SAÉ 2.04 BDD | Rapport

Table of Contents

Projet réalisé part FANTUZ CHROBOT Léo, LYOEN Quentin et GODDEFROY Arthur, groupe A.

Membres du groupe

```
SAE S2.01/S2.02 -- Rapport POO
===

Noms des membres, groupe:

Arthur Goddefroy
Leo Fantuz Chrobot
Quentin Lyoen

Groupe: A
```

1. Version 1

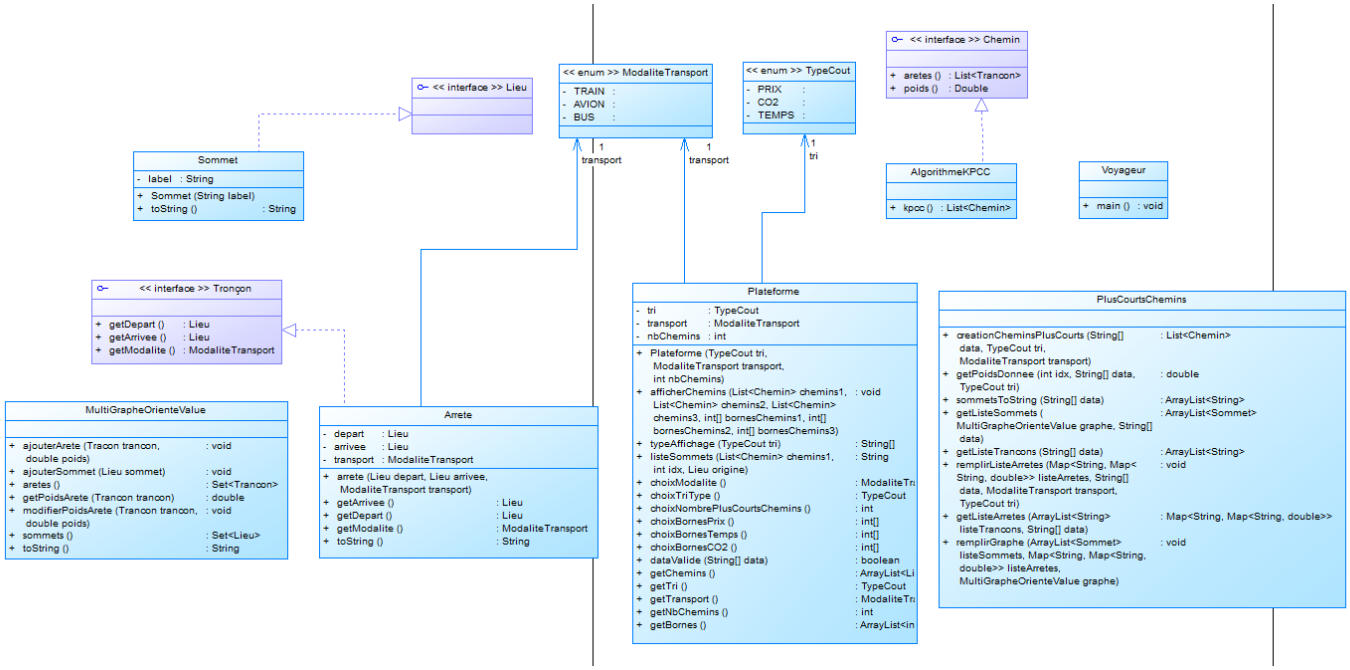
1.1. Lancement de l'application

Afin de lancer et d'utiliser l'application avec commandes, il est bon de mentionner que l'utilisation des librairies situées dans le dossier "lib" de la racine du projet est nécessaire. Ainsi, afin de lancer l'application, ouvrez un terminal et dirigez vous à la racine du projet. Ensuite, exécutez la commande suivant pour compiler: `javac -d bin -cp bin:lib/* src/Version1/*.java`. Puis, pour lancer l'application, utilisez la commande suivante: `java -cp bin:lib/ Version1.Voyageur` suivit des arguments représentant les différentes arêtes du graphe souhaitées sous la forme

"villeDépart;villeArrivée;modalitéTransport;prix(e);pollution(kgCO2e);durée(min)" (ex: `java -cp bin:lib/* Version1.Voyageur "A;B;Train;60;1.7;80" "B;A;Train;60;1.7;80"`). Si aucun argument n'est fourni lors de l'exécution de l'application, des données par défaut sont fournies et utilisées.

1.2. Schématisation du problème

Voici l'UML utilisé afin de schématiser le problème pour cette première version:



Afin de réaliser ce logiciel, nous nous sommes penchés sur l'aspect efficace et compact de notre UML en nous demandant quels mécanismes objets pourraient être le plus adapté à notre problème. Ainsi, nous avons commencé par créer les classes Sommets et Arrete afin de pouvoir implémenter les méthodes des interfaces Lieu et Trancon.

Ensuite, dans la fameuse classe Arrete, nous avons utilisé une implémentation de l'enum ModaliteTransport. Ainsi, la classe Arrete contient comme attribut une ModaliteTransport appelée "transport". Il est également bon de spécifier que la classe Arrete contient également l'implémentation de l'interface Lieu par le biais de 2 attributs ("depart" et "arrivee"), bien que cela ne soit pas directement des Lieu (car Lieu étant une interface, elle ne peut pas être instanciée) mais plutôt des classes implémentant cette dernière telle que Sommet, dans notre cas.

Par la suite, nous avons eu besoin de créer une classe nommée PlusCourtsChemins afin de développer les méthodes permettant les calculs des plus courts chemins possibles pour tout les types de coûts différents, à savoir ici le prix, le temps et le taux de kg de co2 (une enum a ainsi été créer afin de permettre le rassemblement de ces différents types de coûts de manière précis et logique). Bien que la classe PlusCourtsChemins n'est pas directement liée à d'autres classes de notre application, car ne disposant d'aucun attribut, il est bon de spécifier que cette dernière instancie pas mal des autres classes dans ses propres méthodes afin de réaliser ces dernières.

Après tout cela, nous avons créé la classe Plateforme regroupant toutes les méthodes nécessaires à l'utilisation et l'affichage de ces fameux plus courts chemins. Comme pour la classe précédente, pas mal de classes de notre application son instancié dans les différentes méthodes de notre classe Plateforme. Cependant, cette classe dispose également de plusieurs associations.

En effet, cette classe dispose de 3 attributs: tri (étant un objet TypeCout), transport (étant un objet ModaliteTransport) et nbChemins (étant un int). Ainsi, une implémentation des classes TypeCout (crée au préalable) et ModaliteTransport (ainsi disponible dans les librairies fournies à l'origine) a été nécessaire, ainsi qu'un constructeur permettant d'initialiser les valeurs souhaitées.

Pour finir, nous avons créé la classe Voyageur permettant d'exécuter le programme à l'aide des classes PlusCourtsChemins et Plateforme. Ainsi, il s'agit là de notre classe mère devant être exécuté lorsque l'on souhaite lancer l'application. Comme pour les autres classes, elle fait parfois appel à plusieurs instanciations de classes, mais aucune ne fait l'objet d'un lien, car cette classe ne contient qu'une méthode main, donc aucun attribut.

1.3. Analyse technique et critique de l'implémentation des fonctionnalités demandées

Lors de l'élaboration du projet, nous avons réussi à respecter toutes les fonctionnalités exigées. Nous avons un utilisateur muni d'un nom initialisé dans la classe Voyageur grâce à l'utilisation de la classe "Keyboard". L'initialisation du type de tri souhaité (coût, temps ou co2) et des bornes pour chacun des poids pour chaque tri sont quant à elles disposées dans la classe Plateforme sous forme de méthodes static. Une vérification des données fournies est également développée dans la classe Plateforme, vérifiant ainsi si les données fournies contiennent le bon nombre d'éléments ainsi que les valeurs nécessaire (pas de int au lieu d'un String et inversement).

Comme cité dans la partie précédente, l'enum TypeCout a été créé afin d'énuméré les différents type de tri disponible (prix, temps et co2).

Concernant le calcul des chemins les plus courts, nous avons créé dans nos classes Plateforme et PlusCourtsChemins plusieurs méthodes permettant de récupérer les sommets des données fournies ainsi que la récupération et le remplissage (dans le graphe) des différents trançons/arêtes (par le biais des méthodes "sommetsToString", "GetListesSommets", "GetListeTrancons", "getListeArretes" et "remplirListeArretes"), tout cela sous la forme d'ArrayList concernant les getters.

Il est bon de spécifier que les détails techniques ont bels et bien été respectés puisqu'évidemment, les arêtes ont bel et bien été réunies dans une Map schématisée de la manière suivante: HashMap>. Tout cela se créait et remplir dans les méthodes "getListeArretes" et "remplirListeArretes" citées précédemment. Un parcours de cette Map est ainsi fait par le biais d'une double boucle ForEach afin de créer les arêtes dans le graphe pour chaque trançons (avant de créer chaque trançons dans le MultiGrapheOrienteValue, un tri est effectué pour ne garder que la Modalité de transport souhaitée par l'utilisateur afin d'éviter des tours de boucles inutiles).

Pour finir, la création des chemins les plus courts pour une modalité de transport choisi se fait dans la méthode "creationCheminsPlusCourts" en spécifiant les données utilisées, le type de tri utilisé (prix, temps ou co2) ainsi que la modalité de transport souhaité. Cela nous retourne ainsi une liste des plus courts chemins disponibles.

L'affichage quant à lui se fait dans la méthode "afficherChemins" de la classe Plateforme en y spécifiant les 3 listes de chemins dans l'ordre en fonction du type de tri souhaité ainsi que les bornes. Ainsi, pour k nombre de plus courts chemins souhaités, une comparaison des différentes listes de chemins par type de tri est fait (en comparant à chaque fois leurs arêtes) afin de permettre de renvoyer tous les coûts liés à un

trajet, triés par le type de tri souhaité.

Un tri est également fait avec les bornes minimum et maximum de chaque coûts permettant ainsi de n'afficher que les chemins respectant ces bornes.

1.4. Analyse quantitative/qualitative des tests

Concernant les tests, toutes les méthodes testables ont été testées par le biais de méthodes de tests disposées dans des classes de tests dans le dossier test à la racine du projet. Nous avons commencé par faire, pour chacune des classes de tests, une méthode "@BeforeEach" permettant d'instancier tout ce qui est nécessaire à l'utilisation des méthodes de test et qui sera ainsi appelée avant chacune d'entre elles.

Ensuite, par le biais de méthode de test initialisable par le mot-clé "@Test", nous avons élaboré des tests couvrants toute possibilités d'erreur par le biais de différents scénarios, avec des valeurs différentes pour chacune des données initialisées (au moins 3 scénarios différents). Il est possible que certains cas n'aient pas été testés, mais pour sûr, presque tous les cas possibles on été tester.

Cependant, il est vrai que certaines méthodes ne permettent pas de bien tester nos méthodes. La méthode "testRemplirGraphe" par exemple est difficile à tester, car la création et l'affichage du graphe ne se fait pas correctement avec la librairie fournie au début du projet.

Hormis cela, une autre méthode ne disposant pas d'une très grande variété de cas testés est la méthode "testCreationCheminsPlusCourts" dans la classe de test PlusCourtsCheminsTest. En effet, cette fonction ne dispose que d'un test, car peu de cas potentiels d'erreur peuvent être testés ici.

Cependant, nous soulignons que cette méthode sera améliorée dans les versions futures afin d'y insérer les nouvelles fonctionnalités de création de graphe. Ainsi, plus de 90% de notre code pouvant être couvert l'a été afin de permettre une vérification nette et précise de la fonctionnalité correcte de nos méthodes.