TP2

Exercice 1 : Créer une Classe

Comme vu en cours, une classe est la représentation d'un Objet « concret ».

Dans un package (que vous nommerez judicieusement `vehicule`), créez une classe `Vehicule` comportant les attributs suivant :

- `modele`
- `nombrePlaces`
- `poids`

Créez un constructeur prenant en paramètres les trois attributs de `Vehicule` pour les initialiser.

Encapsulation : Dans Java, lorsque vous créez une classe, vous voulez maîtriser totalement la manière dont elle sera utilisée. C'est pour cela que vous devez cloisonner au maximum votre code pour ensuite ouvrir les vannes par des méthodes publiques permettant de manipuler les attributs de votre classe.

Si vous ne l'avez pas déjà fait, encapsulez votre classe en mettant l'accessibilité de vos attributs en `private`, puis en leur créant un getter et un setter chacun. Faites-le pour toutes les classes à venir (et pas que dans ce TP d'ailleurs).

Redéfinissez la méthode `toString` pour afficher les attributs de `Vehicule`.

Créez une classe `Test` dans laquelle vous créerez une méthode `main` pour tester votre code de la façon suivante : pour l'instant, simplement créer un véhicule et l'afficher.

Exercice 2 : Héritage

Créez les classes `Avion` et `Voiture` qui héritent toutes les deux de `Vehicule`. Un avion possède un attribut `altitudeMax`, tandis qu'une voiture a un attribut `nombrePortes`.

Créez un constructeur pour chacune des deux classes précédentes, qui prennent en arguments tous les attributs de `Vehicule` ainsi que les attributs spécifiques à chaque classe fille. Pour chaque classe, redéfinissez la méthode `toString` pour afficher le**s** attribut**s** de la classe concernée.

Instanciez un avion et une voiture, puis affichez-les dans votre classe `Test`.

Véhicule est un terme dont le sens est trop large pour vraiment avoir un sens une fois instancié. Un véhicule est forcément quelque chose de plus spécifique comme un avion, une

voiture, une moto…
Pour représenter cela, faites de `Vehicule` une classe abstraite. Elle ne pourra alors plus être instanciée (faites attention à `Test`). Modifiez l'accessibilité de tous ses attributs en remplaçant le mot-clé `private` par `protected`, pour qu'ils soient accessibles directement par les classes filles de `Vehicule`.

Exercice 3 : Énumérations

Créez une classe enum `Propulsion` (qui auras son propre fichier) avec les valeurs `DIESEL` et `ESSENCE`. Ajoutez un attribut `propulsion` de type `Propulsion` à `Voiture`.

Créez alors un constructeur qui prend en arguments tous les attributs de `voiture`, en tenant compte de la propulsion, mais sans modifier le constructeur déjà existant. Utilisez le mot-clé `this()` pour éviter de réécrire l'initialisation de `nbPortes`. Complétez la méthode `toString` pour prendre en compte le nouvel attribut.

Exercice 4 : Interfaces

Créez les classes `Planeur` et `Chasseur` qui héritent toutes les deux de la classe `Avion`. Ajoutez l'attribut `puissance` aux classes `Voiture` et `Chasseur`, ainsi que l'attribut `portance` à la classe `Planeur`. Créez l'interface `Motorise` définissant la méthode `getConsomation()`, qui ne prend aucun argument et retourne une valeur de type `float`, et faites en sorte que les classes `Voiture` et `Chasseur` implantent cette interface. Attention, chacun de ces deux véhicules ne calcule pas sa consommation de la même manière :

- Calcul de la consommation pour une voiture :
  - Si la voiture roule à l'essence : $(puissance \times 5) \div poids$.
  - Si la voiture roule au diesel : $(puissance \times 3) \div poids$.

  (utilisez un `switch case`)

- Calcul de la consommation pour un chasseur : $puissance \div poids$.

Créer une méthode `toString` pour toutes les classes qui n'en ont pas encore, en affichant la consommation pour les classes qui possèdent cette information.

Exercice 5 : Listes

Dans votre `main`, créez une liste de véhicules que vous remplirez d'une instance de chaque classe existante (`Voiture`, `Avion`, `Chasseur`, `Planneur`). Faites une boucle de type foreach afin de parcourir tous les éléments de cette liste et d'appeler `toString` sur chacun.

Exercice 6 : Attributs et méthodes statiques

Ajoutez un attribut `static nbVehicule` qui sera initialisé à 0 au chargement de la classe, puis incrémentez cet attribut à chaque nouvelle instanciation d'un objet de type `Vehicule`. Créez une méthode statique et publique `getNbVehicule` qui renvoie la valeur de l'attribut correspondant.