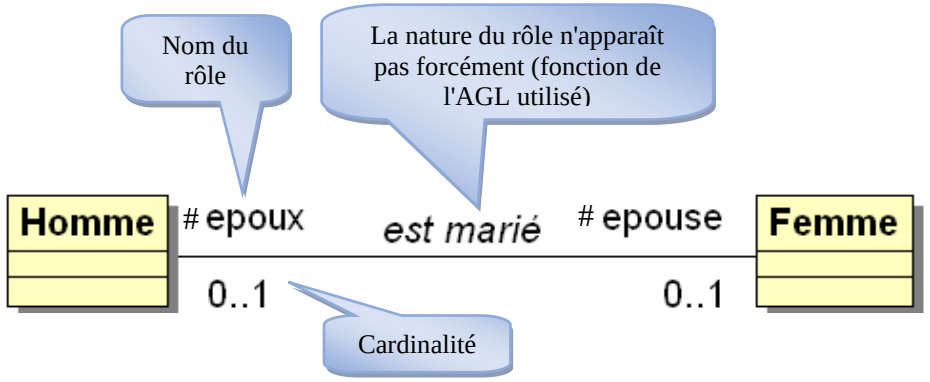
CODAGE DES RELATIONS ENTRE CLASSES EN C++

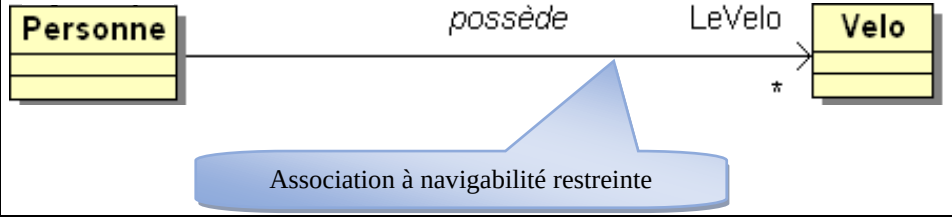
Association

- On utilise l'association quand deux classes sont liées, sans notion de propriété de l'une par rapport à l'autre.
- Les durées de vie ne sont pas forcément liées (la mort d'une instance de la classe A ne provoque pas forcément la mort des objets associés de la classe B)

Association

UML





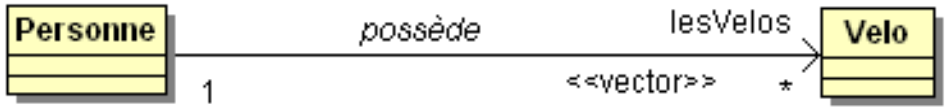
C++

```
class Homme
{
   protected:
      Femme * epouse;
};


class Femme
{
  protected:
     Homme * epoux;
};
```

```
class Personne
{
          Velo ** LeVelo;
};

class Velo
{
       // aucune référence à la classe Personne
};
```

** pour faire référence à un tableau de pointeurs dont le nombre n'est pas connu. Bien sûr, il faut gérer soit même les allocations dynamiques de mémoire.

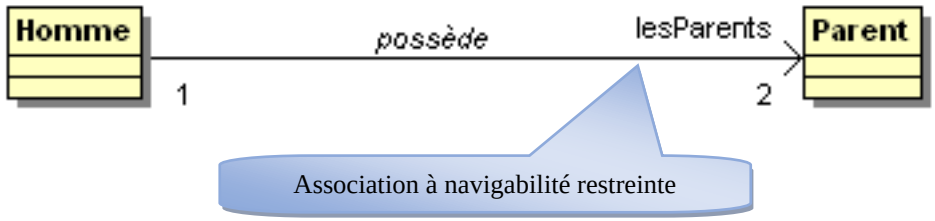
Association




```
#include <vector>
using namespace std;

class Personne
{
  protected:
    vector< Velo *> lesVelos;

};

class Velo
{
};
```

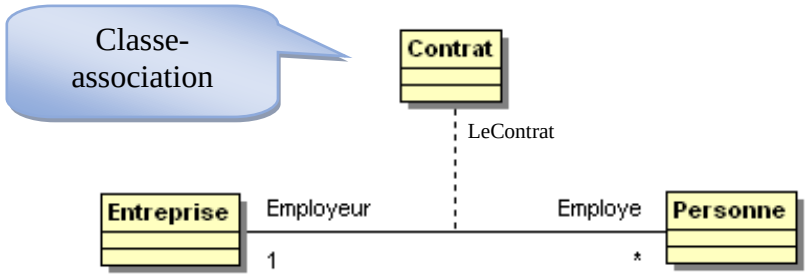
Un tableau de pointeurs sur des Velos (utilisation de la classe vector de la bibliothèque STL).

De cette manière, les durées de vie de la personne et de ses vélos ne sont pas liées.




```
class Homme
{
        Parent * lesParents[2]
};

class Parent
{
      // aucune référence à la classe Homme
};
```

Un tableau de 2 pointeurs représente l'association. De cette manière, les durées de vie entre un homme et ses parents ne sont pas liées




```
class Entreprise
{
      Contrat * LeContrat;
};

class Personne
{
      Contrat * LeContrat;
};

class Contrat
{
      Entreprise *Employeur ;
      Personne ** Employe;
};
```

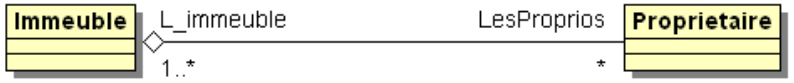
AGREGATION

- On utilise une agrégation quand on peut utiliser le verbe « **comporte** » (Ensemble ---- Sous ensembles)
- L'agrégation est modélisée par un losange vide (du côté de la classe Ensemble)
- Les **durées de vie** entre les classes associées ne sont **pas** forcément **liées** !

Le codage C++ de l'agrégation est le même que pour l'association.

UML est "plus riche" que C++, c'est-à-dire qu'il apporte à travers l'agrégation une notion d'ensemble et sous-ensemble que le C++ n'est pas capable de synthétiser.

Agrégation




```
class Immeuble
{
  protected:
    Proprietaire ** LesProprios;
};

class Proprietaire
{
  protected:
    Immeuble ** L_immeuble;
};
```

** pour faire référence à un tableau de pointeurs dont le nombre n'est pas connu. Bien sûr, il faut gérer soit même les allocations dynamiques de mémoire.

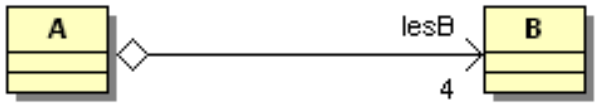
Deux niveaux de pointeurs car on peut être propriétaire de plusieurs immeubles. Si on avait limité à 1 le nombre d'immeubles, on aurait eu un seul niveau de pointeurs

Pour les relations de 1 à plusieurs, les listes ou les vecteurs sont plus simples à coder que le double niveau de pointeurs.


```
class Immeuble
{
  protected:
    list<Proprietaire *> LesProprios;

};

class Proprietaire
{
  protected:
    list<Immeuble *> L_immeuble;
};
```

Pointeurs car AGREGATION (idem si ASSOCIATION)

Stockage des propriétaires et des immeubles dans des listes (vous pouvez aussi utiliser des vecteurs si l'ordre de rangement n'a pas d'importance). Cette solution à base de vecteurs ou de listes est à préconiser quand le nombre de propriétaires ou d'immeubles n'est pas connu à priori



Un objet de la classe A « comporte» 4 objets de la classe B nommés « lesB »

⇨ Les objets « lesB **» ne doivent pas être créés et effacés** dans A.

« lesB » sont déclarés en tant que pointeur, et **le constructeur de A fait référence à 4 pointeurs** définis ailleurs. La destruction d'un objet de la classe A ne provoquera pas la destruction de « lesB ».

A.h

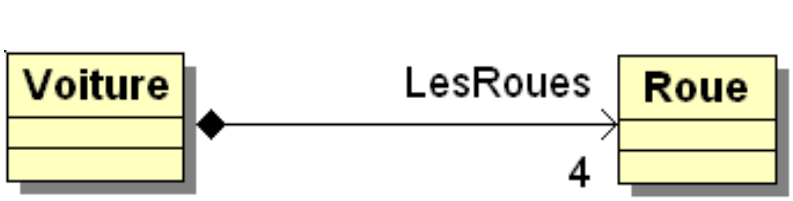
```
class A
{
   B *lesB[4] ;
}
```

A.cpp

```
A :: A( B * refB[4])   // le constructeur
{
   for (int i=0 ; i<4 ; i++) lesB[i] = refB[i] ;
}
```

Composition : agrégation particulière où les parties sont intégrées à l'ensemble

- On utilise une composition quand on peut utiliser le verbe « **est composé de** »
- La composition est modélisée par un losange noirci.
- **Les durées de vie entre les classes associées sont liées (la mort d'un objet de la classe Ensemble provoque la mort des objets composites)**
- **La cardinalité côté de la classe Ensemble (côté losange) est toujours 1**



Une voiture "**est composée** " de 4 roues.

Les durées de vie sont liées :
- On crée les roues en même temps que la voiture.
- Quand la voiture est détruite, les roues aussi !

La composition peut être codée de deux manières **: composition forte** ou **composition applicative** (à privilégier)

Composition forte

Voiture.h	Commentaire

class Roue
{
};
 	
class Voiture
{
protected:
Roue LesRoues[4];
};
 	
Rien ici car navigabilité restreinte
 	
Les roues font complètement partie de la voiture. L'allocation mémoire (statique ici) est faite au sein de la classe Voiture. Quand l'objet de la classe Voiture est détruit, les roues le sont aussi.
 	
Une agrégation aurait été codée :
`Roue * LesRoues[4];`
	Ce type de composition oblige la classe **Roue** à avoir un **constructeur par défaut**.
	
	Toutes les roues sont créées de la même manière.

Composition applicative

Supposons que l'on ne souhaite pas créer les roues de la même manière. Par exemple, supposons que le constructeur de la classe Roue dispose d'un argument qui correspond à la pression de la roue en bars. On pourra comme c'est préconisé avoir une pression des roues avant différente de celle des roues arrière. Dans ce cas-là, la composition forte n'est plus possible ! On va utiliser la **composition applicative**.

Voiture.h	Voiture.cpp

`class Roue`
`{`
`};`
 	
`class Voiture`
`{`
`protected:`
// on déclare 4 pointeurs
`Roue *lesRoues[4];`
`};`
	`Voiture::Voiture( )`
	`{`
	`lesRoues[0]=new Roue(2.3);` // roue avant gauche gonflée à 2,3 bars
	`lesRoues[1]=new Roue(2.3);` // roue avant droite gonflée à 2,3 bars
	`lesRoues[2]=new Roue(2.2);` // roue arrière gauche gonflée à 2,2 bars
	`lesRoues[3]=new Roue(2.2);` // roue arrière droite gonflée à 2,2 bars
	`}`
	
	`Voiture::~Voiture( )`
	`{`
	`for (int i=0 ; i<4 ; i++) delete lesRoues[i];`
	`}`
	
	On crée chaque roue indépendamment.

Généralisation : Cette notion (**héritage**) sera abordée dans un chapitre spécifique.