# 1, GTFSAnalyser.java

Ce programme permet de lire et d'afficher des informations basiques sur les agences de transport, les routes, les arrêts et les trajets d'un fichier GTFS.

* **Initialisation du lecteur GTFS** :

Le programme utilise GtfsReader pour lire les fichiers GTFS depuis un répertoire local spécifié (tisseo\_gtfs\_v2).

* **Stockage des entités GTFS :**

Les entités (agences, routes, arrêts, trajets) sont stockées dans un objet GtfsDaoImpl.

* **Chargement des données :**

Les fichiers GTFS sont lus et chargés dans GtfsDaoImpl via la méthode reader.run().

* **Accès aux données :**

Les différentes entités GTFS, telles que les agences, les routes, les arrêts et les trajets, sont récupérées depuis GtfsDaoImpl.

* **Affichage des données :**

Le programme affiche les informations suivantes dans la console :

* + Les agences (ID et nom)
  + Les routes (ID et nom)
  + Les arrêts (ID et nom)
  + Les trajets (ID et ID de la route associée)
* **Gestion des exceptions :**

En cas d'erreur lors de la lecture des fichiers, une exception est capturée et affichée via e.printStackTrace().

# 2.GTFSAnalyser\_vers1.java

* **Lecture des fichiers GTFS** :

Le programme utilise GtfsReader pour lire des fichiers GTFS depuis un répertoire spécifié (hanoi\_gtfs\_pm) et stocke les entités dans GtfsDaoImpl.

* **Affichage des entités GTFS :**

Le programme affiche les informations des agences, des routes, des arrêts et des trajets à partir des données GTFS chargées.

* **Affichage des trajets et arrêts** :

Pour chaque trajet, les arrêts correspondants sont récupérés via getStopTimesForTrip(). Le programme affiche les noms des arrêts ainsi que les heures d'arrivée et de départ, formatées via formatTime().

* **Calcul de la durée des trajets** :

La durée de chaque trajet est calculée à partir des heures d'arrivée et de départ du premier et dernier arrêt du trajet.

* **Génération d'un graphique** :

Un graphique à barres est généré pour afficher le nombre de trajets par route. Ce graphique est créé avec la bibliothèque JFreeChart et affiché dans une fenêtre Swing (JFrame).

* **Fonctions utilitaires** :
* formatTime() : Convertit le temps en secondes en format hh:mm:ss.
* getStopTimesForTrip() : Filtre les arrêts pour un trajet spécifique à partir de la collection de StopTime.

# 3. Coordonnees\_arret.java

Ce code permet d'explorer les données GTFS pour obtenir des informations détaillées sur les trajets et les arrêts, ainsi que les horaires et coordonnées géographiques des arrêts.

* **Lecture des fichiers GTFS** :
* Le programme utilise GtfsReader pour lire les fichiers GTFS depuis un répertoire spécifié (tisseo\_gtfs\_v2).
* Les entités lues sont stockées dans GtfsDaoImpl.
* **Affichage des trajets avec leurs arrêts et horaires** :
* Pour chaque trajet (Trip), le programme parcourt les arrêts (Stop) associés en récupérant les temps d'arrivée et de départ (StopTime).
* Pour chaque arrêt, le nom, la latitude, la longitude, ainsi que les horaires d'arrivée et de départ (en format HH:mm) sont affichés.
* **Méthode utilitaire pour le formatage du temps** :
* La méthode convertSecondsToHHMMSS convertit le temps en secondes en un format lisible (heures, minutes, secondes).
* **Gestion des erreurs** :
* Le programme vérifie si le répertoire GTFS existe et s'il est valide avant de procéder au traitement. Si une erreur se produit, elle est affichée dans la console.

# 4. Liste\_Arret.java

Ce programme permet de visualiser les informations géographiques entre deux arrêts d'un trajet dans les données GTFS, en affichant les points de tracé ainsi que les arrêts associés.

* **Chargement des données GTFS** :
* Le programme utilise GtfsReader pour lire les fichiers GTFS depuis un répertoire spécifié.
* Les données lues sont stockées dans un objet GtfsDaoImpl.
* **Affichage des points de tracé entre deux arrêts** :
* Le programme prend en entrée un ID de trajet (tripId), ainsi que les noms des arrêts de départ et d'arrivée.
* Il récupère les StopTimes correspondant à ces arrêts pour ce trajet et filtre les points de tracé (shape points) associés entre ces deux arrêts.
* **Affichage des informations de tracé** :
* Pour chaque point de tracé, le programme affiche la latitude, la longitude, la séquence, et éventuellement le nom de l'arrêt correspondant s'il existe.
* Si aucun arrêt ne correspond à un point de tracé spécifique, seule la latitude et la longitude sont affichées.
* **Gestion des erreurs** :
* Si le répertoire GTFS spécifié est invalide ou si les arrêts ou le trajet ne sont pas trouvés, des messages d'erreur appropriés sont affichés.

# 5. LesPointsEntreArret.java

Ce programme est utile pour visualiser les informations géographiques entre deux arrêts sur un trajet donné, en affichant les points de tracé correspondants.

* **Chargement des données GTFS :**
* Le programme utilise GtfsReader pour lire les fichiers GTFS d'un répertoire et stocke les données dans un objet GtfsDaoImpl.
* **Sélection d'un trajet :**
* Un tripId est défini pour identifier le trajet à analyser (dans cet exemple, "1965630").
* **Récupération du Shape ID :**
* Le programme trouve le trajet correspondant à cet tripId et récupère l'identifiant de la forme géographique associée à ce trajet (Shape ID).
* **Récupération des points de tracé :**
* Les points de tracé (ShapePoints) sont ensuite filtrés et triés selon leur séquence.
* **Affichage des points de tracé :**
* Le programme affiche la latitude, la longitude et la séquence des points de tracé associés au trajet.

# 6. TrajetEntreArrets.java

Ce code en Java permet de trouver un trajet entre deux arrêts spécifiques dans un réseau de transport en commun en utilisant des données GTFS (General Transit Feed Specification). Voici un résumé de ses principales fonctionnalités :

* **Chargement des données GTFS** : Le programme charge les données GTFS d'un répertoire local spécifié contenant des fichiers GTFS, comme des informations sur les arrêts, les horaires, les trajets, etc.
* **Recherche de trajets entre deux arrêts** :
  + L'utilisateur spécifie deux arrêts avec leurs identifiants (stopA et stopB).
  + Le programme recherche tous les trajets (Trip) qui passent par ces deux arrêts en se basant sur les séquences des arrêts dans chaque trajet.
  + Pour chaque trajet passant par les deux arrêts, il vérifie si l'arrêt A apparaît avant l'arrêt B dans la séquence du trajet.
* **Affichage des résultats** : Si un trajet valide est trouvé (c'est-à-dire que l'arrêt A précède l'arrêt B dans la séquence du trajet), le programme affiche l'ID du trajet correspondant. Si aucun trajet n'est trouvé, un message indiquant l'absence de trajet entre les arrêts est affiché.
* **Fonctionnalités clés** :
* **Chargement des fichiers GTFS** avec la bibliothèque onebusaway-gtfs.
* **Recherche des trajets communs** entre deux arrêts donnés.
* **Validation de la séquence des arrêts** pour s'assurer que le premier arrêt apparaît avant le second dans un trajet.

Cela permet d'analyser les données de transport en commun pour identifier des connexions valides entre des points de départ et d'arrivée.

7. Horaire\_Arret.java

Ce programme Java permet d’afficher les horaires d’arrivée et de départ pour un arrêt de transport en commun spécifique, en utilisant des données GTFS. Voici un résumé de ses principales fonctionnalités :

Fonctionnalités principales :

* **Chargement des données GTFS** : Le programme charge les fichiers GTFS depuis un répertoire local pour extraire les informations relatives aux arrêts et aux horaires. Il utilise la bibliothèque onebusaway-gtfs pour lire et stocker les données.
* **Filtrage des horaires pour un arrêt spécifique** : On va utiliser spécifie un stop\_id (par exemple, "stop\_point"), qui correspond à un arrêt particulier. Le programme recherche tous les horaires de passage (arrivées et départs) associés à cet arrêt.
* **Affichage des horaires** :
  + Le programme filtre les StopTimes correspondant à l’arrêt donné.
  + Il trie ces horaires par ordre d’arrivée et élimine les doublons pour chaque combinaison de départ/arrivée.
  + Il convertit les horaires (exprimés en secondes depuis minuit) au format HH:MM:SS pour un affichage lisible.
* **Validation des arrêts** : Le programme vérifie si l'arrêt récupéré est bien un objet valide de type Stop. Si ce n’est pas le cas, un message d’erreur est affiché.
* **Gestion des erreurs** : Si aucun horaire n’est trouvé pour l’arrêt spécifié, ou si le répertoire des données GTFS est invalide, des messages d’erreur appropriés sont affichés.
* **Fonctionnalités techniques :**
* **Lecture et parsing des fichiers GTFS** : Le code utilise la classe GtfsReader pour lire les données GTFS depuis un répertoire local.
* **Manipulation des horaires** : Les horaires d’arrivée et de départ des arrêts sont traités et convertis à partir de secondes en heures, minutes et secondes.
* **Tri et affichage des données** : Le programme trie les horaires pour afficher les heures d'arrivée et de départ dans l'ordre chronologique.