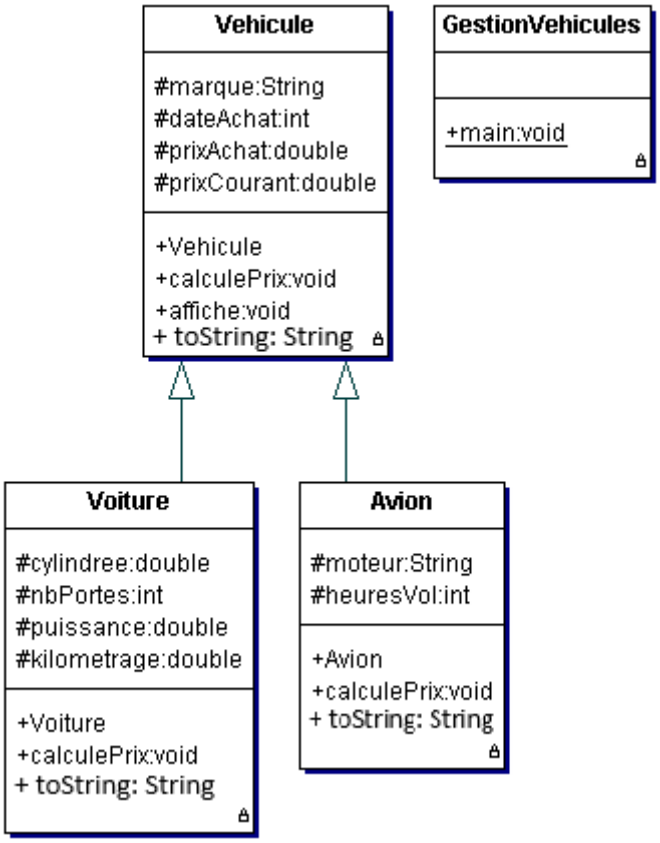
TD n°6 – Classes abstraites et interfaces

Flotte de véhicules



Reprendre l'exercice des semaines précédentes dont les classes sont rappelées ci-dessus.

1. Modifier le code de la classe `Vehicule` pour la transformer en classe abstraite.
2. Quelle(s) méthode(s) n'a(ont) pas besoin d'être implémentée(s) dans cette classe ? Rendre cette(ces) méthode(s) abstraite(s).
3. Pourquoi la classe abstraite `Vehicule`, qui ne peut pas être instanciée, contient quand même un constructeur ?

Pokemons

1. De même que pour `Vehicule`, la classe `Pokemon` doit-elle être abstraite ou non ? Justifier.
2. Rendre la classe `Pokemon` abstraite, et déclarer les méthodes abstraites qui sont implémentées dans les sous-classes.
3. Créer une interface `IAttaque` qui propose une déclaration de la méthode attaquer. Vous devez de nouveau **factoriser le type** du paramètre de la fonction afin de permettre à des Pokemons de s'attaquer, mais également à d'autres types d'objets comme des Joueurs comme nous le verrons dans les TDs suivants.
4. Modifier votre code de manière à implémenter cette interface.

Véhicules

On considère une autre classe modélisant des véhicules. Chacun a un numéro d'identification (attribué) automatiquement et une distance parcourue (initialisée à 0), et parmi eux, on distingue les véhicules à moteur qui ont une capacité de réservoir et un niveau d'essence (initialisé à 0) et les véhicules sans moteur qui n'ont pas de caractéristique supplémentaire. Parmi les véhicules à moteur, les voitures ont un nombre de places, les camions, un volume transporté. Parmi les véhicules sans moteur, les vélos ont un nombre de vitesse.

1. Construire en UML le graphe hiérarchique des classes décrites ci-dessus.
2. Ecrire le code java des classes `Vehicule`, `AMoteur`, `SansMoteur` avec tous les constructeurs nécessaires et la méthode ***toString()***, qui retourne une chaîne de caractères décrivant l'état de l'objet.
3. Ecrire une méthode ***void rouler (double distance)*** qui fait avancer un véhicule. A quel niveau de la hiérarchie faut-il l'écrire ?
4. Ecrire les méthodes ***void approvisionner (double nbLitres)*** et ***boolean enPanne()***.
   A quel niveau de la hiérarchie faut-il les écrire ?
5. Ecrire la classe `Velo` avec constructeur et méthode ***toString()*** et une méthode ***void transporter (String depart, String arrivee)*** qui affiche par exemple :

```
Le vélo n°2 a roulé de Paris à Nice.
```

6. Ecrire la classe `Voiture` avec constructeur, ***toString()*** et une méthode ***void transporter (int n, int km)*** qui affiche par exemple :

```
La voiture n°3 a transporté 5 personnes sur 250 km ou bien
Plus d'essence si elle est à sec.
```

7. Ecrire la classe `Camion` avec constructeur, ***toString()*** et une méthode ***void transporter (String materiau, int km)*** qui affiche par exemple :

```
Le camion n°4 a transporté des tuiles sur 400 km ou bien
Plus d'essence s'il est à sec.
```

8. Peut-on factoriser la déclaration de la méthode ***transporter*** et si oui, à quel niveau ?
9. On considère le ***main*** suivant (à placer dans un fichier `T_Vehicule.java`) :

```
public static void main (String [] args) {
        Vehicule v1 = new Velo (17); // nb vitesses
        Vehicule v2 = new Voiture (40.5, 5); // capacité réservoir, nb Places
        Vehicule v3 = new Camion (100.0, 100.0); // capacité réservoir, volume
        System.out.println ("Vehicules : "+"\n" +v1 +"\n" +v2 +"\n" +v3 +"\n");
        v2.approvisionner (35.0) ; // litres d'essence
        v3.approvisionner (70.0);
        v1.transporter ("Dijon ", "Valence ") ;
        v2.transporter (5, 300) ;
        v3.transporter (" des tuiles", 1000) ;
}
```

Ce programme est-il correct ? Le corriger si nécessaire. Qu'affiche-t-il ?

10. Définir des ***interfaces*** pour l'approvisionnement et le transport. Modifier les classes en conséquence.