TD 6 Les Interfaces

Dans ce TD nous allons nous concentrer sur les interfaces.

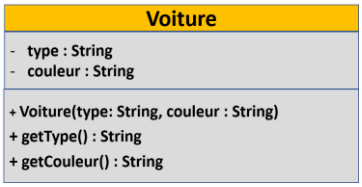
Notre exemple portera sur des voitures avec plusieurs types de combustible. L'idée des interfaces est précisément de spécifier le fonctionnement de faire le plein, mais aussi de faire des restrictions par rapport au type de combustible utilisé.

Comme c'est très facile au début de faire des confusions entre les interfaces et l'héritage, nous allons utiliser les deux dans ce TP.

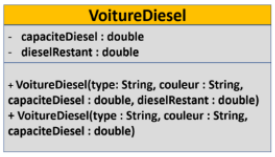
Contexte

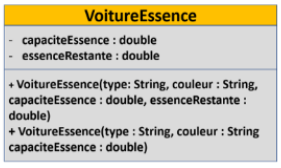
Nous allons partir sur une première classe `Voiture`, qui servira de superclasse à quatre types de voitures : les voitures Diesel (qui prendront du Diesel comme carburant), les voitures Essence, les voitures électriques et les voitures hybrides (qui prennent de l'essence ET de l'électricité). Les voitures thermiques (Diesel, Essence) auront un réservoir d'une capacité donnée. Elles pourront faire le plein en remplissant leur réservoir avec le carburant utilisé. De même pour le réservoir essence des voiture hybrides. Pour les voitures électriques et hybrides, nous allons considérer leur charge électrique comme un pourcentage de la valeur maximale pour la batterie : par exemple 100% (maximum), 90%, 18.3%, etc. D'habitude, on nous conseille de ne jamais recharger au maximum une voiture électrique, sinon de la recharger à 90% tous les jours.

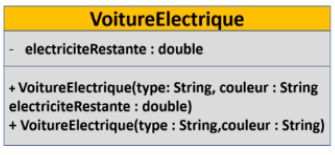
Voici les diagrammes des classes de base qu'on va utiliser (la relation d'héritage n'est pas indiquée sur la figure). Premièrement voici la classe `Voiture` :

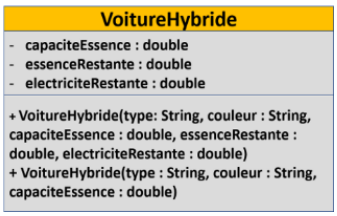


Puis voici les diagrammes des sousclasses `VoitureDiesel`, `VoitureEssence`, `VoitureElectrique`, `VoitureHybride`.









Exercice 1

Pour rentrer dans le rythme, on commence avec quelques méthodes simples dans les classes ci-dessous. N'oubliez pas d'utiliser le diagramme de classe en tant que repère.

- Prenez le temps pour vous familiariser avec les diagrammes de classe ci-dessus. Pouvez-vous vous imaginer à quoi correspond chaque attribut de chaque classe ?

SOLUTION :

Classe Voiture : le type de voiture représente un modèle, et la couleur dénote sa couleur.

Sousclasses VoitureDiesel, VoitureEssence : le premier attribut dénote la capacité totale de carburant dans cette voiture (le type de carburant est Diesel pour VoitureDiesel, Essence pour VoitureEssence). Le deuxième attribut dénote le carburant restant dans la voiture. Il faut bien sûr se rappeler que, même s'ils ne sont pas mentionnés directement, les attributs de la superclasse (type et couleur) sont hérités directement par chaque sousclasse.

Sousclasse VoitureElectrique : la capacité totale de carburant dans le cas d'une voiture électrique est de 100%. L'attribut electriciteRestante dénote la capacité restante d'électricité dans la voiture.

Sousclasse VoitureHybride : les objets de cette classe auront des attributs qui stockent la capacité en essence dans la voiture, ainsi que le carburant et l'électricité restante.

- La classe `Voiture` a deux attributs de type `String`, notamment le type de voiture et sa couleur. En plus, chaque sousclasse aura des attributs liés à leur carburant, tel qu'on a décrit dans les diagrammes de classes ci-dessus.

  Le premier constructeur de chaque sousclasse appelle le constructeur de la superclasse. Le but des constructeurs de la sousclasse sont d'initialiser les attributs de la superclasse aux deux premières valeurs, puis d'initialiser les attributs propres à la classe par les valeurs des paramètres suivants.

  Ecrivez le premier constructeur de la classe `VoitureDiesel`.

SOLUTION :

```
      public VoitureDiesel (String type, String couleur, double capaciteDiesel,
double dieselRestant) {
            super(type, couleur);
            this.capaciteDiesel = capaciteDiesel;
            this.dieselRestant = dieselRestant;
      }
```

- Le deuxième constructeur des classes `VoitureDiesel` et `VoitureEssence`, la quantité de carburant restant (Diesel ou Essence) sera mise à la capacité maximale du réservoir. Pour la classe `VoitureElectrique`, l'électricité restante est mise à la capacité pleine, c.à.d. 100%. Finalement, pour les voitures hybrides on combine les caractéristiques d'une `voitureEssence` et une `voitureElectrique`.

  Ecrivez ce deuxième constructeur pour la classe VoitureHybride.

SOLUTION :

```
	public VoitureHybride(String type, String couleur, double
capacite Essence) {
		this(type, couleur, capaciteEssence, capaciteEssence, 100);
	}
```

- Dans une classe TD6, dans la méthode principale (`main`) créez 4 objets représentant 4 voitures spécifiques : une Renault Clio (Diesel, capacité réservoir 55 l), une Tesla Model S (Electrique), une Peugeot 208 (Essence, capacité réservoir 50 l) et une KIA Optima Hybride (hybride, capacité réservoir 55l). Vous pouvez choisir les couleurs de votre choix pour ces voitures. Pour chaque voiture thermique la quantité de carburant restant sera mis à 10, et pour les valeurs électriques la quantité de carburant restant sera mis à 10%.

  Ecrivez le code qui vous permet de créer ses objets et puis d'afficher leurs couleurs dans la méthode main.

SOLUTION :

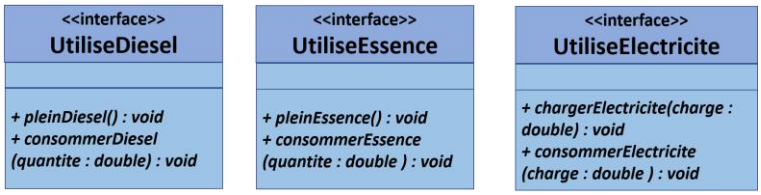
```
	final VoitureDiesel renaultClio = new VoitureDiesel("Renault Clio", "blanche",
55, 10);
	final VoitureEssence peugeot208 = new VoitureEssence("Peugeot 208",
"argentee", 55, 10);
	final VoitureElectrique teslaS = new VoitureElectrique("Tesla Model S",
"rouge", 10);
	final VoitureHybride kiaOptimaHybride = new VoitureHybride("Kia Optima
Hybride", "jaune", 50, 10, 10);

	final Voiture[] toutesVoitures = {renaultClio, peugeot208, teslaS,
kiaOptimaHybride};

	for(int i = 0; i < toutesVoitures.length; i++) {
		System.out.println(toutesVoitures[i].getCouleur());
	}
```

Exercice 2

Mettre le mauvais carburant dans une voiture peut causer des problèmes sérieux pour une voiture. Il est essentiel alors de ne jamais le faire. Nous allons s'imaginer ces trois interfaces : `UtiliseEssence`, `UtiliseDiesel`, `UtiliseElectricite`, dont les diagrammes de classe sont donnés ci-dessous.



- Rappel : Est-ce qu'une classe peut hériter de plusieurs classes ?

SOLUTION :

Non, une classe peut hériter au maximum d'une seule superclasse.

- Rappel : Est-ce qu'une classe peut implémenter plusieurs interfaces ?

SOLUTION :

Oui, une classe peut implémenter autant d'interfaces qu'elle veut.

- Rappel : comment déclare-t-on une interface ?

SOLUTION :

interface <nomDInterface>

- Rappel : une interface est-elle une classe ?

SOLUTION :

Non, une interface n'est pas une classe. Par exemple, une interface ne peut jamais contenir des attributs, ne peut jamais inclure des méthodes concrètes et toutes ses méthodes sont implicitement publiques et abstraites.

- Rappel : Est-ce qu'une interface peut contenir des méthodes concrètes ?

SOLUTION :

Non, jamais. Une interface ne peut contenir que de méthodes abstraites.

- Rappel : Comment écrit-on des méthodes dans une interface ?

SOLUTION :

C'est la syntaxe habituelle des méthodes abstraites, sauf qu'on ne doit plus écrire les mots public ou abstract. Notamment :

`<type de sortie ou void> <nom de la méthode>(<paramètres en entrée>)`

- Rappel : Pourquoi le texte des méthodes des interfaces ci-dessus est-il mis en Italique ?

SOLUTION :

Puisque les méthodes d'une interface sont implicitement abstraites.

- Ecrivez les trois interfaces ci-dessus

SOLUTION :

```
interface UtiliseDiesel {

        void pleinDiesel();
        void consommerDiesel(double quantite);
}

interface UtiliseElectricite {

        void chargerElectricite(double charge);
        void consommerElectricite(double charge);
}

interface UtiliseEssence {

        void pleinEssence();
        void consommerEssence(double quantite);

}
```

Exercice 3

Nous voulons faire les diverses voitures implémenter les interfaces écrites dans l'exercice précédent. Notamment nous voulons que la classe `VoitureDiesel` suit le fonctionnement indiqué par l'interface `UtiliseDiesel`. La classe `VoitureEssence` devrait fonctionner comme indiqué par `UtiliseEssence`. La classe `VoitureElectrique` devra suivre `UtiliseElectricite`. Finalement la classe `VoitureHybride` devra implémenter les méthodes des interfaces `VoitureElectrique` et `VoitureEssence`.

- Rappel : comment peut-on indiquer qu'une classe fonctionne selon les spécifications d'une interface ?

SOLUTION :

Le mot clé est implements. La syntaxe est :

`<mention> class <nomClasse> <si héritage : extends <nomSuperclasse>> implements <interfaces, séparées par des virgules>`

- Rappel : les classes `VoitureDiesel`, `VoitureEssence`, `VoitureElectrique`, `VoitureHybride` sont des classes concrètes. Doivent-elles impérativement détailler les méthodes qu'elles implémentent de leurs interfaces respectives ?

SOLUTION :

Oui, car elles sont des classe concrètes, tandis que les méthodes d'une interface sont impérativement abstraites. Une classe concrète doit toujours implémenter les méthodes abstraites de ses superclasses et des interfaces qu'elles implémentent.

- Rappel : une classe peut-elle avoir des méthodes en dehors de celles indiquées par les interfaces qu'elle implémente ?

SOLUTION :

Oui. Le fait d'implémenter une interface n'indique que la présence de quelques méthodes (de l'interface) au sein de la classe. La classe pourra après implémenter plusieurs autres méthodes.

- Ecrivez les méthodes que la classe `VoitureElectrique` implémente de l'interface `UtiliseElectricite`. Le paramètre de la méthode `chargerElectricite` représente la charge maximale jusqu'à laquelle on veut charger la voiture. Au contraire, le paramètre de la méthode `consommerElectricite` représente la quantité d'électricité qu'on a dépensé.

SOLUTION :

```
	@Override
	public void chargerElectricite(double charge) {
		this.electriciteRestante = charge;
		if(this.electriciteRestante > 100) {
			this.electriciteRestante = 100;
		}
	}

	@Override
	public void consommerElectricite(double charge) {
		this.electriciteRestante -= charge;
		if(this.electriciteRestante < 0) {
			this.electriciteRestante = 0;
		}
	}
```

Exercice 4

Dans cet exercice nous voulons utiliser les classes et méthodes vues dans les exercices antérieurs. Vous pouvez désormais supposer que toutes les méthodes indiquées dans les diagrammes de classe sont bien implémentées.

- Vous avez les quatre voitures créées dans l'exercice 1, qu'on va appeler : `renaultClio`, `teslaS`, `peugeot208` et `kiaOptimaHybride`. Dans la méthode main écrivez du code permettant à chaque voiture de l'exercice précédent de faire son plein en carburant et de se recharger jusqu'à une charge maximale de 90(%). La voiture hybride fera les deux : il fera son plein en carburant et se rechargera jusqu'à 90%.

  SOLUTION :
  Idéalement on voudrait bien utiliser notre tableau de voitures pour faire le plein de chaque voiture. Malheureusement, ceci n'est pas actuellement possible car « faire le plein » corresponde à des diverses méthodes dans les différentes classes.
  On fait donc le plein une voiture par une.

```
renaultClio.pleinDiesel();
peugeot208.pleinEssence();
teslaS.chargerElectricite(90);
kiaOptimaHybride.pleinEssence();
kiaOptimaHybride.chargerElectricite(90);
```

- Disons qu'on a le code suivant dans la méthode `main` :

```
VoitureDiesel fiatPunto = new VoitureDiesel("Fiat Punto", "rouge", 45, 30);
fiatPunto.pleinDiesel() ;
fiatPunto.recharge(90) ;
```

  Est-ce que ce code compile ? Pourquoi (ou pourquoi pas) ?

  SOLUTION :
  La première ligne compile : on appelle le constructeur correctement, avec le nombre et type correct de paramètres. La deuxième ligne compile correctement : on appelle bien la méthode `pleinDiesel()`, qui existe dans la classe `VoitureDiesel`. Par contre la troisième ligne de code ne fonctionne pas, car une méthode recharge n'existe pas dans la classe `VoitureDiesel`.

- Même question pour le code suivant :

```
Voiture seatIbiza = new VoitureDiesel("SEAT Ibiza", "bleue", 55, 20);
seatIbiza.pleinDiesel() ;
```

  SOLUTION :
  Le code ne compile pas. La voiture `seatIbiza` est déclarée en tant que `Voiture`, même si on instancie cet objet en appelant le constructeur de la classe `VoitureDiesel`.

- Nous allons remplacer le code de la méthode main par un autre code. Premièrement vous allez considérer que les prochains objets ont été correctement instanciés :

  - `renaultClio` : modèle Renault Clio, Diesel, capacité 55, actuellement à 10

- `seatIbiza` : modèle SEAT Ibiza, Essence, capacité 55, actuellement à 10
- `fordMondeo` : modèle Ford Mondeo, Essence, capacité 65, actuellement à 10
- `peugeot208` : modèle Peugeot 208, Essence, capacité 50, actuellement à 10
- `teslaS` : modèle Tesla S, Electrique, actuellement à 10% charge
- `teslaX` : modèle Tesla X Electrique, actuellement à 15% charge
- `toyotaYarisHybride` : modèle Toyota Yaris Hybride, voiture hybride, capacité 50l, actuellement à 20, charge actuelle 10%
- `kiaOptimaHybride` : modèle Kia Optima Hybride, voiture hybride, capacité 55l, actuellement à 10, charge actuelle 20%

Dans une station d'essence les véhicules vont s'aligner dans des queues qui stockent des voitures d'un même type. On va modéliser les queues en tant que tableaux. Le premier tableau s'appellera `pompeDiesel` et sera de type `UtiliseDiesel[]`, le deuxième, `pompeEssence`, sera de type `UtiliseEssence[]`, le troisième, `stationElectrique`, sera de type `UtiliseElectricite[].` Déclarez et initialisez ces trois tableaux. Les hybrides feront leur plein en carburant ET se rechargeront, donc les voitures vont apparaître dans les deux queues.

SOLUTION :

Cet exercice vous donne un exemple de polymorphisme avec des interfaces. On disait tout à l'heure qu'il n'y avait pas une seule méthode qui nous permettait de faire le plein de tous les divers types de voitures qu'on avait dans notre méthode main. Ceci était fait express : on veut diviser les voitures par leur fonctionnement (tel que l'on fait en réalité également lorsqu'on fait le plein d'une voiture).
Le code sera donc :

```
	final UtiliseDiesel[] voituresPleinDiesel = {renaultClio};
	final UtiliseEssence[] voituresPleinEssence = {peugeot208, seatIbiza,
fordMondeo, kiaOptimaHybride, toyotaYarisHybride};
	final UtiliseElectricite[] voituresPleinElectricite = {teslaS, teslaX,
kiaOptimaHybride, toyotaYarisHybride};
```

- Pour chaque élément de chaque tableau faites son plein en carburant et/ou rechargez-le à 90%.

SOLUTION :

```
	for (int i = 0; i < voituresPleinDiesel.length; i++) {
		voituresPleinDiesel[i].pleinDiesel();
	}

	for (int i = 0; i < voituresPleinEssence.length; i++) {
		voituresPleinEssence[i].pleinEssence();
	}

	for (int i = 0; i < voituresPleinElectricite.length; i++) {
		voituresPleinElectricite[i].chargerElectricite(90);
	}
```