Compte-rendu du TP6 – C++

# Partie A

## Classe Terminal

La classe Terminal est la classe mère de tous les terminaux de transport existant dans notre programme. Les classes Gare, AeroportRegional, AeroportInternational et HubAeroport héritent de cette classe.

## Classe HubMultimodal

La classe HubMultimodal est une combinaison d’un HubAeroport et d’une Gare. Comme il est risqué de faire un héritage en losange de HubMultimodal depuis Terminal, Nous avons préféré faire hériter HubMultimodal de HubAeroport et de lui rajouter une Gare comme variable.

# Partie B

## Classe Moyen

La classe Moyen est la classe mère des moyens qui existent dans notre représentation. C’est donc à partir de cette classe qu’héritent les classes Avion, Avion Électrique et Train. Moyen contient donc les champs pour la capacité, la vitesse et l’empreinte.

## Classe Ligne

Notre classe Ligne est une classe *template*, comme demandée dans l’énoncé. Elle attend en paramètre des sous-types de Moyen. Une ligne est déterminée par son origine, sa destination et son type de Moyen, (Avion, avion électrique ou train).

Avec cette classe nous avons rencontré quelques difficultés :

* Tout d’abord, comment faire pour stocker dans la classe Voyage des Ligne<T> dans une même liste ? En effet, dans une même classe on ne peut stocker des Ligne<Avion> et des Ligne<Train> dans la même liste car ce sont deux types bien distincts. Pour pallier ce problème nous avons utilisé la solution du *type erasure (*[*https://alp.developpez.com/tutoriels/type-erasure/*](https://alp.developpez.com/tutoriels/type-erasure/)*).*

Cela consiste à faire hériter la classe Ligne d’une classe abstraite que nous avons nommée AbstractLigne. Cette dernière n’implémente que les méthodes que l’on aurait implémentées par défaut dans la classe Ligne. En revanche ces méthodes ne contiennent qu’une ligne : l’appel à des méthodes virtuelles pures de la forme *do\_methode.* C’est donc dans la classe Ligne que nous implémentons les méthodes *do\_metode* avec une visibilité privée. Ceci nous permet donc de stocker une liste d’AbstractLigne dans la classe Voyage, et pouvoir, par l’intermédiaire de cette classe abstraite, utiliser les *templates* de Ligne.

* Aussi, il n’est pas possible de définir les méthode de Ligne dans un .cpp séparé du .h. Nous avons donc défini nos méthodes dans un fichier .tpp. Nous l’incluons à la fin du .h.
* Pour finir, il est demandé dans l’énoncé d’ajouter un champ fréquence. C’est le seul moment où ce champ est évoqué. Nous n’avons alors pas pu comprendre comment utiliser ce champ dans nos calculs, et dans le programme d’une manière générale. De plus, dans le cas où nous avons bien compris ce que veut représenter ce champ, nous nous retrouvons avec des valeurs complétement aberrantes en faisant le lien avec la partie C.

Par exemple, pour un trajet de Lyon à Paris en Avion, avec 18 000 passagers, il nous faut 180 avions en une journée pour emmener tout le monde, car un avion à une capacité de 100. Ceci est très loin de la réalité, et nous a enfoncé dans notre incompréhension.

Vous retrouverez ce champ dans le code, mais il n’est pas utilisé.

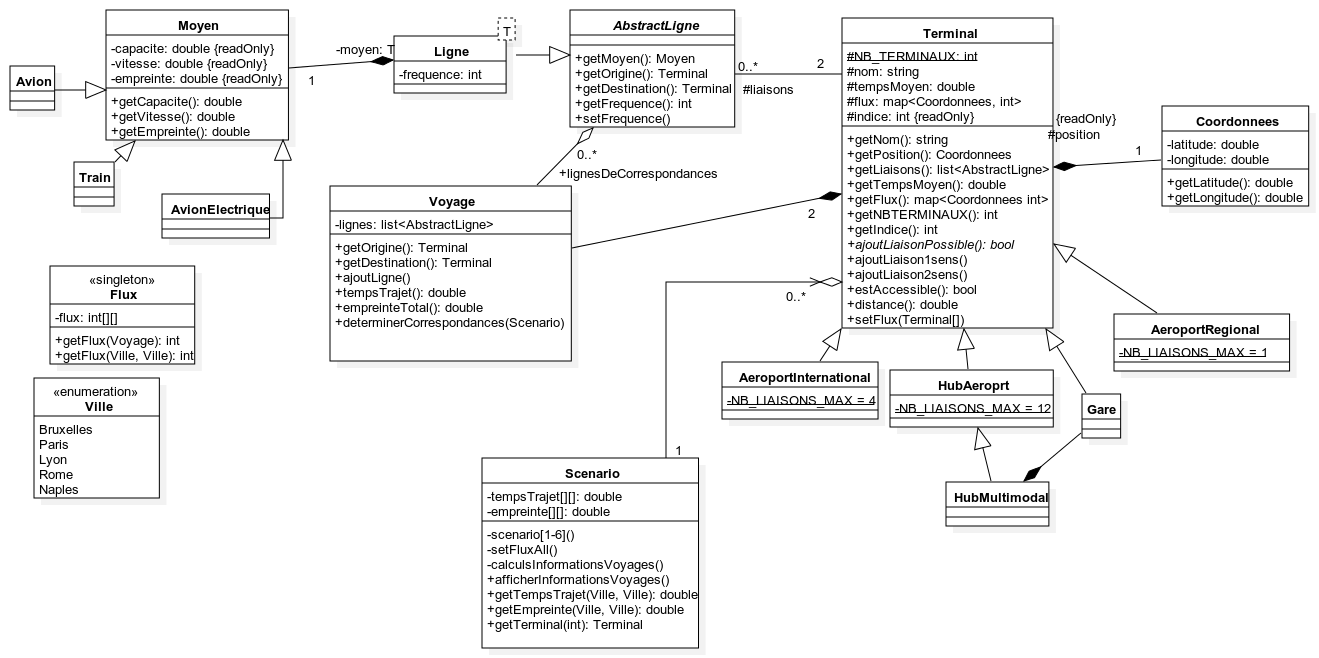
## Classe Voyage

La classe Voyage est celle où tous les calculs sont faits.

Tout d’abord elle est construite grâce à l’origine et la destination du voyage, mais aussi avec tous les terminaux par lequel un voyageur est susceptible de passer. Ceci nous sert à déterminer les lignes successives que devra prendre un voyageur. Ces correspondances sont calculées à l’aide de l’algorithme de Dijkstra, pour établir le plus court chemin entre l’origine et la destination.

Ensuite la méthode temps trajet calcul le temps qu’un voyageur met pour faire ce voyage, et la méthode empreinteTotale nous donne l’empreinte totale du voyage au complet, avec tous les voyageurs.

# Diagramme de classe de la modélisation de l’infrastructure de transport



# Etude des scenarii