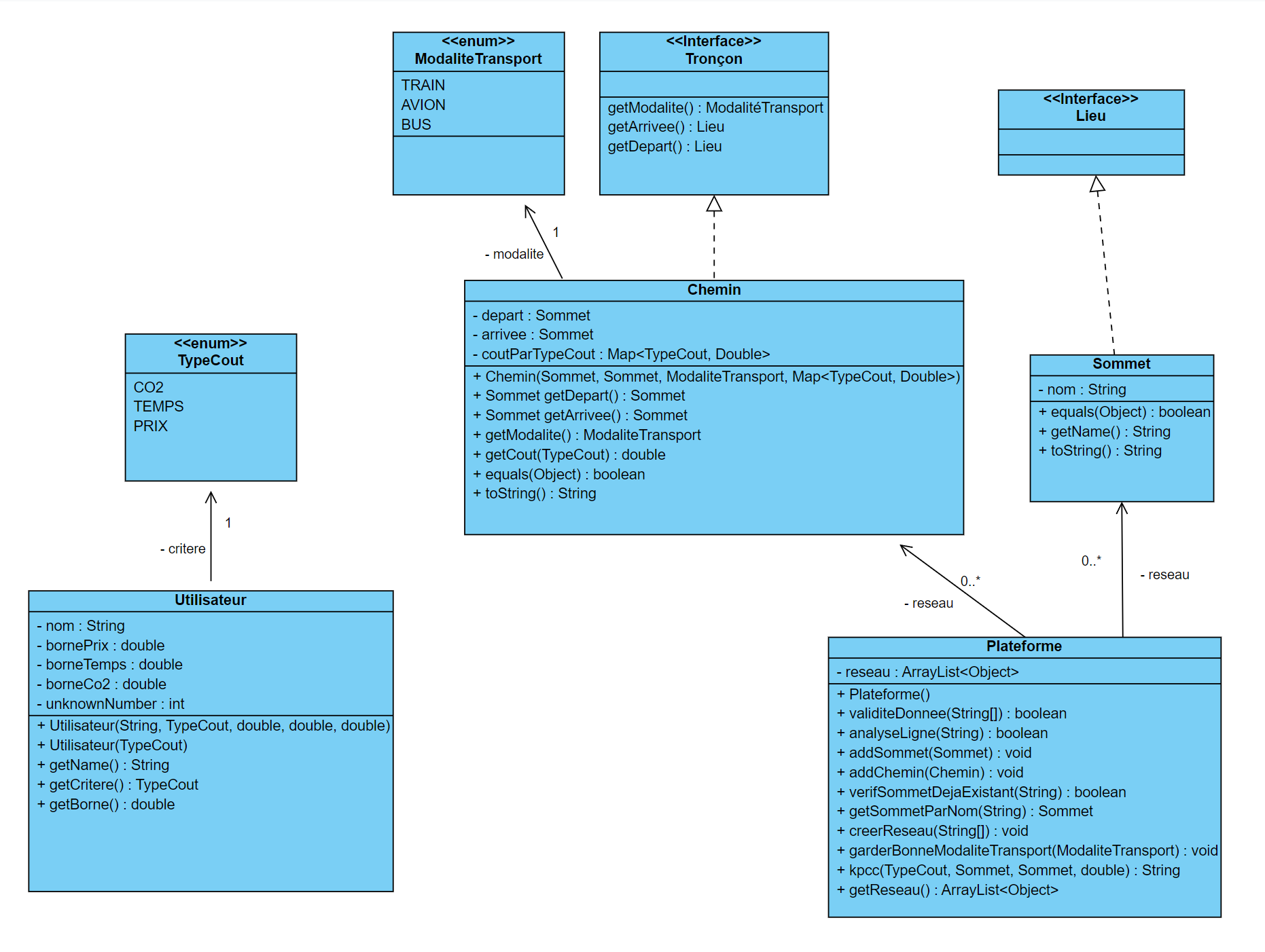
**Rapport SAE 2.01 & 2.02 Version 1**

**RANDOUX Martin, DESPREZ Mathéo, LENCEL Antoine**

**Notre diagramme UML :**

****

**Explication de la structure du diagramme UML**

1. **Énumérations (Enums)** :
   * **ModaliteTransport** : Définit les différents types de transport disponibles (TRAIN, AVION, BUS).
   * **TypeCout** : Regroupe les types de coûts considérés (CO2, TEMPS, PRIX), ce qui permet une gestion uniforme des différents critères de coût.
2. **Interfaces** :
   * **Tronçon** et **Lieu** :
3. **Class** :
   * **Sommet** : Représente un point de départ ou d'arrivée dans un chemin.
   * **Chemin** : Modélise un chemin entre deux sommets avec une modalité de transport spécifique. La **Map coutParTypeCout** permet de stocker les coûts associés à chaque type de coût.
   * **Utilisateur** : Contient les informations sur l'utilisateur, y compris les bornes des différents coûts et les critères de sélection. La classe permet de personnaliser les recherches de trajets en fonction des préférences de l'utilisateur.
   * **Plateforme** : Gère l'ensemble du réseau de sommets et de chemins. Les méthodes telles que **addSommet**, **addChemin**, et **creerReseau** permettent de construire et manipuler le réseau de manière structurée et efficace. Les méthodes de validation et d'analyse assurent l'intégrité des données dans le réseau.

**Explication plus en détail de chaque class :**

**Class Chemin :**

La class **Chemin** implémente l’interface **Trancon** et possède quatre attributs privés :

**depart** et **arrive :** représente les **Sommets** de départ et d’arrivée.

**modalite :** représente la modalité de transport.

**coutParTypeCout :** une **Map** pour stocker les coûts par type (**CO2**, **TEMPS**, **PRIX**).

La class possède un constructeur qui prend les sommets de départ et d’arrivé, la modalité de transport et une Map.

**boolean equals(Object obj)**

Cette méthode vérifie si deux objets **Chemin** sont égaux.

**double getCout(TypeCout typeCout)**

Cette méthode renvoie le coût associé à un type de coût spécifique (CO2, TEMPS, PRIX) à partir de la **Map** **coutParTypeCout**.

**Sommet getArrivee(), Sommet getDepart(), ModaliteTransport getModalite()**

Ces méthodes renvoient respectivement le sommet d'arrivée du chemin, le sommet de départ du chemin et la modalité de transport du chemin.

**String toString()**

Cette méthode renvoie une représentation textuelle du chemin.

**Class Sommet :**

La class **Sommet** implémente l’interface **Lieu** et possède un seul attribut privé :

**name** : représente le nom du **Sommet**.

La class possède un constructeur qui prend le nom du Sommet sous forme d’un String.

**boolean equals(Object obj)**

Cette méthode vérifie si deux objets **Sommet** sont égaux.

**String getName()**

Cette méthode renvoie le nom du sommet

**String toString()**

Cette méthode renvoie une représentation textuelle du sommet.

**Class Utilisateur :**

La class Utilisateur possède six attributs privés :

name : le nom de l’utilisateur.

critere : le critère de coût choisi par l’utilisateur.

bornePrix : la borne maximum pour le prix du trajet.

borneTemps : la borne maximum pour le temps pris par le trajet.

borneCo2 : la borne maximum pour le CO2 produit par le trajet.

unknownNumber : un compteur statique utilisé pour générer des noms d’utilisateur par défaut.

La class possède un constructeur qui prend un nom, un critère, les bornes sur le trajet.

Elle posséde un second constructeur qui prend uniquement le critère et qui lancera le premier constructeur avec des valeurs par défaut pour le reste.

**String getName()**

Cette méthode renvoie le nom de l’utilisateur.

**TypeCout getCritere()**

Cette méthode renvoie le critère de coût choisi par l’utilisateur.

**double getBorne()**

Cette méthode renvoie la borne maximum correspondant au critère de coût.

**Class Plateforme :**

La class Plateforme possède un attribut privé :

reseau : qui est une ArrayList d’objet qui contient des sommets et des chemins

La class possède un constructeur qui initialise l’ArrayList

**ArrayList<Object> getReseau()**

Cette méthode renvoie l’attribut reseau

**boolean validiteDonnee(String[] data)**

Cette méthode vérifie la validité des données en utilisant la méthode **analyseLigne** pour chaque ligne de données. Si une des lignes n’est pas valide, la méthode renvoie false.

**boolean analyseLigne(String ligne)**

Cette méthode vérifie si une ligne contient exactement 5 points-virgules (**;**), ce qui indique que la ligne est au format attendu.

**void addSommet(Sommet s)**

Cette méthode ajoute un sommet au réseau s'il n'est pas déjà présent. Elle vérifie l'existence du sommet en comparant chaque sommet du réseau avec celui à ajouter.

**void addChemin(Chemin c)**

Cette méthode ajoute un chemin au réseau s'il n'est pas déjà présent. Elle vérifie l'existence du chemin en comparant chaque chemin du réseau avec celui à ajouter.

**boolean verifSommetDejaExistant(String name)**

Cette méthode vérifie si un sommet avec un nom donné existe déjà dans le réseau.

**Sommet getSommetParNom(String name)**

Cette méthode renvoie un sommet avec un nom donné s'il existe dans le réseau, sinon elle renvoie **null**.

**void creerReseau(String[] datas)**

Cette méthode crée un réseau à partir d'un tableau de données. Pour chaque ligne de données. Les sommets sont ajoutés au réseau s'ils n'existent pas déjà. Un chemin est créé avec les sommets et les informations de coût, puis ajouté au réseau.

**void garderBonneModaliteTransport(ModaliteTransport modaliteTransport)**

Cette méthode supprime tous les chemins du réseau qui n'ont pas la modalité de transport spécifiée.

**String kpcc(TypeCout typeCout, Sommet depart, Sommet arrive, double limitePoid)**

Cette méthode implémente l'algorithme KPCC pour trouver les chemins optimaux dans un graphe pondéré orienté. Elle :

* Crée un graphe à partir des sommets et chemins du réseau.
* Exécute l'algorithme KPCC pour trouver les chemins entre deux sommets en respectant une limite de poids.
* Retourne une chaîne de caractères décrivant les chemins trouvés qui respectent la limite de poids.