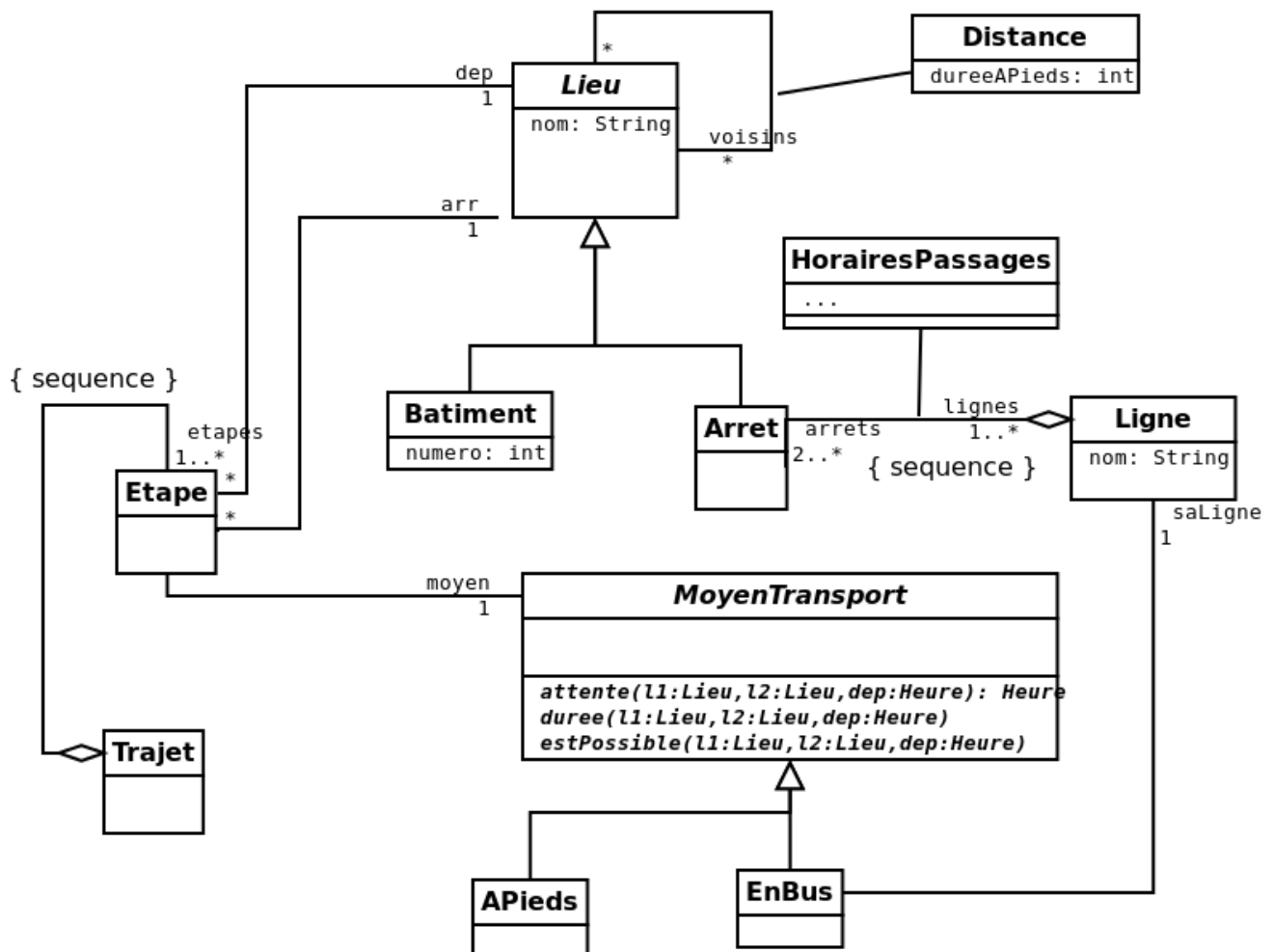


« Concepts Objets » - TP

On considère des déplacements sur le campus d'Orsay (avec différents types de lieux et de moyens de transports). Le modèle UML correspondant (et vu en TD) est donné ci-dessous.



Ce modèle de classes est en partie implémenté dans la hiérarchie de squelettes de classes Java fournie pour ce TP. Dans la hiérarchie fournie, la « classe association » correspondant à la relation de voisinage de **Lieu** est implémentée via une méthode `distance(Lieu l)` qui renvoie la distance à pieds entre deux lieux voisins (pour simplifier le programme de test fourni, la fonction suppose que le temps de parcours entre deux voisins ne dépend pas du sens de parcours). De même, la « classe association » qui modélise les horaires de passage des bus est implémentée par deux tableaux associés à chaque instance d'une ligne de bus qui fournissent la liste des horaires de départ depuis la tête de ligne et les durées respectives entre deux arrêts (on suppose que la durée entre deux arrêts est constante dans la journée). Le modèle de classes comporte en plus deux associations (non représentées sur le diagramme) entre les classes **Trajet** et **Lieu** qui correspondent aux extrémités globales d'un trajet (en plus des extrémités de chaque étape de ce trajet).

Le squelette de chacune des méthodes à écrire est donné, mais son corps lève systématiquement l'exception `UnsupportedOperationException`. Un programme de test est aussi fourni. Le tout est immédiatement compilable. **Vous ne devez pas modifier la signature des méthodes fournies pour que le programme de test puisse s'exécuter correctement.**

Dans toutes les questions ci-dessous on évitera autant que possible de tester les classes des objets, soit directement via `instanceof`, soit indirectement par des « `cast` ». On utilisera aussi systématiquement les versions génériques des classes de la bibliothèque Java.

1. En partant du squelette des classes fournies dans le sous-répertoire `lieux`, complétez l'ensemble des méthodes demandées afin de pouvoir exécuter le programme de test de la classe `TestLieux.java`; Ajoutez toute méthode qui serait nécessaire pour un bon fonctionnement. Pour cette première question, les méthodes `nbChgt` et `meilleur` de `Trajet` sont inutiles, de même que l'interface `Compareteur` et ses implémentations.
2. On s'intéresse maintenant à fournir différents comparateurs de trajets afin de pouvoir fournir le meilleur trajet d'une collection donnée, suivant un critère passé en paramètre. Écrire trois classes `CompTemps`, `CompAttente` et `CompChgt` qui implémentent l'interface `Compareteur` et qui fournissent respectivement un comparateur qui minimise la durée totale d'un trajet, un comparateur qui minimise le temps d'attente, et un comparateur qui minimise le nombre de changements de moyens de transports (changer de ligne de bus revient à changer de mode de transports). En plus de ces trois classes, vous aurez à écrire les méthodes `nbChgt` et `meilleur` de la classe `Trajet` ainsi que différentes méthodes `compare` et `equals`. Votre implémentation doit permettre de dérouler de façon satisfaisante le programme de test de la classe `TestLieuxSuite.java`.
3. Copiez votre hiérarchie de classes dans le nouveau répertoire `lieux2` et modifiez-la en vous arrangeant pour qu'il n'y ait toujours qu'**une seule instance d'un moyen de transport (ou une seule instance par ligne pour le déplacement en bus)** en adaptant le principe des singletons. Comment gérez-vous les lignes de bus? Les programmes de test des classes `TestLieuxQ3.java` et `TestLieuxSuiteQ3.java` supposent l'existence d'une méthode `getInstance` adaptée à chaque moyen de transport. **Simplifiez en conséquence les classes qui peuvent l'être.**