DM Covoiturage

Sur le parking de l'IUT, rien ne va plus. Pour améliorer la situation, l'équipe enseignante a décidé de vous faire travailler sur *le covoiturage*.

Le problème dans toute sa généralité (transporter des gens de manière à remplir le parking au minimum) est assez compliqué. Pour commencer, on va faire *pas mal de simplifications* :

- on suppose que les voitures ne sont pas nominatives (i.e. si A. détient une voiture et habite à Sandillon, on dira qu'il y a une voiture disponible à Sandillon).
- tout le monde commence à 8h00 et termine à 18h00 (pour ne pas avoir à gérer des différences d'emploi du temps).

Dans la suite, on vous demande d'implémenter des classes en vous donnant attributs et méthodes.

D'autres méthodes seront probablement nécessaires (pour segmenter le code, ou encore avoir des méthodes d'affichage pour le debuggage, etc.); n'hésitez pas à les rajouter !

Manière de travailler

Pour ce DM, nous avons écrit des tests vous permettant de tester vos réponses. Pour commencer, récupérez le fichier BatterieTestsDevoirMaison.class et copiez le dans le dossier de votre travail, puis exécutez le de la manière suivante :

```
java -cp .:/usr/share/java/junit.jar BatterieTestsDevoirMaison
```

Vous devez obtenir une sortie ressemblant à ceci :

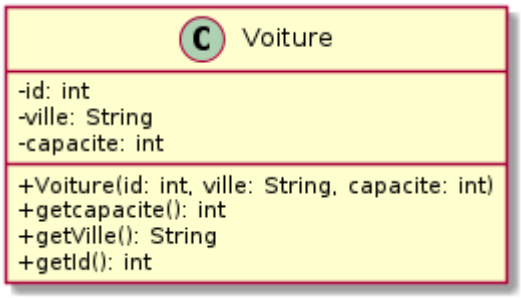
```
Nombre de tests effectués : 2
Nombre de tests réussis: 0
Sur certains tests, votre code a levé des exceptions :
Dans le test : testPersonne(BatterieTestsDevoirMaison), l'erreur suivante a été
rencontrée:
  java.lang.NoClassDefFoundError: Personne
Dans le test : testVoiture(BatterieTestsDevoirMaison), l'erreur suivante a été
rencontrée:
  java.lang.NoClassDefFoundError: Voiture
....
```

Sur cet exemple, deux tests ont été réalisés (testPersonne et testVoiture), et chacun a échoué car la classe correspondante (Personne et Voiture) n'est pas encore définie.

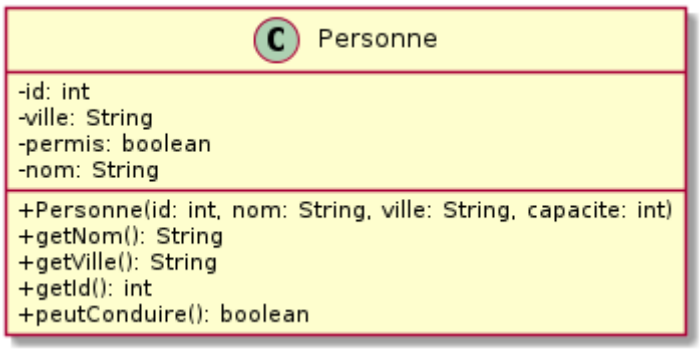
Par ailleurs, vous devez faire vos propres tests avec une classe Executable.

Objets

1. Écrivez une classe Voiture correspondant au diagramme suivant :

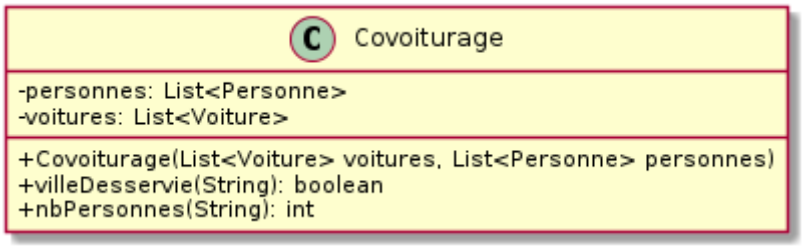


2. Compilez votre code puis relancez les tests. Vous devez maintenant avoir un nombre de tests passés égal à 1.
3. De la même façon créez une classe Personne :



   - un attribut *id* correspondant à un identifiant unique (entier),
   - un attribut *nom* correspondant au nom de la personne
   - un attribut *ville* correspondant à la ville de résidence de la personne,
   - un attribut *permis* valant vrai si la personne a le permis et faux sinon,
   - un constructeur prenant en argument un identifiant, un nom, une ville et un booléen à vrai si la personne a le permis et faux sinon,
   - une méthode `peutConduire` renvoyant le booléen vrai si la personne a le droit de conduire et faux sinon (i.e. a le permis),

4. Créez une classe Covoiturage :



Covoiturage - premiers algorithmes

1. Ajoutez à la classe Covoiturage :
   - une méthode capaciteSuffisante prenant en entrée une ville et renvoyant vrai si il y a assez de place dans les voitures de la ville pour toutes les personnes y habitant et faux sinon,
   - une méthode villeEstDans prenant en entrée un tableau de villes et un ville et renvoyant vrai si la ville appartient au tableau et faux sinon (on est ici en train de reproduire le *in* de python, ce sera utile pour la suite).

Covoiturage - suite

1. Ajoutez à la classe Covoiturage :
   - une méthode getVilles renvoyant la liste des villes possibles représentées (celles des personnes et des voitures); la méthode précédente peut vous être utile,
   - une méthode capaciteSuffisante ne prenant pas d'argument et renvoyant vrai si il y a suffisamment de places dans les voitures dans chaque ville pour que tout le monde puisse faire ses trajets,
   - une méthode estPossible renvoyant vrai si non seulement la capacité est suffisante mais si en plus il y a assez de personnes en capacité de conduire *Attention : pensez à prendre en priorité les voitures de grande capacité de façon à minimiser le nombre de chauffeurs.*.,
   - une méthode Attribution qui renvoie un tableau d'entiers : à la position i du tableau se trouve le numéro de la voiture qui conduira la personne d'identifiant i.

Recherche

Pour savoir dans quelle voiture une personne ira, il lui faut connaître son identifiant.

1. Ajoutez une méthode getIdentifiant(String nomPersonne) qui renvoie l'identifiant de la personne nomPersonne. Si une telle personne n'existe pas, cette méthode renvoie -1. Si plus d'une personne porte ce nom, la méthode renvoie l'un des identifiants existant pour ce nom.

2. Si la liste des personnes était triée par ordre croissant de nom (par ordre alphabétique), la recherche pourrait être rendue plus efficace. Ajoutez une méthode triePersonnes qui trie la liste des personnes par ordre de nom croissant.

3. Ajoutez une méthode getIdentifiantDichotomique(String nomPersonne) qui a le même comportement que getIdentifiant, mais en utilisant une recherche dichotomique (donc bien plus efficace).