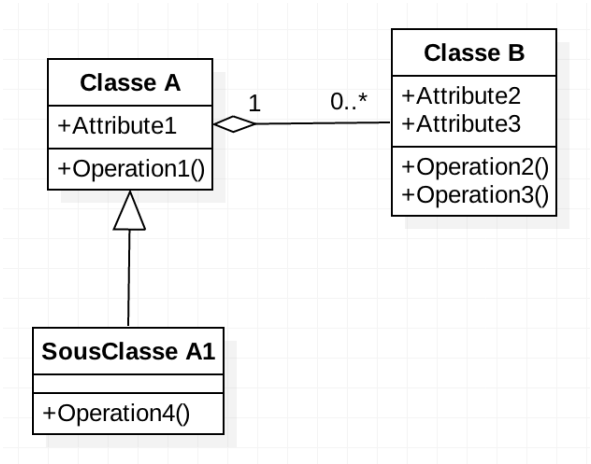
Langage de modélisation
Diagramme de classe

I/ Introduction

Le diagramme de classes est le point central dans un développement orienté objet. En analyse, il a pour objectif de décrire la structure des entités manipulées par les utilisateurs. En conception, le diagramme de classes représente la structure d'un code orienté objet ou, à un niveau de détail plus important, les modules du langage de développement.

II/ Représentation

Le diagramme de classe met en œuvre des classes, contenant des attributs et des opérations, et reliées par des associations ou des généralisations.



Exemple de diagramme des classes

III/ Classe et objet

Une classe représente la description abstraite d'un ensemble d'objets possédant les mêmes caractéristiques. On peut parler également de type.
Exemples : La classe 'Voiture', la classe 'Personne'.

Un objet est une entité aux frontières bien définies, possédant une identité et encapsulant un état et un comportement. Un objet est une instance (ou occurrence) d'une classe.

Exemple : Pascal DUPONT est un objet instance de la classe 'Personne'. Le cours que vous tenez entre vos mains est une instance de la classe 'cours'.

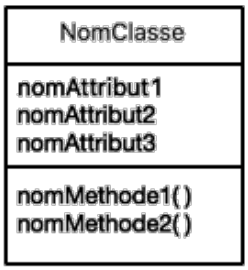
IV/ Attribut et opération

Un attribut représente un type d'information contenu dans une classe.

Exemples : vitesse courante, cylindrée, numéro d'immatriculation, etc… sont des attributs de la classe 'Voiture'.

Une opération représente un élément de comportement (un service) contenu dans une classe. Nous ajouterons surtout les opérations en conception objet, car cela fait partie des choix d'attribution des responsabilités aux objets.

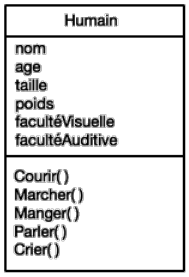
V/ La forme simplifiée de représentation des classes



Représentation d'une classe

- La première partie contient le nom de la classe.
- La deuxième partie contient les attributs. Ceux-ci contiennent l'information portée par un objet. L'ensemble des attributs forme la structure de l'objet.
- La troisième partie contient les méthodes. Celles-ci correspondent aux services offerts par l'objet. Elles peuvent modifier la valeur des attributs. L'ensemble des méthodes forme le comportement des objets.

Exemple :



Cette forme de représentation des classes est la plus simple car elle ne fait pas apparaître les caractéristiques des attributs et des méthodes en dehors de leur nom. Elle est très souvent utilisée lors d'une première phase de modélisation.

1) L'encapsulation

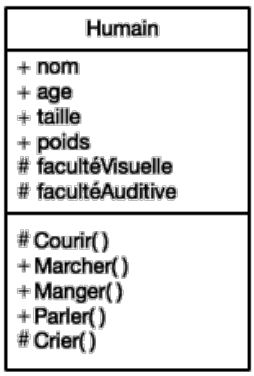
Certains attributs et méthodes ne sont pas exposés à l'extérieur de l'objet. Ils sont encapsulés et appelés attributs et méthodes privés de l'objet.
UML, comme la plupart des langages objets modernes, introduit trois possibilités d'encapsulation :

- L'attribut privé ou la méthode privée : la propriété n'est pas exposée en dehors de la classe, y compris au sein des sous-classes.
- L'attribut protégé ou la méthode protégée : la propriété n'est exposée qu'aux instances de la classe et des sous-classes.
- L'encapsulation de paquetage : la propriété n'est exposée qu'aux instances des classes de même paquetage.

La notion de propriété privée est rarement utilisée car elle amène à instaurer une différence entre les instances de classes. Cette différence est liée à des aspects assez subtils de la programmation par objets. Quant à l'encapsulation de paquetage, elle provient du langage Java. Elle est réservée à l'écriture de diagrammes destinés aux développeurs.

L'encapsulation est représentée par un signe 'plus', un signe 'moins, un 'dièse ' ou un 'tilde' précédent le nom de l'attribut. Le tableau suivant détaille la signification de ces signes.

public	+	Élément non encapsulé visible par tous.
protégé	#	Élément encapsulé visible dans les sous-classes de la classe.
privé	-	Élément encapsulé visible seulement dans la classe.
paquetage	~	Élément encapsulé visible seulement dans les classes du même paquetage.

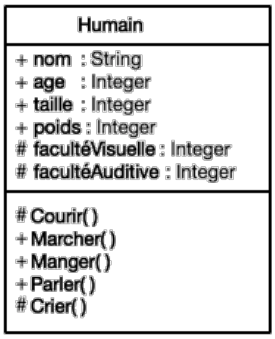


2) Les types

Nous appelons ici variable tout attribut, paramètre et valeur de retour d'une méthode. D'une façon générale, nous appelons variable tout élément pouvant prendre une valeur.

Le type d'attribut, d'un paramètre et de la valeur de retour d'une méthode est précisé au niveau de la représentation de la classe.

Exemple 'Classe Humain avec les attributs typés' :



3) La cardinalité

Une variable (attribut, paramètre et valeur de retour d'une méthode) peut contenir plusieurs variables. Dans la plupart des langages de programmation, une telle variable est appelée un tableau ou une liste.

La cardinalité est indiquée à la suite du type avec la syntaxe suivante :

[borneInf … borneSup]

Le nombre de valeurs que prend la variable est alors compris entre borneInf et borneSup. Il est possible d'indiquer une seule borne qui précise le nombre de valeurs que doit prendre exactement la variable. Le symbole * peut être utilisé comme borneSup. Il signifie qu'il n'y a pas de borne supérieure limitant le nombre de valeurs que peut prendre la variable. La syntaxe [0..*] peut également être écrite [*].

Exemple :
Nous faisons l'hypothèse qu'un humain peut posséder plusieurs noms avec un minimum de 1 et un maximum de 3.
La syntaxe pour l'attribut nom est alors la suivante :

+ nom : String [1..3]

4) Les propriétés des variables

UML permet d'attribuer des propriétés à une variable (attribut, paramètre et valeur de retour d'une méthode) en les indiquant entre accolades. Les propriétés suivantes figurent parmi les plus utilisées.

{readOnly} : cette propriété indique que la valeur de la variable ne peut pas être modifiée. Elle doit être initialisée avec une valeur par défaut.

{redefines nomAttribut} : cette propriété n'est applicable qu'à un attribut. Elle spécifie la redéfinition de l'attribut de nom nomAttribut de l'une des surclasses. La redéfinition peut notamment porter sur le nom de l'attribut ou sur son type. En cas de changement de type, le nouveau type doit être compatible avec l'ancien, c'est à dire que son ensemble de valeurs doit être inclus dans l'ensemble des valeurs de l'ancien type.

{ordered} : lorsqu'une variable peut contenir plusieurs valeurs (cardinalité supérieur à 1), les valeurs doivent être ordonnées.
{unique} : lorsqu'une variable peut contenir plusieurs valeurs (cardinalité supérieur à 1), chaque valeur doit être unique (interdiction d'avoir des doublons). Cette propriété s'applique par défaut.
{nonunique} : lorsqu'une variable peut contenir plusieurs valeurs (cardinalité supérieur à 1), la présence de doublons est possible.

Pour attribuer plusieurs propriétés à une variable, il faut les séparer par des virgules.

Exemple :
Nous reprenons l'hypothèse qu'un humain peut posséder plusieurs noms avec un minimum de 1 et un maximum de 3. Nous désirons maintenant que l'attribut multivalué soit composé de noms uniques et ordonnés. Nous utilisons alors la syntaxe suivantes :

+ nom : String[1..3] {unique, ordered}

5) La signature des méthodes

Une méthode d'une classe peut prendre des paramètres et renvoyer un résultat. Les paramètres sont des valeurs transmises :

- À aller, lors de l'envoi d'un message appelant la méthode ;
- Ou au retour d'appel de la méthode.

Le résultat est une valeur transmise à l'objet appelant lors du retour d'appel.
Ces paramètres ainsi que le résultat peuvent être typés. L'ensemble constitué du nom de la méthode, des paramètres avec leur nom, leur type, leur cardinalité, leurs propriétés ainsi que du type de résultat avec sa cardinalité et ses propriétés s'appelle la signature de la méthode.

Une signature prend la forme suivante :

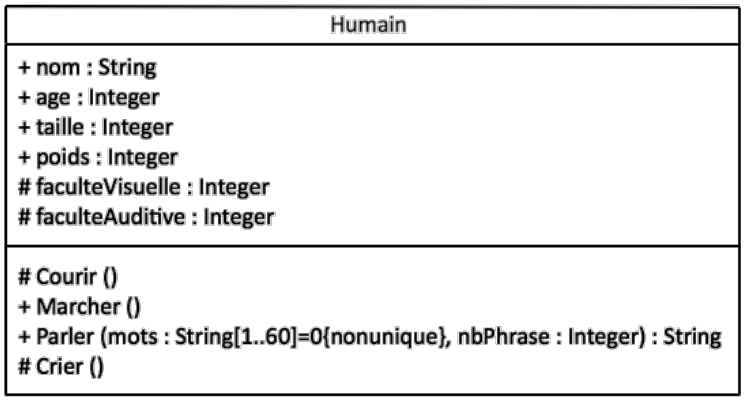
nomMethode (nomParametre : type[borneInf..borneSup]=ValeurDéfaut{proprietes}, …) :
typeResultat[borneInf..borneSup]{proprieties}

Exemple, la méthode 'parler()':

parler (mots : String[1..60] = 0 {nonunique}, nbPhrase : Integer) : String

Rappelons que le nombre de paramètres peut être nul et que le type du résultat est optionnel.

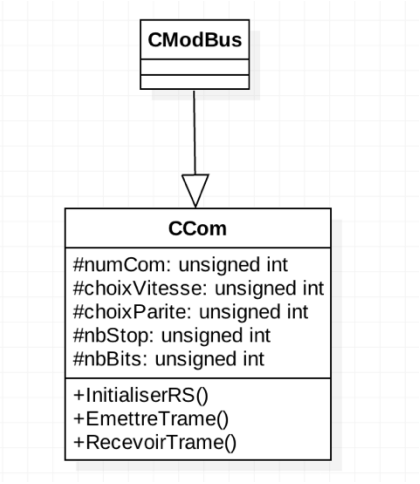
Exemple :



VI/ Relation entre les classes

1) Héritage

Représentation UML :



La classe CModBus hérite de la classe CCom

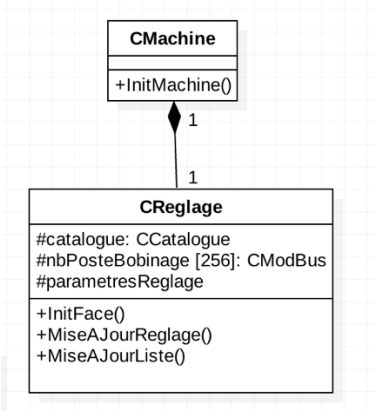
Déclaration en C++ :

```
class CModBus : public CCom

{

        …

} ;
```

2) Composition (ou agrégation par valeur)

L'agrégation : il s'agit d'une relation entre deux classes, spécifiant que les objets d'une classe sont des composants de l'autre classe. Une relation d'agrégation permet donc de définir des objets composés d'autres objets. L'agrégation permet d'assembler des objets de base, afin de construire des objets plus complexes.

Représentation UML :



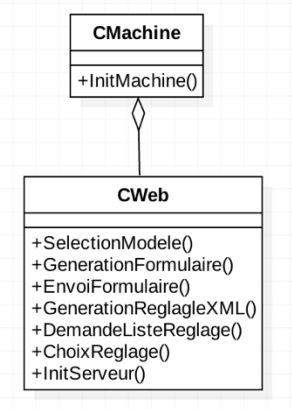
Déclaration en C++ :

```
class CReglage ;

class CMachine

{

        private :

                CReglage reglage ;

                …

} ;
```

Un objet CMachine utilise un objet de la classe CReglage.
Contrainte : la vie d'un objet CReglagle est liée à celle de l'objet CMachine.

3) Agrégation par référence

Représentation UML :



Déclaration en C++ :

```
class CWeb;

class CMachine

{

        private :

                CWeb &navigateur; // ou CWeb *navigateur ;

                …

} ;
```

Un objet CMachine utilise un objet de la classe CWeb.

4) Association

Une association représente une relation sémantique durable entre deux classes.

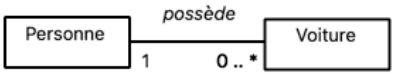
Exemple : une personne peut posséder des voitures. La relation *possède* est une association entre les classes 'Personne' et 'Voiture'.

Attention : même si le verbe qui nomme une association semble privilégier un sens de lecture, une association entre concepts dans un modèle du domaine est par défaut bidirectionnelle. Donc implicitement, l'exemple précédent inclut également le fait qu'une voiture est possédée par une personne.

Comment les représenter ?

Aux deux extrémités d'une association, on doit faire figurer une indication de multiplicité. Elle spécifie sous la forme d'un intervalle d'entiers positifs ou nuls le nombre d'objets qui peuvent participer à une relation avec un objet de l'autre classe dans le cadre d'une association.

Exemple : une personne peut posséder plusieurs voitures (entre zéro et un nombre quelconque). Une voiture est possédée par une seule personne.

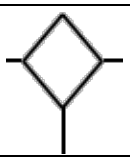
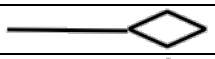


La cardinalité des associations

La cardinalité située à une extrémité d'une association indique à combien d'instances de la classe située à cette extrémité, une instance de la classe située à l'autre extrémité est liée.
Il est possible de spécifier, à une extrémité d'une association, la cardinalité minimale et la cardinalité maximale pour indiquer un intervalle de valeurs auquel doit toujours appartenir la cardinalité.
La syntaxe de spécification des cardinalités minimale et maximale est décrite dans le tableau ci-après.
(En l'absence de spécification explicite des cardinalités minimales et maximales, celles-ci valent 1).

Spécification	Cardinalités
0 .. 1	Zéro ou une fois
1	Une et une seule fois
*	De zéro à plusieurs fois
1 .. *	De une à plusieurs fois
M .. N	Entre M et N fois
N	N fois

VII/ Récapitulatif des relations entre les classes

Type de relation	Représentation
Association binaire	
Association ternaire	
Héritage	
Composition/ Agrégation par valeur	
Agrégation par référence