# Héritage : Exercice StarWars

A travers cet exercice, nous allons découvrir la conception de hiérarchie de classes ainsi que les différentes notions liées à l’héritage.

Pour cela créez un projet ***StarWars*** contenant le package **heritage**.

## Etape 1

Dans le package ***ste***, créez un sous package ***starwars***.

Nous allons définir la classe ***Main*** qui sera le point d’entrée de notre programme et contiendra la méthode ***main***.

### Héritage et redéfinition de méthodes

Nous allons définir 2 classes représentant des vaisseaux :

* ***XWing***
* ***MilleniumFalcon***

Ces classes contiendront les méthodes :

* **public** **void** decoller()
* **public** **void** afficherType()

Le MilleniumFalcon peut atteindre la vitesse de la lumière, il contiendra un méthode supplémentaire :

* **public** **void** vitesseDeLaLumiere()

Nous allons définir une 3eme classe ***DeathStar***, qui correspond à la station spatiale qui peut attirer des vaisseaux. Il lui faut donc une méthode ***attirerVaisseau,***  cette méthode doit pouvoir attirer n’importe quel type de vaisseau.

A ce stade, il faudrait 2 méthodes ***attirerVaisseau***, une qui prend en argument le type ***XWing*** et une qui prend en argument le type ***MilleniumFalcon.***

Donc, si un nouveau type de vaisseau vient s’ajouter il faudra à créer une nouvelle méthode ***attirerVaisseau.***

Pour pallier à ce genre de situation qui peut vite devenir ingérable il faut penser à créer une classe générique ***Vaisseau.***

Les vaisseaux ont chacun des caractéristiques propres, mais ils ont aussi beaucoup de choses en commun. Dans notre programme, on veut travailler sur les particularités de chaque vaisseau, mais aussi pouvoir manipuler les vaisseaux plus globalement sans faire de distinctions.

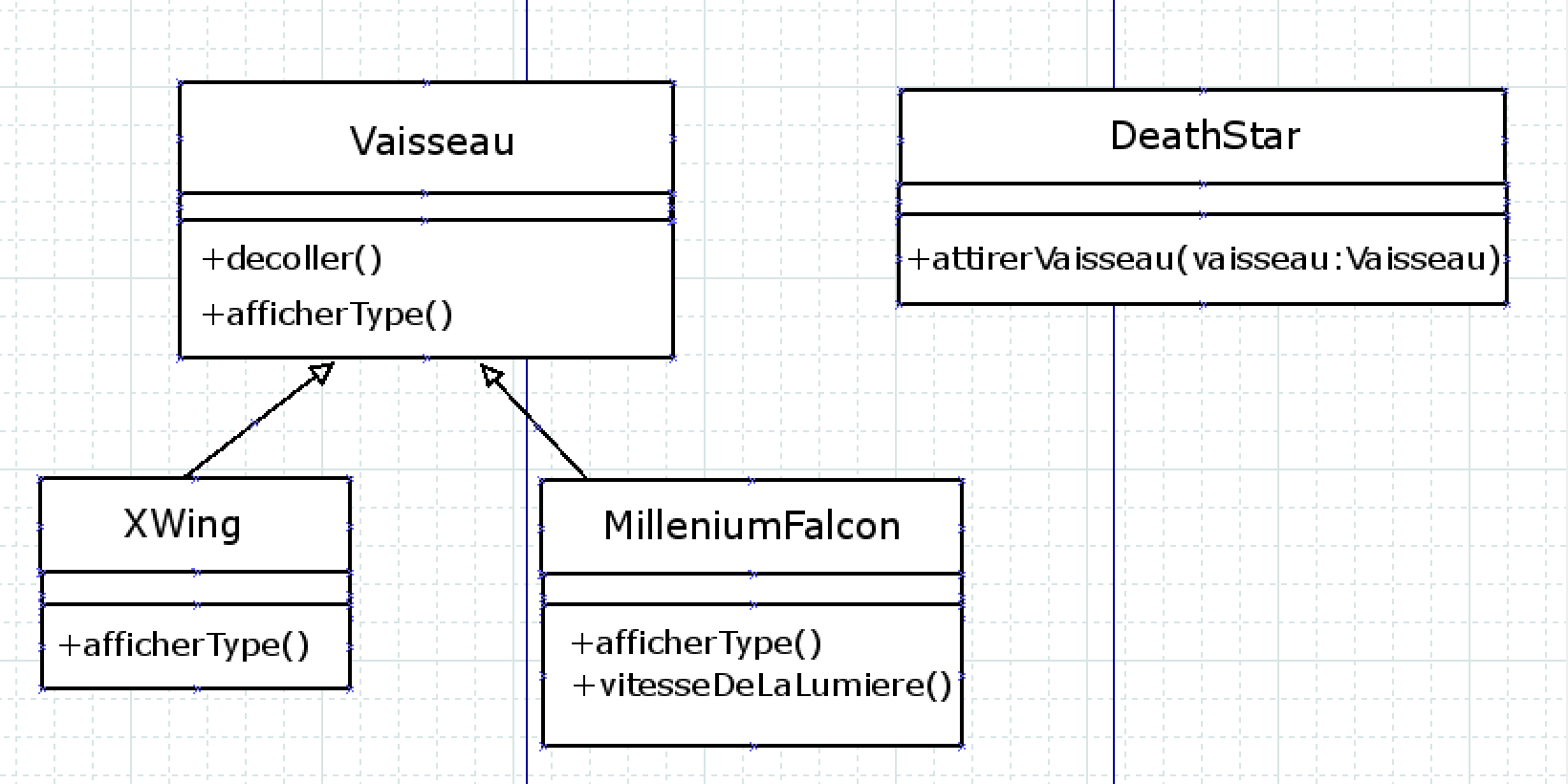
Plus concrètement: on veut disposer d'une classe ***Vaisseau*** générique et définir les classes ***Xwing***, ***MilleniumFalcon*** comme une version particulière de ***Vaisseau***.

Voilà ce que l’on nomme l’héritage.

La classe ***Vaisseau*** ne définit que les caractéristiques communes à tous les appareils. On dira que que ***Xwing*** et ***MilleniumFalcon*** sont des *classes dérivées* de ***Vaisseau*** et que ***Vaisseau*** représente une *superclasse ou classe de base de* ***Xwing*** et ***MilleniumFalcon.***

La classe ***Xwing*** est dérivée de ***Vaisseau***. On signale cet état de fait à l'aide du mot clé ***extends***. Par défaut, la classe dérivée est iden­tique à sa superclasse. Nous devons donc redéfinir les méthodes spécifiques à Xwing : ***afficherType().***

Une classe dérivée peut aussi contenir des champs ou des méthodes qui n'existent pas dans la superclasse, pour ***MilleniumFalcon*** il y aura un méthode supplémentaire : ***vitesseDeLaLumiere().***



### Accès aux membres

Le mot clé ***public*** définit le mode d’accès le moins restrictif : les classes, méthodes ou variables ***public*** est accessible par tout objet.

Le mot clé ***private*** définit le mode d’accès le plus restrictif : les variables, méthodes ou classes ***private*** ne sont accessibles que par la classe dans laquelle elles sont définies.

NB : à rappeler que le principe d’encapsulation permet de garantir l'intégrité des données contenues dans l'objet. Ce mécanisme consiste à rassembler les données et les méthodes au sein d'une classe en empêchant l'accès aux données par un autre moyen que les méthodes d’accesseurs (getters) et mutateurs (setters). Dès lors, le mot clé ***private*** est utilisé dans la déclaration des variables d’instances de la classe.

Nous voulons ajouter une méthode ***tirerMissile*** dans la classe Vaisseau :

**private** **int** nombreDeMissiles;

**public** **void** tirerMissile(){

nombreDeMissiles=nombreDeMissiles-1;

}

tirerUnMissile est public: on peut donc écri­re sans problèmes vaisseauDeLuke.tirerUnMissile() car la méthode est héritée. Elle s'exécute et accède bien au champ nombreDeMissiles, lui aussi hérité. Cepen­dant, comme il est en private dans la superclasse, il n'existe aucun moyen de définir une nouvelle méthode dans Xwing qui accè­de à ce champ. Il faut donc introduire un nouveau mode d'accès moins restrictif : ***protected.***

Le mode d'accès **protected** ressemble à private, sauf qu'un membre protected hérité est accessible dans la classe dérivée (et au niveau package).

***A retenir :***

|  |
| --- |
| * **extends**: Une classe hérite d'une autre classe par le biais du mot clé ***extends*** (La classe dérivée étend la superclasse/*classe de base*). * **Héritage**:   + Une classe ne peut hériter que d'une seule classe.   + Une classe dérivée hérite des propriétés et aptitudes de sa superclasse et peut alors être personnalisée en y ajoutant des données, méthodes ou en modifiant le comportement des méthodes dont elle a hérité.   + L'héritage peut se faire sur plusieurs niveaux (superclasse, sous-classes, sous-classes de sous-classes, …). Les relations d'héritage sont à considérer comme une arborescence où chaque classe hérite non seulement de son parent immédiat mais aussi de tous les ancêtres de ce parent. * ***Redéfinition de méthodes (overriding) :***   + La classe dérivée peut redéfinir les méthodes de la classe de base, il est recommandé d’utiliser l’annotation ***@Override***. En plaçant ce mot-clé dans le commentaire de la méthode réécrite, le compilateur vérifiera que la méthode est correctement utilisée (mêmes arguments, même type de valeur de retour) et affichera un message d'avertissement si ce n'est pas le cas.   + La méthode redéfinie doit avoir la même signature et le même type de retour que la méthode de la superclasse * ***protected :*** Les variables, méthodes ou classes définies comme protégées ne sont accessibles que par les classes dérivées et classes du même package |