de ne gérer qu'un seul type de donnée : **Trajet***. Cela respecte le principe Ouvert/Fermé (Open/Closed Principle).

Caractéristique	Trajet	TrajetSimple	TrajetCompose
Villes (Dep/Arr)	Protected	Héritées	Héritées
Moyen Transport	-	Privé (char*)	-
Conteneur Enfants	-	-	Tableau Dyn.
Cycle de vie	Virtual Destructor	Destructor	Deep Destructor

B. Structure de Données et Mémoire

Cette partie illustre la gestion mémoire pour le jeu d'essai "Catalogue C" demandé.

Jeu d'essai C :

1. **TS1** : Lyon → Bordeaux (Train)
2. **TC2** : [Lyon → Marseille (Bateau)] + [Marseille → Paris (Avion)]
3. **TS3** : Lyon → Paris (Auto)

Le schéma ci-dessous détaille le **Tas (Heap)**. Il met en évidence :

1. Le tableau principal de pointeurs **Trajet**** dans le Catalogue.
2. Les objets concrets (TS1, TC2, TS3) avec leur pointeur virtuel (**vptr**).
3. L'agrégation interne au Trajet Composé (TC2 possède son propre tableau dynamique).

1. Structure du Catalogue

Le Catalogue utilise un tableau dynamique de pointeurs **Trajet****. Chaque case pointe vers un objet concret (**TrajetSimple** ou **TrajetCompose**). Le polymorphisme permet d'appeler la bonne méthode **Afficher()** selon le type réel de l'objet.

2. Gestion des Chaînes de Caractères

Chaque attribut de type chaîne (**m_villeDepart**, **m_villeArrivee**, **m_moyenTransport**) est un pointeur **char*** vers un bloc mémoire alloué séparément sur le tas. La taille de chaque bloc correspond à **strlen(chaine) + 1** pour inclure le caractère nul **'\0'**.

3. Pointeur Virtuel (vptr)

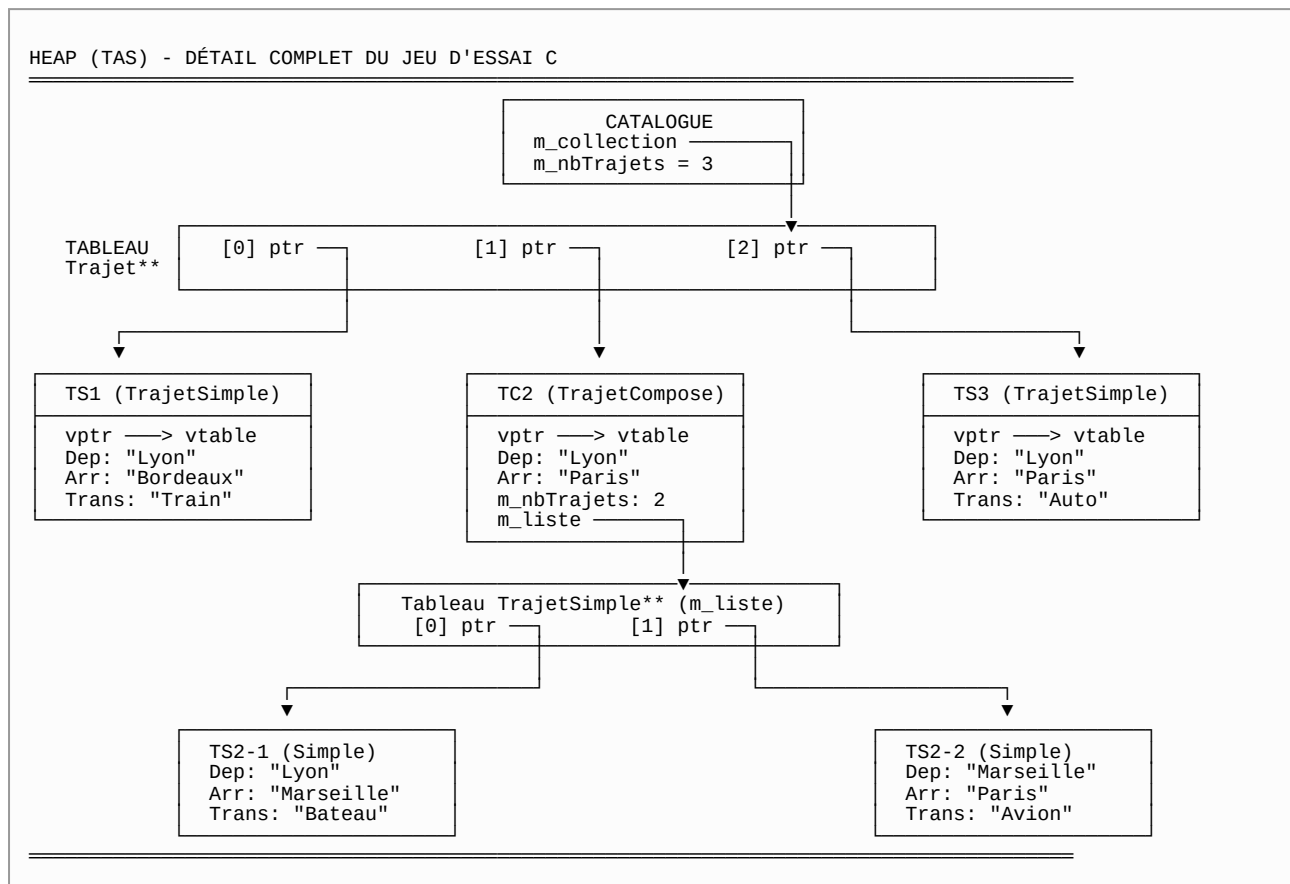
Chaque objet polymorphe contient un pointeur caché (**vptr**) vers sa table de fonctions virtuelles (**vtable**). C'est ce mécanisme qui permet au compilateur de résoudre les appels de méthodes virtuelles à l'exécution (late binding).

4. Agrégation dans TrajetCompose

Un **TrajetCompose** contient son propre tableau dynamique de **TrajetSimple****. Les villes de départ et d'arrivée du composé sont calculées dynamiquement : départ = premier sous-trajet, arrivée = dernier sous-trajet.

Représentation Mémoire Détaillée

Voici la vue précise des blocs mémoire alloués sur le tas (Heap) :



C. Problèmes Rencontrés et Améliorations

1. Gestion de la Mémoire

Destructeur Virtuel : La classe `Trajet` déclare un destructeur `virtual`. Sans cela, `delete trajet*` n'appellerait que le destructeur de base, causant des fuites mémoire.

Destruction Profonde : Le destructeur du `Catalogue` et de `TrajetCompose` parcourt le tableau et appelle `delete` sur chaque élément avant de libérer le tableau.

2. Copie Profonde (Deep Copy).

La copie par défaut est superficielle : elle copie les adresses, pas les données. Nous avons donc redéfini le constructeur de copie et l'opérateur d'affectation pour `TrajetSimple` (et `TrajetCompose`) afin de réallouer chaque `char*` et de recopier les chaînes avec `strcpy`, en respectant la règle des trois.

3. Bug d'Entrée / Affichage

Un bug d'affichage (inversion paramètres) a été corrigé. De plus, l'interdiction de `getline` impose l'usage de `cin >>` : les villes composées doivent utiliser un underscore (ex: `Le_Havre`) car l'espace coupe la saisie.

4. Constante MAP et Valgrind

La constante MAP entoure des messages de trace dans tous les constructeurs/destructeurs. En compilant avec -DMAP, nous avons suivi la création et la destruction de chaque objet, puis vérifié l'absence de fuites mémoire avec valgrind (résultat : "All heap blocks were freed -- no leaks are possible").

5. Recherche Avancée par DFS

La recherche avancée utilise un DFS récursif avec backtracking pour explorer toutes les combinaisons de trajets. Complexité exponentielle, mais adaptée à la taille du catalogue.

6. Axes d'Amélioration

Problème	Solution Proposée
Gestion manuelle new/delete	Utiliser <code>std::unique_ptr</code> ou <code>std::shared_ptr</code> pour fiabiliser la gestion mémoire.
Tableaux C (Trajet**)	Remplacer par <code>std::vector<Trajet*></code> ou <code>std::vector<std::unique_ptr<Trajet>></code> .
Chaînes char*	Utiliser <code>std::string</code> pour simplifier la copie et éviter les erreurs de taille.
Saisie avec <code>cin >></code>	Le consigne interdit <code>getline</code> , donc les villes multi-mots nécessitent <code>_</code> . Avec <code>getline</code> , on pourrait saisir "Le Havre" directement.
DFS sans mémorisation	Ajouter un cache/mémoïsation, pré-filtrer les trajets par ville, ou utiliser Dijkstra/A* pour les chemins optimaux.
Pas de persistance	Sérialiser le catalogue dans un fichier texte ou binaire pour sauvegarder/recharger les trajets.

Annexe : Manuel d'Utilisation

1. Compilation et Exécution

Le fichier `makefile` fourni permet de gérer le cycle de vie du projet.

```
$ make          # Compile et crée l'exécutable 'trajets'
$ ./trajets     # Lance l'application
$ make clean    # Supprime les fichiers objets et l'exécutable
$ make CXXFLAGS="-DMAP" # Compile avec traces de débogage
```

2. Guide du Menu

L'interface utilisateur en ligne de commande propose les options suivantes :

Option	Description
1	Ajout Trajet Simple : Vous serez invité à saisir la ville de départ, d'arrivée et le moyen de transport.
2	Ajout Trajet Composé : Mode interactif. Ajoutez des trajets simples successivement. Le système valide que (Arrivée N = Départ N+1). Tapez '0' pour valider le composé.
3	Afficher : Affiche le contenu actuel du catalogue avec un ID pour chaque trajet.
4	Recherche Simple : Affiche les trajets (simples ou composés) qui relient exactement Ville A à Ville B.
5	Recherche Avancée : (Bonus) Tente de combiner des trajets existants dans le catalogue pour créer un nouvel itinéraire.
0	Quitter : Appelle les destructeurs et libère toute la mémoire.