TD 4 L'héritage

Exercice I

Vous aves les classes Java suivantes :

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

- Ordonnez ces classes en indiquant quelle(s) classe(s) héritent de quelles classes.
- Et si on rajoute la classe `Organisme` ?
- 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 ?

Exercice 2

Nous allons simuler des divers types de véhicules : par exemple des voitures et des camions. Ces véhicules ont quelques caractéristiques communes, mais aussi des caractéristiques spéciales. Ils ont un type de carburant, une capacité du réservoir, un nombre maximal de passagers. Les deux types de véhicules peuvent faire le plein, consommer du carburant et prendre des passagers ; par contre, un camion pourra aussi charger/décharger de la marchandise.

Ceci justifie l'utilisation de l'héritage.

Dans cet exercice, nous allons commencer sur une superclasse `Vehicule`.

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

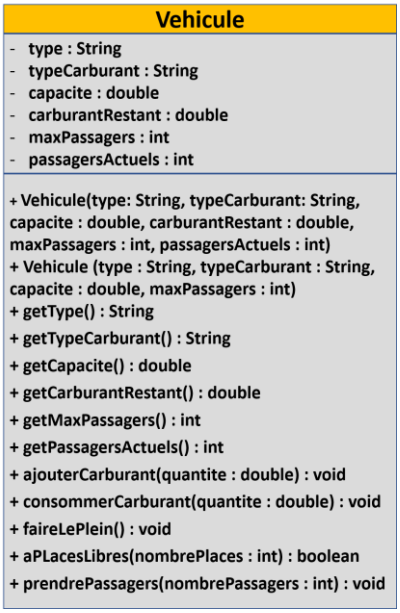
- Un attribut `type` de type `String` qui indique si le véhicule est une (”`Voiture`” ou un ”`Camion`”),
- Un attribut de type `String` qui indique type de carburant qu'on utilise
- La capacité du réservoir du véhicule,

- Le volume de carburant dont il dispose maintenant,
- Le nombre maximal de passagers qu'il peut accueillir, y compris le chauffeur,
- Le nombre actuel de passagers.

Les méthodes de cette classe sont :

- Deux constructeurs
- Des getters pour chaque attribut
- Une méthode toString() qui retourne [<nom d'attribut> : <valeur de l'attribut>] séparés par des vergules.
- Des méthodes pour manipuler la quantite de carburant : `void consommerCarburant()`, `void faireLePlein()` et `void ajouterCarburant()`
- Des méthodes pour manipuler le nombre de passagers, notamment `boolean aPlacesLibres()` et `void prendrePassagers(int nombrePassagers).`

Consultez les diagrammes de classe ci-dessous.



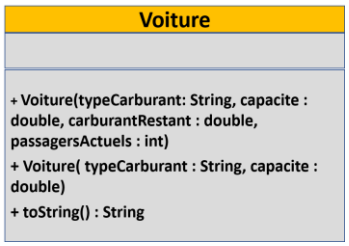
- Le premier constructeur mettra les valeurs de chaque attribut de la classe à la valeur correspondante mise en entrée. Pour le deuxième constructeur, le carburant restant sera mis par défaut à la valeur de la capacité du réservoir, tandis que le nombre actuel de passagers sera 1 (le chauffeur). Ecrivez ce constructeur.
- Les méthodes `ajouterCarburant(double quantite)` et `consommerCarburant(double quantite)` nous permettent de modifier la quantité de carburant actuelle qui existe dans la voiture par la quantité mise en entrée, sans dépasser la capacité et sans tomber sous 0. Ecrivez la méthode `ajouterCarburant`.
- La méthode `void faireLePlein()` remplira la voiture d'essence. Utilisez la méthode `ajouterCarburant(double quantite)` pour écrire cette méthode.
- La méthode `boolean aPlacesLibres()` rend `true` s'il y a encore de la places pour prendre des 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 y ajoute le nombre de passagers. Ecrivez la méthode `void prendrePassagers(int nombrePassagers)`.

Exercice 3

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". Meme 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` est une sousclasse de la classe `Vehicule`.

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



Notamment, cette classe n'a aucun attribut par rapport à sa superclasse. Elle aura, par contre, deux constructeurs et elle modifiera la méthode toString() de sa superclasse.

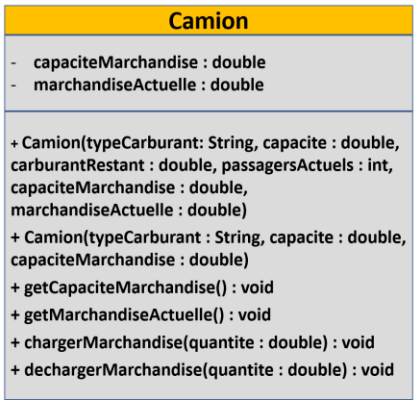
- Rappel : qu'est-ce qu'une superclasse et qu'est-ce qu'une sousclasse ?
- Qu'est-ce que la classe `Voiture` hérite de sa superclasse ?
- Les attributs de la classe `Vehicule` sont des valeurs privées. Est-ce qu'on aura accès à ces attributs à partir de la classe `Voiture` ?
- Quel est le mot dédié qui indique à Java qu'une classe hérite d'une autre classe ?
- Quel est le mot dédié qui fait référence à une superclasse à partir d'une de ses sousclasses ?
- Est-ce qu'on peut hériter de plusieurs classes ?
- Pourquoi la classe `Voiture` ne peut pas utiliser les constructeurs de la classe `Vehicule` directement ?
- Ecrivez le premier constructeur de la classe `Voiture` en utilisant le constructeur de sa superclasse.
- Le deuxième constructeur de la classe `Voiture` crée une voiture qui a le réservoir plein et seulement 1 passager (le chauffeur). Indiquez deux façons d'écrire ce deuxième constructeur et utilisez une de ces méthodes pour écrire le deuxième constructeur.
- Dans la classe `Voiture` on veut utiliser un autre format pour renvoyer les attributs des objets de type `Voiture`, notamment :
  `Voiture[typeCarburant> ; Carburant : <carburantRestant> / <capacite> ; Passagers : <passagersActuels> / <maxPassagers>]`

  Quelle est la mention qu'il faut ajouter lorsqu'on réécrit une méthode de la superclasse ?
  Ecrivez la méthode `String toString()` de la classe `Voiture`.

Exercice 4

Voici le diagramme de classe de la classe `Camion`, qui hérite de la classe `Vehicule` de la même façon que la classe `Voiture`.

La classe `Camion` aura deux attributs propres, au-delà des attributs hérités de la superclasse (`Vehicule`) :



- Un attribut `capaciteMarchandise` de type `double`, qui indique la quantité totale (en m^3) de marchandise que le camion peut porter
- Un attribut `marchandiseActuelle` de type `double` qui indique la quantité totale (en m^3) de marchandise portée actuellement par le camion.

La classe Camion aura également les méthodes suivantes :

- Deux constructeurs (le deuxième met la quantité de marchandise actuelle à 0)
- Deux getters pour les nouveaux attributs
- Deux méthodes pour manipuler la marchandise `void chargerMarchandise(double quantite)` et `void dechargerMarchandise(double quantite).`

On va faire écrire le code suivant :

- Ecrivez le premier constructeur de la classe `Camion`
- Ecrivez la méthode `void dechargerMarchandise(double quantite)`

Exercice 5

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)`.

- Dans la méthode principale (main) nous aurons le code suivant :

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

Ce tableau contiendra deux voitures (places 0 et 2) et un camion (position 1), les trois avec ¼ de la capacite maximale de carburant. Vous pouvez choisir les autres paramètres librement.

Ecrivez du code qui vous permettra justement de remplir le tableau.

- Ce code est un exemple de quel concept de la Programmation Orientée Objet ?
- Pourquoi est-ce qu'on peut mettre un objet de type `Camion` dans un tableau de type `Vehicule[]` ?
- Utilisez les méthodes `toString()` pour faire afficher les caractéristiques de chaque vehicule. Quelles méthodes sont utilisées dans chaque cas ?
- Faites tous les véhicules faire le plein.
- On suppose qu'il y a deux passagers dans chaque vehicule. Ecrivez du code qui essaie de faire chaque vehicule prendre un nouveau passager. Quel devrait être le résultat ?
- Qu'est-ce que se passe lorsqu'on écrit la ligne de code dans la classe principale :

```
vehicules[1].chargerMarchandise(12.5);?
```