M2103-- TD4

Préambule

1. Expliquez brièvement la relation d'héritage et son utilité pour la programmation orientée objet.
2. Etant donne une classe `Superclasse` et une classe `Sousclasse`, quel est le mot dédié, utilisé dans la déclaration de la sousclasse, qui indique à Java qu'une classe hérite d'une autre classe ?
3. Quel est le mot dédié qui fait référence à une superclasse à partir d'une de ses sousclasses ?
4. Une sousclasse peut hériter :
    a. D'au maximum une classe seulement ;
    b. D'au maximum deux classes, si elles sont abstraites ;
    c. De n'importe combien de superclasses si ces-dernières sont toutes concrètes ;

    Vous aves les classes Java suivantes :

    ```
    Animal
    Arbre
    Humain
    Fleur
    Loup
    Lion
    Mammifère
    Insecte
    Pigeon
    Abeille
    Plante
    ```

5. Ordonnez ces classes dans des pyramides, en indiquant quelle(s) classe(s) héritent de quelles classes.
6. Et si on rajoute la classe `Organisme` ?
7. Les objets dans toutes ces classes peuvent grandir. Par contre, seulement les animaux et les insectes peuvent bouger. Au contraire, seulement les plantes peuvent faire une photosynthèse pour se nourrir. Dans quelles classes ci-dessus voulez-vous ajouter les méthodes qui représentent ces trois capacités : grandir, bouger, photosynthese ?

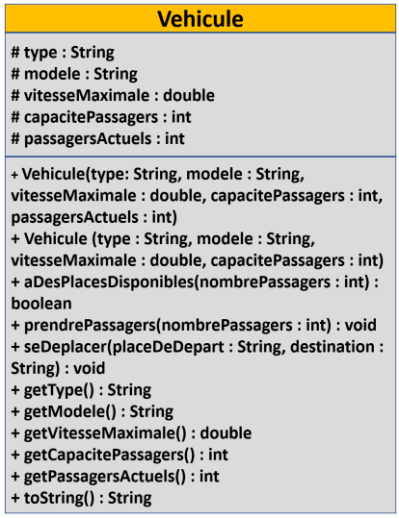
Contexte

Nous allons simuler des divers types de véhicules : par exemple des voitures et des avions. Ces véhicules ont quelques caractéristiques communes, mais aussi des caractéristiques spéciales. Ils ont une vitesse maximale et un modèle. Les véhicules peuvent se déplacer et peuvent prendre des passagers. Par contre, les avions feront partie d'une compagnie aérienne et auront un nombre de partitions pour les sièges (ceci influence leur capacité de prendre plus ou moins de passagers).

La modélisation de ce type d'écosystème justifie l'utilisation de l'héritage.

Exercice 1

Dans cet exercice, nous allons commencer sur une superclasse `Vehicule` dont le diagramme de classe est donné ci-dessous.



Dans la classe `Vehicule` (voir le diagramme de classe ci-dessous), nous avons les prochains attributs :

- Un attribut `type` de type `String` qui indique si le véhicule est une "`Voiture`", "`Camion`", ou "`Avion`"
- Un attribut `modele` de type `String` qui indique le modèle du véhicule
- Un attribut `vitesseMaximale` de type double qui indique la vitesse maximale de ce véhicule en km/h
- Un attribut `capacitePassagers` de type `int` qui indique le nombre maximal de passagers que le véhicule peut accueillir, y compris son conducteur (chauffeur ou pilote)
- Un attribut `passagersActuels` qui stocke le nombre de passagers actuellement dans le véhicule.

Ces attributs auront la mention protected, plutôt que private.

Les méthodes de cette classe sont :

- Deux constructeurs
- Des getters pour chaque attribut
- Une méthode `String toString` qui retourne `<modele>`, `<vitesseMaximale>`, `<passagersActuels>/<capacitePassagers>`.
- Deux méthodes pour manipuler le nombre de passagers, notamment `aDesPlacesLibres` et `prendrePassagers.`

1. Pour ce premier exercice nous allons nous concentrer sur les constructeurs. Le premier initialise chaque attribut de la classe à la valeur correspondante mise en entrée. Pour le deuxième constructeur, le nombre actuel de passagers sera 1 (le chauffeur). Ecrivez le deuxième constructeur.

2. La méthode `boolean aDesPlacesLibres(int nombrePlaces)` retourne `true` s'il y a encore de la place pour prendre `nombrePlaces` passagers dans le véhicule, et `false` autrement. La méthode `void prendrePassagers(int nombrePassagers)` vérifie si on a encore de la place pour le nombre indiqué de passagers et, si c'est le cas, on ajoute ces passagers dans le véhicule. Sinon, aucun passager n'est pris au bord.

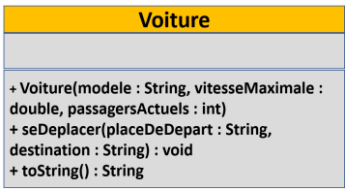
   Ecrivez la méthode `void prendrePassagers(int nombrePassagers)`.

3. Modifiez le code de la question précédente pour mettre en place une politique différente. S'il y a assez de place dans la voiture pour prendre `nombrePassagers` passagers, alors la politique est la même ; par contre si on n'a pas assez de place, on prendra le nombre de passagers possibles jusqu'à remplir la voiture. Réécrivez cette méthode.

Exercice 2

Nous aurons une classe `Voiture` qui hérite de la classe `Vehicule`. Les objets de la classe `Voiture` auront l'attribut type de la classe `Voiture` mis à `"Voiture"`. Même si ceci n'est pas toujours le case en réalité, nous allons supposer que toutes les voitures ont 5 places (y compris le chauffeur).

La classe `Voiture` aura le diagramme de classe suivant :



Notamment, cette classe n'a aucun attribut en plus par rapport à sa superclasse. Ella aura, par contre, un constructeur et réécrira les méthodes `seDeplacer` et `toString` de sa superclasse.

1. La classe `Voiture` a un constructeur. Etant donné la structure de la classe `Vehicule`, ce constructeur est :
   a. Non-obligatoire, car toute classe Java peut utiliser le constructeur par défaut dans l'absence d'un autre constructeur.
   b. Non-obligatoire, mais souhaitable dans ce cas-ci (bonne pratique)
   c. Obligatoire

2. Combien d'attributs un objet de type `Voiture` a-t-il ?
3. Ecrivez le constructeur de la classe `Voiture` en utilisant le constructeur de sa superclasse.
4. Dans la classe `Voiture` on veut que la méthode String toString() retourne le texte :
   `Voiture[<modele>, <vitesseMaximale>, <passagersActuels>/<maxPassagers>]`

   Ecrivez la méthode `String toString()` de la classe `Voiture`.
5. Voici la méthode `seDeplacer` de la classe `Vehicule` :

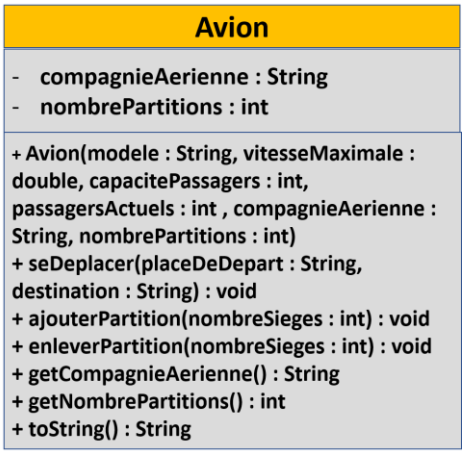
```
public void seDeplacer(String pointDeDepart, String destination) {
      System.out.println("Deplacement : " + pointDeDepart + " vers " +
destination);
}
```

   Ecrivez une méthode `seDeplacer` pour la classe `Voiture` qui retourne le message :

   `Vous prenez la route entre <placeDeDepart> et <destination>`

Exercice 3

La classe `Avion` (dont le diagramme de classe est donnés ci-dessous) hérite également de la classe `Vehicule`. Celle-ci aura deux variables en plus par rapport à sa superclasse.



- Un attribut `compagnieAerienne` de type `String` qui indique le nom de la compagnie aérienne propriétaire de l'avion ;
- et un attribut `nombrePartitions` de type `int` qui indique le nombre de partitions utilisées pour les sièges (ceci indiquera le nombre de passagers en fonction du modèle).

La classe `Avion` aura également les méthodes suivantes :

- Un constructeur qui met le type du véhicule à “Avion” et le reste des attributs aux valeurs données en entrée.
- Deux getters pour les nouveaux attributs
- Deux méthodes pour manipuler le nombre de partitions (en augmentant ou en baissant le nombre de places dans l’avion).

1. Ecrivez le constructeur de cette classe.
2. La méthode `void ajouterPartition(int nombreSieges)` incrémente par 1 le nombre de partitions dans un avion et diminue par `nombreSieges` la capacite de l’avion. Si le nombre de passagers actuels dépasse cette nouvelle capacité, alors le nombre excessif de passagers sera mis dehors, pour ne pas dépasser la capacité. Ecrivez cette méthode.

Exercice 4

Dans cet exercice le but sera d’utiliser le code réalisé pour les exercices précédents. Sauf indication contraire tout le code ci-dessus sera dans une classe TD4 dans une méthode `public static void main(String[] args)`. Le but sera d’utiliser la notion de polymorphisme.

1. Cochez les caisses correspondant à des instructions qui compileront et s’exécuteront correctement (en isolation) :

   ☐ `Vehicule avion = new Vehicule("Avion", "Boeing 747-400", 920, 524, 384);`
   ☐ `Vehicule avion = new Avion("Boeing 747-400", 920, 524, 384, "Air France", 2);`
   ☐ `Avion avion = new Avion("Boeing 747-400", 920, 524, 384, "Air France", 2);`
   ☐ `Avion avion = new Vehicule("Avion", "Boeing 747-400", 920, 524, 384);`

2. Reprenez les instructions qui compilent et s’exécutent dans l’exercice précédent. Expliquez ce qui se passe à chaque fois si on prend l’instruction en question et on rajoute dans la méthode `main` l’instruction suivante :

   ```
   System.out.println(avion);
   ```

3. Même question si on y rajoute l’instruction :

   ```
   avion.ajouterPartition(20);
   System.out.println(avion);
   ```

4. Nous allons déclarer et instancier le tableau suivant :

   ```
   Vehicule[] vehicules = new Vehicule[3]
   ```

Reprenez les lignes de codes qui compilaient à l'exercice 1. Lesquels des objets ci-dessus pourraient être mis dans le tableau vehicules en utilisant l'instruction :

```
vehicules[0] = avion;
```

5. Supposons qu'après la déclaration de l'objet `vehicules` on rajoute la ligne de code :
   `vehicules[2] = new Avion("Boeing 747-400", 920, 524, 384, "Air France", 2);`

   Qu'est-ce qui se passe si on appelait la méthode enleverPartition pour cet objet en utilisant l'instruction :

   ```
   vehicules[2].enleverPartition(20);
   ```

6. Nous voyons que la méthode `seDeplacer()` est différente pour chacun type de véhicule ci-dessus. Quelle serait la conséquence de l'enlever de la classe `Vehicule` et de la garder dans toutes les sousclasses (avec le code actuel) ?