**GLOSSAIRE JAVA**

Java est un langage de programmation *orienté objets*

#### **Classe:** Une **classe** est la représentation d'un type de donnée,

une définition de la structure et du comportement communs à un ensemble d'objets et un créateur d'objet.

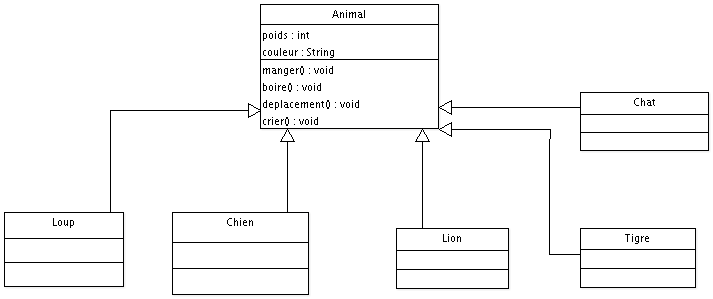
**Classe abstraite :** Identique à une classe normale mais on ne peut pas l’instancier ( si Animal est une classe abstraite il n’est pas possible de faire

Animal obj = new Animal(); //Erreur de compilation !

abstract class Animal{

abstract void manger(); //Une méthode abstraite uniquement dans une classe abstraite

}



#### **Objet :** Un **objet** est une entité, une chose concrète ou abstraite qui peut être reconnue distinctement. On appelle ainsi une instance d'une classe.

#### **Type:** Un **type** est un ensemble de caractéristiques communes à des objets.

En programmation, le type est une notion abstraite mais importante, concrétisée par différents moyens.

#### **Instance:** Une [classe](http://alsoftware.pagesperso-orange.fr/french/formation/java/glossaire.html#classe) est une description abstraite d'un ensemble d'objets. Une instance de classe est un objet construit selon le modèle fourni par la classe. Un objet et une instance de classe peuvent être considérés comme synonyme.

**Héritage :** C'est un principe fondamental de la programmation objet. Les sous-classes d'une classe disposent de tous les attributs et méthodes de sa super-classe, moyennant néanmoins quelques nuances liées aux [modificateurs de visibilité](http://alsoftware.pagesperso-orange.fr/french/formation/java/glossaire.html#modificateur) (privé, public) et au principe de la [redéfinition](http://alsoftware.pagesperso-orange.fr/french/formation/java/glossaire.html#redefinir) des méthodes.

**Classifieur**: Un **classifieur** est une définition et un ensemble d'objets ou de classifieurs. La notion de classifieur est un généralisation de celle de classe :

* Toute classe est un classifieur.
* Tout ce qui ressemble à une classe (une définition de la structure et du comportement communs à un ensemble d'objets) est un classifieur.
* Tout ce qui ressemble à un ensemble de classes (ou de classifieurs) est un classifieur. *Remarque :* En Java, les classes, les interfaces, les paquetages… sont des classifieurs.

#### **Méthode:** On appelle ainsi ce qu'on appellerait fonction ou procédure dans d'autres langages. Toute méthode fait partie d'une classe. Une méthode peut être soit [d'instance](http://alsoftware.pagesperso-orange.fr/french/formation/java/glossaire.html#dInstance), soit [de classe](http://alsoftware.pagesperso-orange.fr/french/formation/java/glossaire.html#deClasse).

#### **Modificateurs de visibilité (public, private,protected):** Il est possible d'indiquer pour une classe ou un champ (attribut ou méthode) d'une classe un certain degré d'accessibilité. Cela se fait avec les mots [public](http://alsoftware.pagesperso-orange.fr/french/formation/java/reserve.html#public), [private](http://alsoftware.pagesperso-orange.fr/french/formation/java/reserve.html#private) ou [protected](http://alsoftware.pagesperso-orange.fr/french/formation/java/reserve.html#protected) situés au début de l'en-tête de la classe ou du champ en question. Ces mots sont les modificateurs de visibilité.

#### Il faudra des getters et des setters pour accéder à des attributs private.

Les méthodes auxquelles on voudra accéder devront être public

**Constructeur**: Un **constructeur** est une fonction particulière dont le rôle est de créer un nouvel objet.

**Destructeur**: Un **destructeur** est une procédure particulière dont le rôle est de détruire un objet existant.

|  |
| --- |
| **class** Personne {  String nom;Personne conjoint;Date dateNaissance;  */\* un constructeur, avec un paramètre de nom « n » et de type « String » \*/*Personne(String n) {*/\* Ici, le code du constructeur nous importe peu. \*/*}  **void** setConjoint(Personne c) {…}  **int** âge() {…}  } |

**Accesseur(getter, setter)**: Un **accesseur** est une méthode qui permet d'accéder en lecture ou en écriture à la valeur d'un attribut. Par convention, le rôle des accesseurs se limite à rendre, pour ceux en lecture, ou à modifier, pour ceux en écriture, la valeur de l'attribut concerné, et rien de plus.

**Surcharge(*overloading*)**: La **surcharge** est la capacité que possède un classifieur d'avoir plusieurs membres portant le même nom. Réécrire une méthode dans uns classe fille.

**Encapsulation**: L'**encapsulation** consiste à réunir au sein d'un classifieur la structure (attributs) et le comportement (méthodes) d'un type de donnée.

Les avantages de l'encapsulation sont :

* Elle cache les détails de programmation au sein de chaque classifieur. Donc, les programmeurs, à l'exception de celui du classifieur, n'ont pas à se préoccuper de la manière dont celui-ci est programmé.
* Inversement, les détails étant cachés, les autres programmeurs ne peuvent les prendre en compte et le programmeur du classifieur est ainsi libre de modifier sans risque ces détails.
* L'encapsulation améliore la cohérence : tous les membres d'un classifieur s'appliquent à un type de donnée particulier et sont les seuls à le faire.
* Elle améliore aussi la facilité de modification et de maintenance des programmes : tout ce qui concerne un type de donnée particulier est regroupé dans son classifieur et non plus disséminé dans tout le programme.

**Agrégation, composition**: L'**agrégation** (on dit aussi **composition**) est la capacité d'un objet à être constitué de manière concrète ou abstraite par d'autres objets ou à faire référence à d'autres objets.

**Redéfinition**: La **redéfinition** est une technique qui permet, dans un classifieur héritier, de donner une nouvelle définition d'un membre déjà présent dans un classifieur hérité.

La redéfinition est souvent confondue avec la surcharge, alors que ce sont deux choses totalement différentes. Par exemple, si on a un classifieur Parallélogramme qui définit une méthode aire, alors le classifieur Carré, qui hérite de Parallélogramme, peut redéfinir aire pour lui donner un code plus adapté.

**Polymorphisme**: Le **polymorphisme** est la capacité d'un objet à prendre plusieurs formes, c'est-à-dire à être manipulé en fonction de son classifieur, mais aussi en fonction d'autres classifieurs compatibles avec le sien.

Le polymorphisme est, avec l'encapsulation et l'héritage, le troisième point fort spécifique aux langages à objets.

Par exemple, un objet de type Employé peut être manipulé comme instance du classifieur Employé bien sûr, mais aussi comme instance (indirecte) de Personne ou encore comme un très général Objet.

Ex: class abstraite vehicule avec une méthode venir => méhode venir différente dans la classe voiture, dans la classe trotinette…

**Interface**: Une interface est une classe 100 % abstraite,elle contient des méthodes abstraites, mais sans le mot clé *abstract*.

**public interface I{**

**public void A();**

**public String B();**

**}**

**public class X implements I{**

**public void A(){**

**//…**

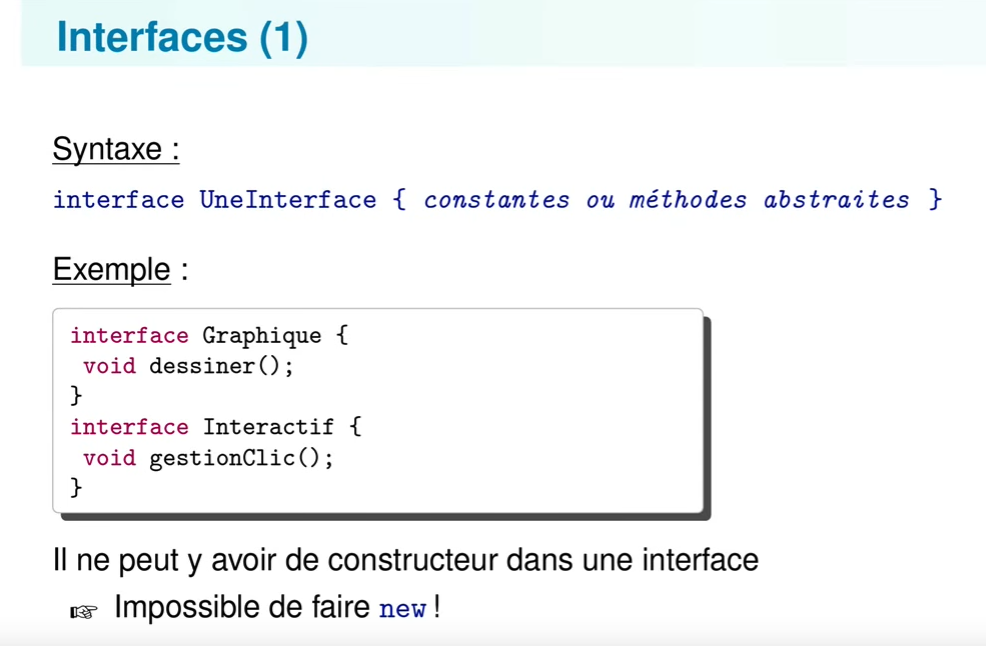
**}**

**public String B(){**

**//…**

**}**

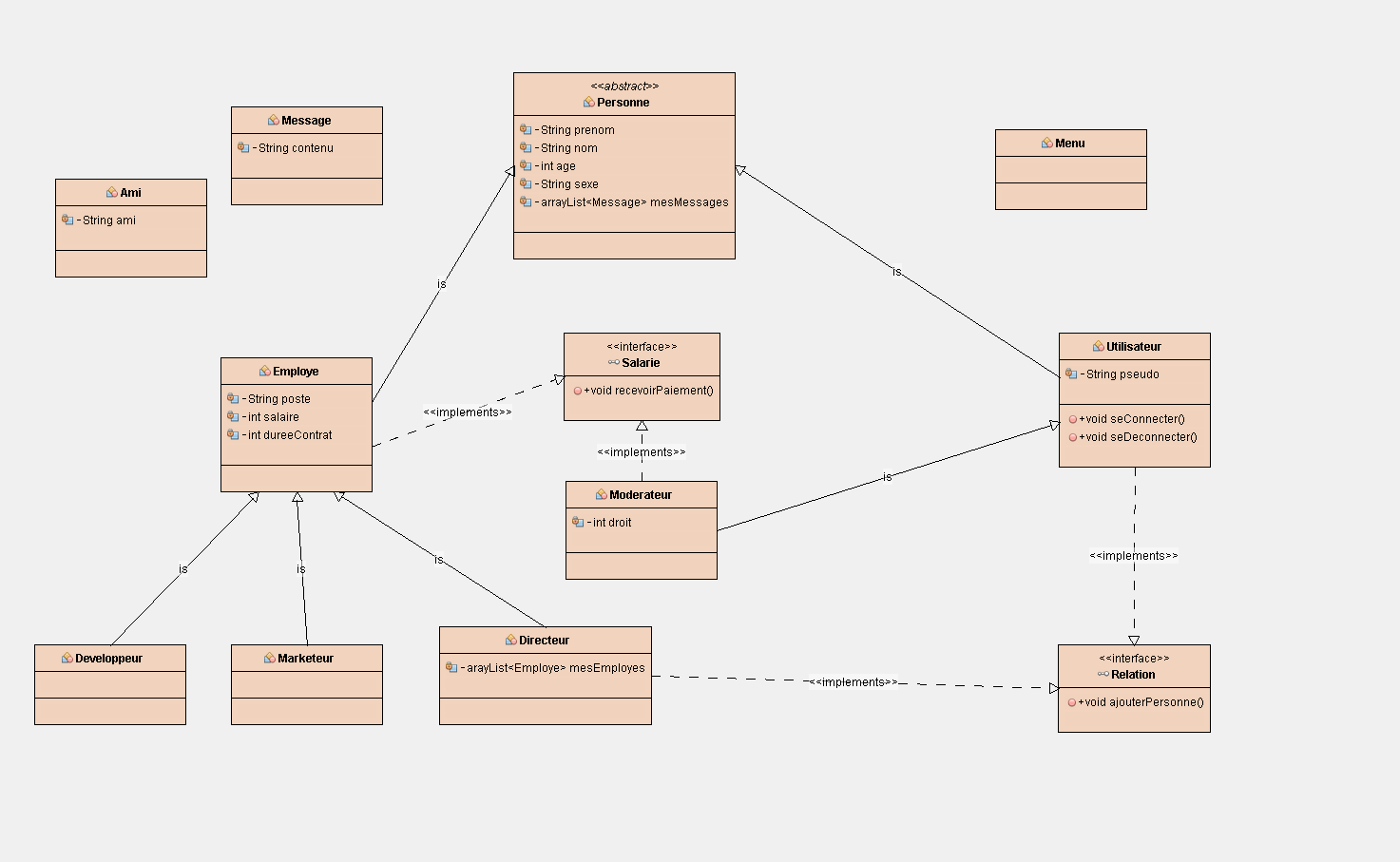
**}**

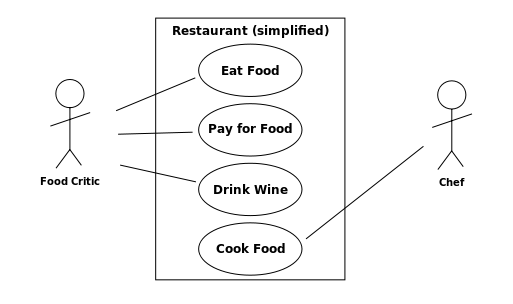


**ArrayList**:

* add() permet d'ajouter un élément ;
* get(int index) retourne l'élément à l'indice demandé ;
* remove(int index) efface l'entrée à l'indice demandé ;
* isEmpty() renvoie « vrai » si l'objet est vide ;
* removeAll() efface tout le contenu de l'objet ;
* contains(Object element) retourne « vrai » si l'élément passé en paramètre est dans l'ArrayList.

**Diagramme de classe UML:**





**Diagramme de use case:**

Les diagrammes de cas d'utilisation sont des [diagrammes UML](https://fr.wikipedia.org/wiki/Unified_Modeling_Language#Les_diagrammes) utilisés pour donner une vision globale du comportement fonctionnel d'un système [logiciel](https://fr.wikipedia.org/wiki/Logiciel)