

Rapport POO

SAE

Dylan Lecocq
Amaury Vanhoutte
18/05/2024

Sommaire

Sommaire	2
I - Version 1	3
A. Structuration du projet	3
B. Lancement de l'application	3
C. Diagramme UML	4
D. Implémentation des fonctionnalités	5
E. Tests réalisés	6
II - Version 2	7
A - Structuration du projet	7
B - Lancement de l'application	7
C - Diagramme UML	7
D - Implémentation des fonctionnalités	8
E - Tests réalisés	9

I - Version 1

A. Structuration du projet

Le projet est structuré sous la forme d'une arborescence des fichiers, comme suit :

Name	Date Modified	Size	Kind
> bin	Yesterday at 20:57	--	Folder
> doc	13 May 2024 at 16:24	--	Folder
> graphes	Yesterday at 20:56	--	Folder
> lib	14 May 2024 at 17:03	--	Folder
> src	14 May 2024 at 17:02	--	Folder
> tests	Yesterday at 20:56	--	Folder

Les librairies nécessaires au bon fonctionnement de l'application se trouvent dans le repertoire lib/ ,les ressources de l'application dans le repertoire src/ tandis que les classes de tests se trouvent dans le repertoire tests. Le repertoire graphes contient le rendu pour la partie graphe du projet. Les repertoires doc/ et bin/ servent respectivement à stocker la documentation des différentes classes du projet, ainsi que les fichiers compilés de ces même classes.

B. Lancement de l'application

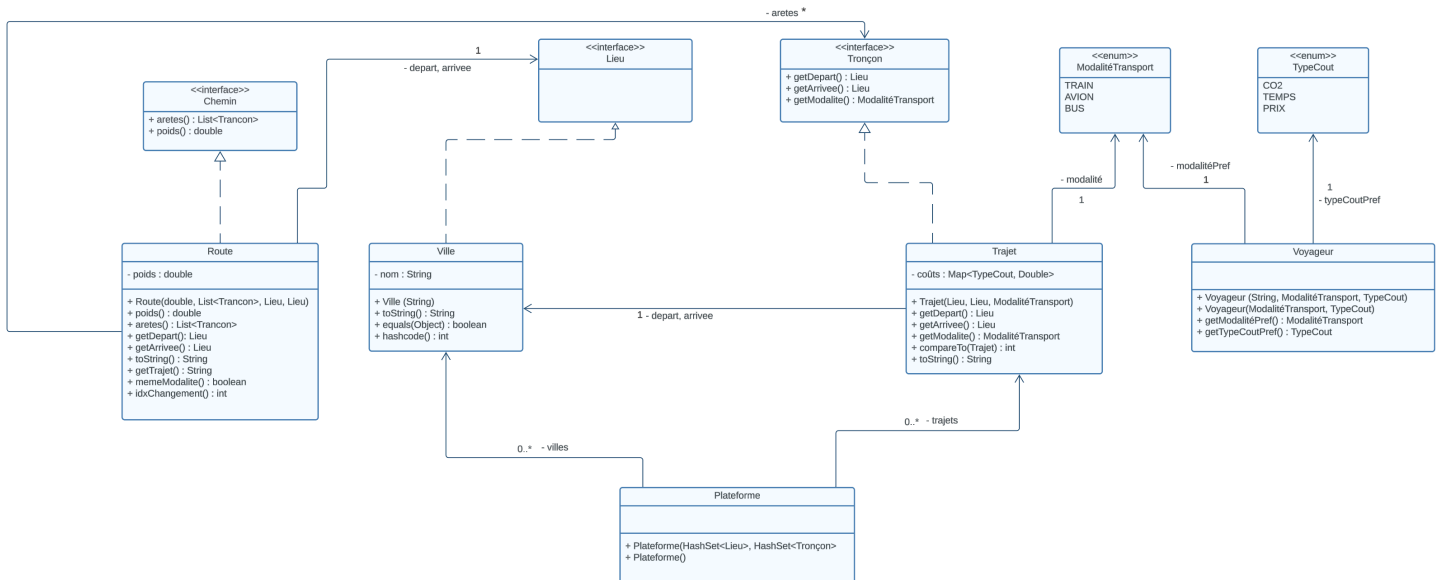
Comme indiqué précédemment, l'application se trouve dans le repertoire src/. Une fois placé à la racine du projet, pour pouvoir démarrer l'application il faut, dans un premier temps compiler l'ensemble des classes, ce qui peut être fait avec la commande suivante :

```
javac -cp ".:lib/*" -d bin src/*.java
```

Puis pour démarrer l'application on utilise : **java -cp "bin:lib/*" App**

C. Diagramme UML

Voici le diagramme UML représentant l'ensemble des classes du projets :



Dans notre implémentation, on retrouve deux principes de POO, l'implémentation, qui nous permet de réaliser les diverses interfaces présentes dans les bibliothèques fournies, qui définit des comportements pour les classes les implémentent.

Puis on retrouve de l'encapsulation, en effet plusieurs classes contiennent des attributs du type d'autres classes qui sont en visibilité `private`, ce qui les rends inaccessibles par l'utilisateur, excepté en utilisant les méthodes fournies.

D. Implémentation des fonctionnalités

Pour cette version 1, les fonctionnalités demandées sont les suivantes :

- Vérification de la validité des données fournies
- Récupérer les informations pour reconstituer le réseau ;
- Filtrer les données sur un moyen de transport spécifique ;
- Déterminer si, selon la modalité choisie, il existe un voyage possible entre les villes sélectionnées par l'utilisateur ;
- afficher les meilleurs voyages possibles, ordonnés selon le critère le plus important pour l'utilisateur ;
- exclure des alternatives qui excèdent les bornes définies par l'utilisateur.

La vérification des données se fait à l'aide d'une méthode `verifyData()`, qui vérifie si l'ensemble des champs attendus sont présents et s'ils sont au bon format grâce à une autre méthode `isNumeric()` qui vérifie si une chaîne de caractère passé en paramètre est numérique.

La récupération des données se fait après la vérification de celles-ci, grâce à une méthode `retrieveData()` qui prends en paramètre le tableau des données ainsi que la plateforme à laquelle on veut ajouter ces données.

Le filtre selon un moyen de transport se fait quant à lui au moment de l'ajout des données au graphe, grâce à une méthode `ajouterVillesEtTrajets()` qui prends en paramètre le graphe auquel on souhaite ajouter les données, la plateforme les contenant, le type de coût et le moyen de transport souhaités afin de n'obtenir un graphe n'ayant que les arêtes respectant les modalités définies par l'utilisateur.

L'affichage des meilleurs voyages est rendu possible grâce à une méthode dédiée `afficherPCC()` qui prends en paramètre une liste de chemins, le type de coût préféré de l'utilisateur.

E. Tests réalisés

Nous avons réalisé une classe de test destinée à vérifier le bon fonctionnement des méthodes principales de l'application, et ainsi améliorer la robustesse de l'application en général. Elle permet actuellement de tester 6 méthodes différentes. Chacune d'entre elle teste différents cas, d'exception ou non, et contrôle le résultat renvoyé par la fonction.

II - Version 2

A - Structuration du projet

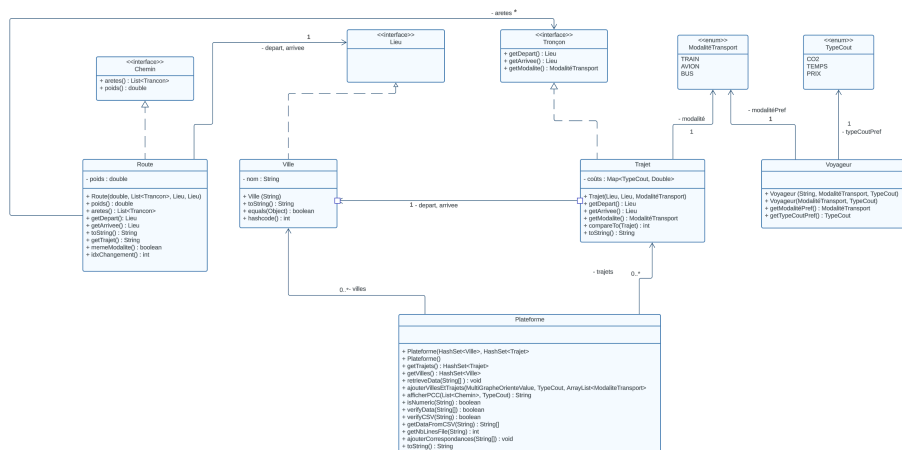
La structuration du projet est identique à la version 1.

B - Lancement de l'application

Les commandes pour compiler et exécuter cette version sont identiques à celles de la version 1. C'est à dire `javac -cp ".:lib/*" -d bin src/*.java` pour la compilation et `java -cp "bin:lib/*" App` pour l'exécution.

C - Diagramme UML

Pour la version 1, certaines méthodes se trouvaient dans la classe principale du projet et n'apparaissaient donc pas sur le diagramme UML, pour la version 2, ces méthodes ont toutes été déplacées dans la classe Plateforme, ce qui explique les méthodes ajoutées sur le diagramme UML.



D - Implémentation des fonctionnalités

Pour cette version 2, les fonctionnalités demandées sont les suivantes :

- Fonctionnalités de la version 1
- Import des données depuis des fichiers CSV
- Signaler les problèmes via un mécanisme d'exception
- Filtrer l'affichage d'une solution, pour n'afficher que les points d'intérêt

La plupart des méthodes s'occupant des fonctionnalités de la version 1 n'ayant pas été modifiées, celles-ci garde le même comportement dans cette version 2. Cependant, certaines ont dû être adaptée afin de prendre en charge le mécanisme d'exception et le nouvel affichage, c'est le cas des méthodes suivantes :

- `afficherPCC(List<Chemin>, TypeCout)` qui lance une exception de type `NoTripException` si aucun voyage respectant les critères n'a été trouvé. Et qui n'affiche plus que les points d'intérêts des Trajets.
- `verifyData(String[])` qui lance une exception de type `InvalidStructureException` si les fichiers CSV ne respectent pas la structure requise.
- `toString()`, la méthode `toString()` de la classe `Route` a été modifiée afin de prendre en charge le nouvel affichage requis.

De nouvelles méthodes sont disponibles dans cette version 2 :

- `verifyCSV(String)` et `getDataFromCSV(String)` qui permettent respectivement de vérifier l'existence et la structure du fichier passé en paramètre puis de récupérer les données de celui-ci si il est valide. Elles lancent une exception de type `FileNotFoundException` si le chemin passé en paramètre ne correspond à aucun fichier existant, et une exception de type `InvalidStructureException` si la structure de celui-ci n'est pas conforme.
- `getNbLinesFile(String)` qui permet de récupérer le nombre de lignes dans un fichier, elle lance une exception de type `FileNotFoundException` si le fichier n'existe pas.

E - Tests réalisés

En plus des tests réalisés pour la version 1, nous avons ajouté au sein de la classe de test, de tests pour chacune des 3 nouvelles méthodes ajoutées pour cette version, qui sont `verifyCSV(String)`, `getDataFromCSV(String)` et `getNbLinesFile(String)`. Chacun des nouveaux tests vérifie, comme celles de la version 1, des cas d'exceptions et des cas ordinaires afin de s'assurer que le programme continue de s'exécuter même s'il rencontre un problème