

Implementació de les relacions

Concepte previ: identificador d'un l'objecte (Id): tota classe amb cardinalitat superior a 1, especialment si és persistent, ha de tenir un atribut (o una combinació d'atributs) que permeti identificar individualment cada un dels objectes (*primary key* o PK). Un valor de Id correspondrà a un objecte i solament a un, i viceversa.

La tendència actual és a treballar amb un Id automàtic, ocult i desconegut per l'usuari, és a dir, transparent. Això vol dir que el sistema ha de controlar que no es produeixin repeticions. Per exemple:

```
long IdPersona // PK de Persona = sense duplicats
```

Introducció: només té interès comentar les relacions de composició i d'associació, ja que les d'herència les ha de resoldre el compilador o si convé el mecanisme de delegació ja comentat, i les de dependència s'implementen amb una simple crida des d'una classe a un mètode d'una altra (només caldrà tenir cura amb la visibilitat, ja que l'objecte que fa la crida ha de poder veure l'objecte cridat).

Relacions de composició: de cardinalitat 1 (en Java)

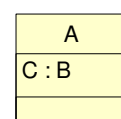
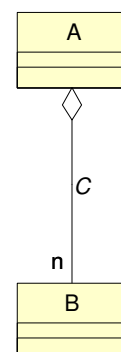
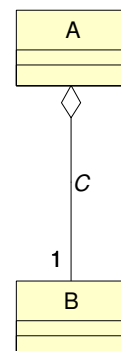
```
class A ...
{
    ...
    B C;
    ...
}
```

De cardinalitat múltiple

```
class A ...
{
    ...
    estHom<B> C;
    ...
}
```

On `estHom` és una *estructura homogènia* com una taula (*array*), llista, etc. on s'emmagatzemen els objectes de la classe B.

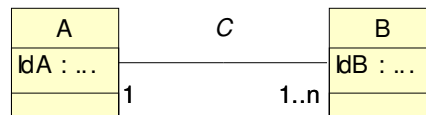
Es pot observar que UML admet que un atribut d'un tipus estructurat es pugui declarar amb una agregació o simplement en el quadre dels atributs (però, mai les dues formes simultàniament). El criteri és fer-ho amb una agregació quan B és una classe nova, creada en el domini del projecte, i en canvi, quan B pertany a una biblioteca de classes, ja sigui estàndard del compilador o sigui del dipòsit (*repository*) de classes pròpies (de l'organització), és més còmode usar aquest segon format.



Relacions d'associació: podeu trobar qui proposa implementar les agregacions i les associacions de la mateixa manera, és a dir, tractar les associacions com si fossin agregacions. Però això comporta un problema en el cas de classes persistents ja que en el moment de guardar-les, si es tractés les associacions de la mateixa manera que les agregacions, s'omplirien els fitxers —o bases de dades— d'informació redundant, o, per ser exactes, duplicada cosa totalment inadmissible. Per aquesta raó s'hauria de convertir l'associació en alguna referència estable a l'objecte associat abans de guardar-la.

La solució normal és guardar com a atribut l'identificador de l'objecte (o PK) en el fitxer —o de la taula de la BD— que implementa la persistència en lloc de l'objecte associat complet. En conseqüència, és millor treballar directament amb el PK ja que així en el moment de gestionar la persistència no cal fer transformacions. (Transformacions que a vegades poden arribar a ser prou penoses en el moment de recuperar la informació dels fitxers).

Així, amb aquesta idea, si tenim:



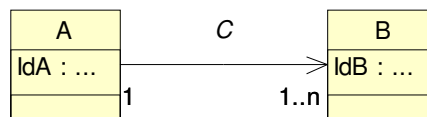
(en Java)

```

class A ...
{
    ...
    tipusIdA IdA;
    ...
    estHom<tipusIdB> C;
    ...
}

class B...
{
    ...
    tipusIdB IdB;
    ...
    tipusIdA C;
    ...
}
  
```

Si fos de navegació limitada només hi hauria l'atribut *C* a la classe des de la qual s'hagués de poder navegar. Per exemple:



Llavors només la classe *A* tindria l'atribut *C*.

(en Java)

```

class A ...
{
    ...
    tipusIdA IdA;
    ...
    estHom<tipusIdB> C;
    ...
}

class B...
{
    ...
    tipusIdB IdB;
    ...
}
  
```