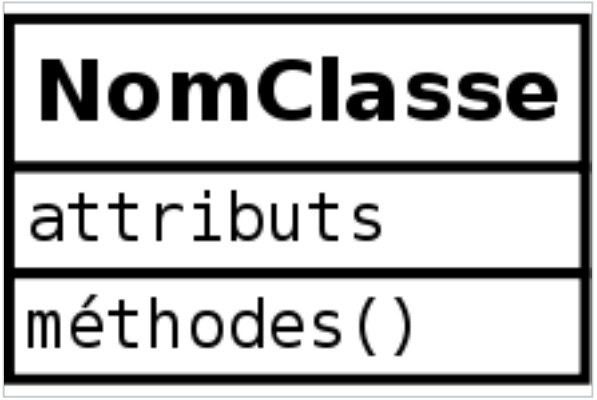
Le diagramme de classe

Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que les différentes relations entre celles-ci. Ce diagramme fait partie de la partie statique d'UML car il fait abstraction des aspects temporels et dynamiques.

Schéma d'une classe



Une classe est représentée par un rectangle séparé en trois parties :

- la première partie contient le nom de la classe
- la seconde contient les attributs de la classe
- la dernière contient les méthodes de la classe

La seconde et la dernière représentent le **comportement** de la classe.

- **Première partie : le nom de la classe**

Modèle d'une classe abstraite.

Modèle d'une classe template.

Il est écrit dans le rectangle du haut.

Dans une classe classique, le nom est écrit en romain (exemple : « ClasseClassique »).

Le nom des classes abstraites est écrit en italique (exemple : « *ClasseAbstraite* »).

Les classes template ont, dans leur angle supérieur droit, un rectangle dont la bordure est en pointillé et qui contient les types des paramètres.

- **Seconde partie : les attributs**

La syntaxe d'un attribut est la suivante :

Visibilité nomAttribut [multiplicité] : typeAttribut = Initialisation ;

Visibilité

La notion de visibilité indique qui peut avoir accès à l'attribut.

Elle ne peut prendre que 4 valeurs possibles :

Caractère	Rôle	Mot clé	Description
+	accès public	public	Toutes les autres classes ont accès à cet attribut.
#	accès protégé	protected	Seules la classe elle-même et les classes filles (héritage) ont accès à cet attribut.
~	accès package	package	Classe visible uniquement dans le package.
-	accès privé	private	Seule la classe elle-même a accès à cet attribut.

Afin de respecter le principe fondamental d'encapsulation, tous les attributs devraient être *privés*.

Pour qu'un attribut privé ou protégé soit récupérable, on utilise en général un *getter* (ou *accesseur*); pour qu'il soit modifiable, on utilise en général un *setter* (ou *mutateur*).

Nom de l'attribut

Il ne doit pas comporter d'espaces, de signes de ponctuation ou d'accents. Pour remplacer les espaces, plusieurs conventions existent : on peut intercaler un symbole _ entre les mots ou utiliser la méthode CamelCase qui consiste à mettre la première lettre de chaque mot en capitale (par exemple, *nom de l'objet* peut devenir : *nom_objet* ou *NomObjet*).

- **Troisième partie : les méthodes**

La syntaxe d'une méthode est la suivante :

Visibilité nomFonction(directionParamètreN nomParamètreN : typeParamètreN) : typeRetour

Visibilité

La notion de visibilité est la même que celle des attributs.

Direction du paramètre

Indique si le paramètre est rentrant (in), s'il est sortant (out) ou s'il est rentrant et sortant (inout).

Exemples de méthode

//méthode publique getAge() retournant un entier

+ getAge() : int

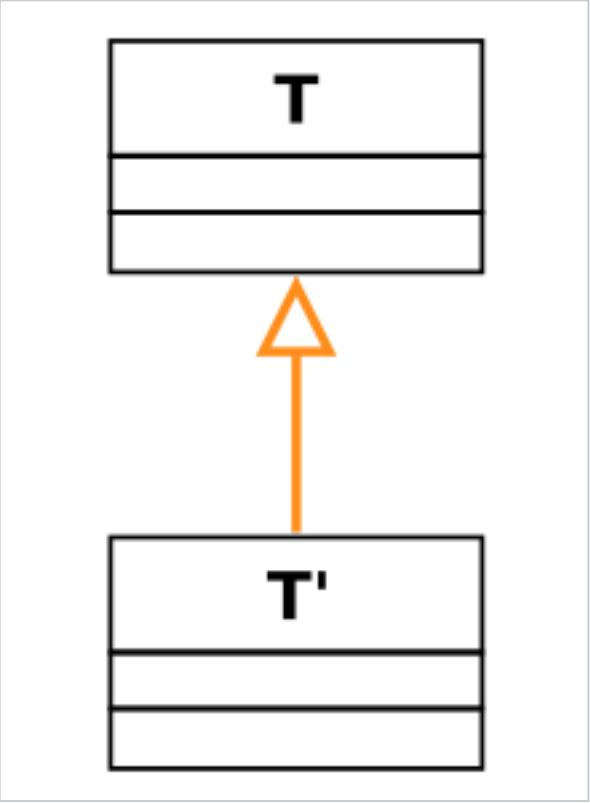
//méthode protégée calculerAge() prenant comme paramètre dateNaissance de type Date et ne retournant rien (void)

calculerAge(in dateNaissance : Date) : void

Relations entre les classes

Ces relations ne sont pas exclusives au diagramme de classes, elles peuvent également s'appliquer à l'ensemble des diagrammes statiques.

Héritage

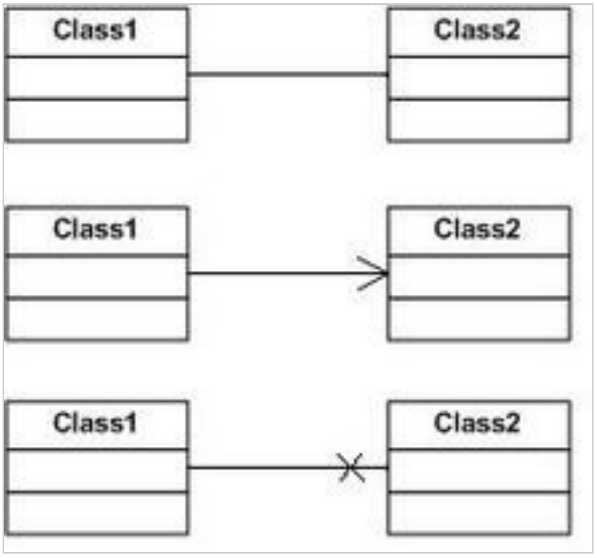


T' hérite de T.

L'héritage est un principe de division par généralisation et spécialisation, représenté par un trait reliant les deux classes et dont l'extrêmité du côté de la classe mère comporte un triangle.

La classe fille hérite de tous les attributs et méthodes, qu'ils soient publics, protégés ou privés. Cependant, elle ne peut pas utiliser directement les attributs et méthodes privés (que ce soit en lecture ou en écriture), sauf par l'intermédiaire d'une méthode héritée (publique ou protégée).

Association



Navigabilité des associations

1- Bidirectionnelle

2- Mono-directionnelle, Invocation de méthode

3- interdit une association.

L'association est une connexion sémantique entre deux classes (relation logique). Une association peut être nommée. L'invocation d'une méthode est une association. Elle peut être binaire, dans ce cas elle est représentée par un simple trait, ou n-aire, les classes sont reliées à un losange par des traits simples. Ces relations peuvent être nommées. L'association n'est utilisée que dans les diagrammes de classe.

multiplicité : comparable aux cardinalités du système Merise, sert à compter le nombre minimum et maximum d'instances de chaque classe dans la relation liant 2 ou plusieurs classes.

navigabilité : indique si on pourra accéder d'une classe à l'autre. Si la relation est entre les classes A et B et que seulement B est navigable, alors on pourra accéder à B à partir de A mais pas réciproquement. Par défaut, la navigabilité est dans les 2 sens.

Une relation structurelle décrit un ensemble de liens. Un lien est une connexion entre objets.

Exemple : Un contrat concerne un client.

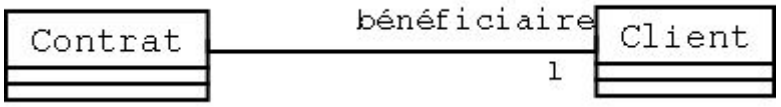
```
class Contrat {
  Client bénéficiaire;
  ...
```

```
}
```

Le bénéficiaire est une des caractéristiques d'un contrat ; la relation est structurelle.

Cette relation est représentée par un **trait plein**, pouvant être orienté. La multiplicité est notée du côté du rôle cible de l'association.

Elle spécifie le nombre d'instances pouvant être associées (liées) avec une seule instance source de la relation. Dans la phrase *un contrat concerne un client*, *contrat* est la source et *client* est la destination.

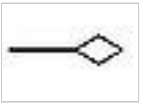


Une multiplicité est exprimée soit par un couple de valeur N..M soit par une seule valeur lorsque N est M sont égaux.

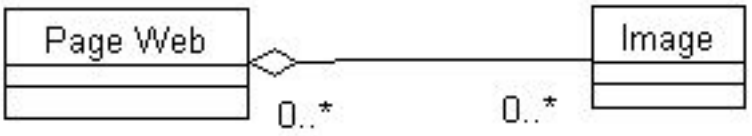
Tableau 1. Multiplicité

Exemple	Interprétation
1..1 ou 1	Un et un seul
0..1	zéro ou un seul
0..* ou *	zéro à plusieurs
3..4	trois à quatre
4	quatre et seulement quatre

Agrégation



.



Commentaires:

- Une page peut contenir des images mais celles-ci peuvent appartenir à d'autres pages.

- la destruction d'une page n'entraîne pas celle de l'image mais seulement la suppression du lien.

Bien sûr nous aurons très souvent une cardinalité 1..1 ou 0..1 côté agrégat.

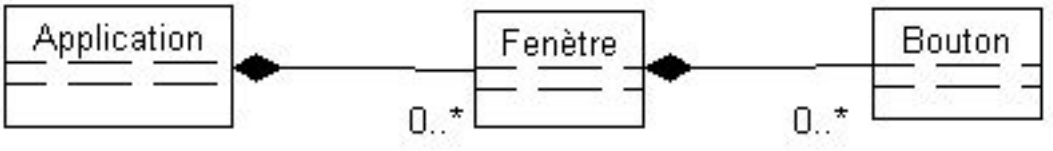
L'appartenance est dite faible car l'agrégé pourra participer à d'autres agrégats et son cycle de vie n'est pas subordonné à celui de son agrégat. Plus haut la disparition d'une configuration n'entraîne pas la disparition des périphériques.

Composition

.

La composition est une agrégation avec cycle de vie dépendant : la classe composée est détruite lorsque la classe mère disparait. L'origine de cette association est représentée par un losange plein.

Il s'agit d'une appartenance forte. La vie de l'objet composant est liée a celle de son composé. La notion de composant est proche de celle d'attribut, si ce n'est que "l'attribut" est "rehaussé" au rang de classe. On parlera de de réification, on reviendra sur cette notion plus loin.



Commentaires:

- La composition se modélise par un losange noir côté composé.

- Une application contient de 0 à n fenêtres qui contiennent de 0 à n boutons.

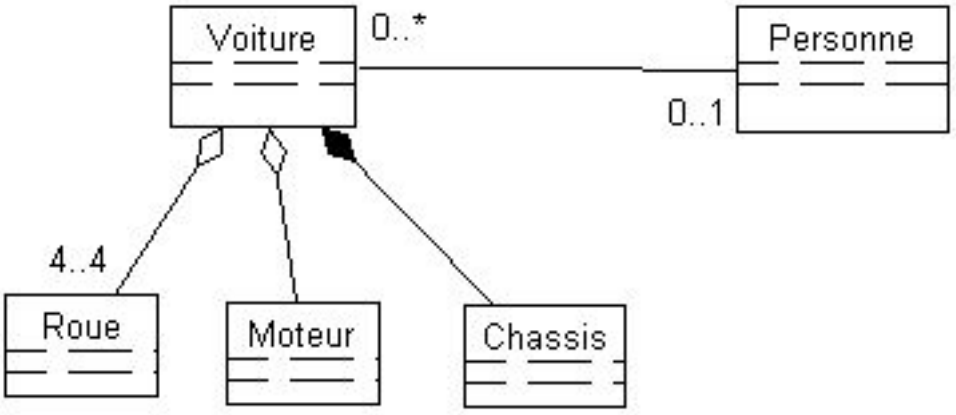
- La fermeture de l'application entraîne la destruction des fenêtres qui entraîne la destruction des boutons.

- la non-présence des valeurs de multiplicités est synonyme de 1..1

Règles :

- Un composant ne peut appartenir à un moment donné qu'à un seul composé.
- La cardinalité ne peut être que de 1 maximum coté composant.
- La suppression du composé entraîne celle du composant.

Exemple récapitulatif.



Commentaires

- Le châssis est un élément indissociable d'une voiture, d'où la composition.

- Le moteur et les roues peuvent être utilisés dans d'autres voitures.

- Notez les valeurs 4..4 qui caractérisent plus précisément les valeurs de multiplicité

- Les absences de cardinalité sont assimilable à 1..1

- L'association entre *Voiture* et *Personne* n'est pas nommée, cela est conseillé lorsque son nom est trivial: "appartient", "concerne" etc ...afin de na pas alourdir le modèle, sans rien apporter à la sémantique.

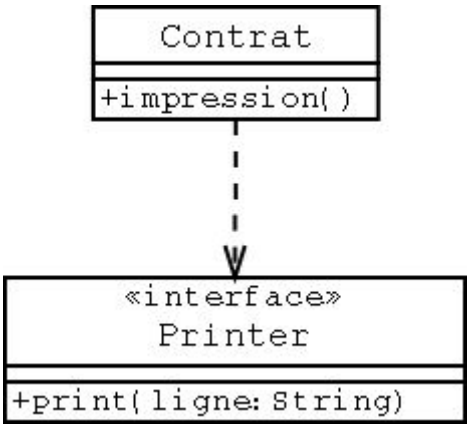
Dépendance

Relation entre éléments du modèle ne nécessitant pas forcément un lien entre objets. Lorsque cette relation est réalisée par des liens entre objets, ces derniers sont limités dans le temps, contrairement à d'autres relations plus structurelles (cas d'une association - voir au-dessus).

Un élément A dépend d'un élément B, lorsque A utilise des services de B.

De ce fait, tout changement dans B peut avoir des répercussions sur A.

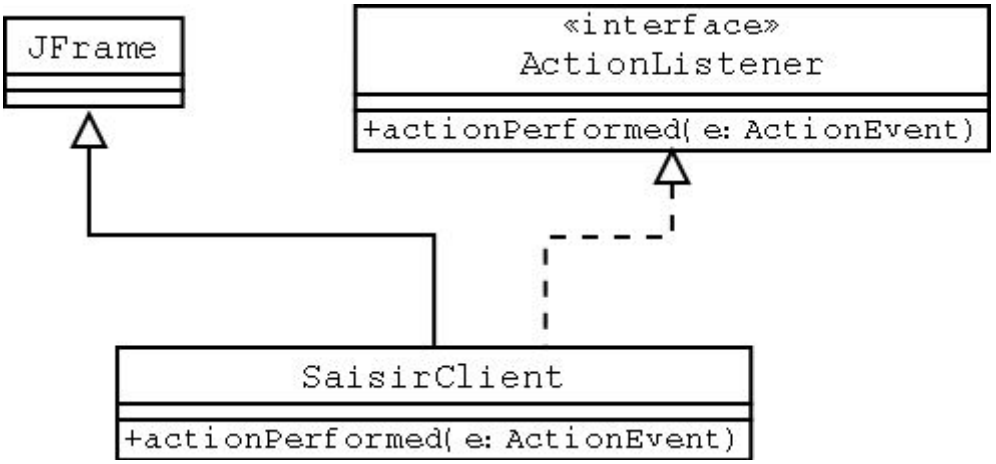
Exemple : Un contrat dispose d'un service d'impression (méthode impression), qui utilise une méthode (print), dont la spécification est déclarée par l'interface Printer.



Relation de réalisation

Relation dans laquelle une interface définit un contrat garanti par une classe d'implémentation.

La fenêtre SaisirClient **réalise** le contrat définit par l'interface ActionListener.
Notation UML : Un **trait discontinu** partant de la classe d'implémentation et allant vers l'interface, se terminant par une **pointe de flèche fermée**, la même utilisée par la spécialisation/généralisation.



Notation simplifée : Une interface peut être représentée par un **petit cercle** avec le nom de l'interface placé juste en dessous. Le cercle peut être attaché à une ou plusieurs classes d'implémentation.