DESARROLLO DE APLICACIONES WEB

Anexo - Unidad 8

Relaciones entre clases

1r DAW

IES La Mola de Novelda

Departament d'informàtica

Professor: Joan Carbonell Picó

Índice

Anexo Unidad 8: Relaciones entre clases

1.- INTRODUCCIÓN

Las relaciones existentes entre las distintas clases nos indican cómo se comunican los objetos de esas clases entre sí. Los mensajes “navegan” por las relaciones existentes entre las distintas clases.

Existen distintos tipos de relaciones:

- Asociación (conexión entre clases).
- Dependencia (relación de uso).
- Generalización/especialización (relaciones de herencia).

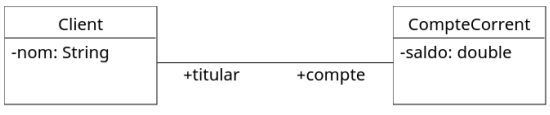
1.1.-ASOCIACIÓN

Una asociación es una relación estructural que describe una conexión entre objetos. Gráficamente, se muestra como una línea continua que une las clases relacionadas entre sí. Sirve para representar una relación débil entre dos clases. Por ejemplo, si destruyo una dirección, el cliente continua existiendo, y al contrario:

1.2.-NAVEGACIÓN DE LAS ASOCIACIONES

Aunque las asociaciones suelen ser bidireccionales (se pueden recorrer en ambos sentidos), en ocasiones es deseable hacerlas unidireccionales (restringir su navegación en un único sentido).

- Ejemplo 1: Asociación bidireccional con multiplicidad 0..1 o 1.




```
public class Client {
        private String nom;
        public CompteCorrent compte;
}
```

```
public class CompteCorrent {
        private double saldo;
        public Client titular;
}
```

Para crear la asociación entre Client y CompteCorrent podemos ver que desde el lado de la

clase Client se asocia con la clase CompteCorrent con el rol +compte. En este caso un cliente tiene una cuenta corriente. Este rol se convertirá en un nuevo atributo de la clase Client. Lo mismo haremos con la dirección de la clase CompteCorrent a la clase Client. El rol también me indica su visibilidad.

- Ejemplo 2: Asociación unidireccional (direccional) con multiplicidad 0..1 o 1.


```
public class Usuari {
        private String nom;
        public Clau clau;
}
```

```
public class Clau {
        private int codi;
}
```

En este caso un usuario tiene una clave pero no se puede asociar de clave a usuario. Por la asociación de Usuari con la clase Clau, el rol +clau se implementará como un atributo de la clase Usuari con la visibilidad indicada en el rol. En este caso una instancia de la clase Usuari se le asocia con una única instancia de la clase Clau. Aqui vemos que la asociación es de 1 a 1.

- Ejemplo 3: Asociación bidireccional con multiplicidad *.


```
public class Persona {
        private String nom;
        public Gos[] mascotes;
}
```

```
public class Gos {
        private String nom;
        public Persona propietari;
}
```

Podemos ver que la multiplicidad nos indica la cantidad de objetos que de una determinada clase se asocian con cierta cantidad de objetos de otra clase. En este caso se nos indica que una persona puede tener o ninguno o muchos perros (o mascotas como dice el rol). La multiplicidad * es la misma que 0..*. Un perro tiene un determinado propietario que es una persona. En el ejemplo, como aún estamos utilizando estructuras estáticas, tenemos que indicar el espacio que reservamos en el array en el constructor de Persona. En este caso, indicaremos el número de perros que va tener esa persona, por ejemplo 4. Si utilizamos una estructura dinámica, el número de perros será indeterminado .

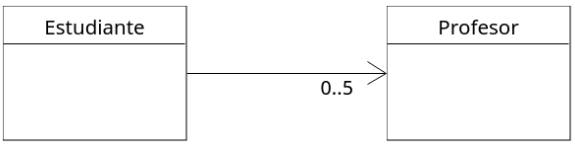
1.3.-MULTIPLICIDAD

La multiplicidad de una asociación determina cuántos objetos de cada tipo intervienen en la relación, es decir, el número de instancias de una clase que se relacionan con una instancia de la otra clase. Hay que tener en cuenta que:

1. Cada asociación tiene dos multiplicidades (una para cada extremo de la relación).
2. Para especificar la multiplicidad de una asociación hay que indicar la multiplicidad mínima y la multiplicidad máxima (mínima..máxima)
3. Cuando la multiplicidad mínima es 0, la relación es opcional.
4. Una multiplicidad mínima mayor o igual que 1 establece una relación obligatoria.

Multiplicidad	Significado
1	Uno y sólo uno
0..1	Cero o uno
N..M	Desde N hasta M
*	Cero o varios
0..*	Cero o varios
1..*	Uno o varios (al menos uno)

1.4.-IMPLEMENTA EN JAVA LA SIGUIENTE ASOCIACIÓN.

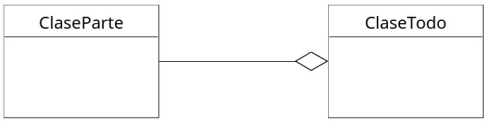


1.5.-CASOS PARTICULARES.

Estos casos particulares de asociaciones indican la relación entre un todo y sus partes:

1.5.1.- AGREGACIÓN (Usa).

Una agregación representa la relación “ es parte de” entre objetos. La agregación se indica con un rombo hueco en el extremo de la línea de asociación correspondiente al conjunto o agregado. Su notación es la siguiente:



En una agregación, una ClaseTodo tiene una relación con una o varias ClaseParte. La ClaseParte puede existir independientemente de la ClaseTodo (la que tiene el rombo vacio). Si destruyo la ClaseTodo, la ClaseParte continua existiendo.

Es una relación similar a la composición pero menos fuerte.

- Ejemplo 1: Una habitación se compone (es un agregado) de una o más paredes. Una pared es parte de una o más habitaciones. Si desaparece la habitación, las paredes siguen teniendo sentido.

- Ejemplo 2:

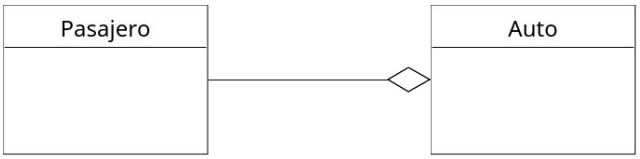

```
public class Cotxe{
        private Estereo estereo;
        void ensamblar(Estereo est){
                estereo = est;
        }
}
```

```
public class Estereo {

}
```

Para vincular un objeto de la clase Estereo a una instancia de la clase Cotxe, ésta invocará al método ensamblar para que forme parte de esa instancia de Cotxe. De esa manera vincularemos ese objeto Estereo a un objeto Cotxe

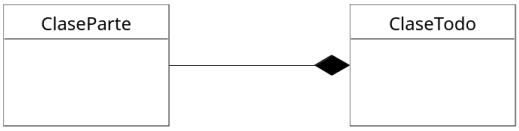
- Ejemplo 3: Implementar en Java la siguiente agregación.



1.5.2.- COMPOSICIÓN (Posee).

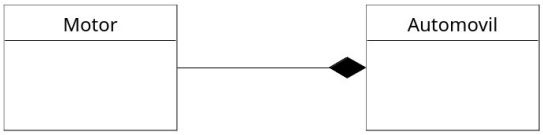
Una composición es un tipo de agregación que trae consigo una estrecha relación entre

el agregado y sus componentes. Los componentes sólo tienen sentido como parte del compuesto. Cada componente solo puede pertenecer a un todo. En este caso, las partes sólo existen asociadas al compuesto (sólo se accede a ellas a través del compuesto). Su notación es la siguiente:



En este caso, la ClaseTodo es la responsable de crear y de destruir la ClaseParte y la ClaseParte no puede existir sin la ClaseTodo. Una instancia de la ClaseTodo puede tener una o más instancias de la ClaseParte. Si destruyo la instancia de la ClaseTodo también se destruyen las instancias de la ClaseParte que tuviera esa instancia de la ClaseTodo.

- Ejemplo 1: Implementar en Java la siguiente composición.



2.- DEPENDENCIA

Una dependencia es una relación de uso entre dos entidades, una usa a la otra. Es el tipo de relación más débil y básica entre clases. Se representa con una línea punteada y una flecha <---, que indica que clase depende de cual.

A ----> B

- Aquí, la clase A depende de la clase B.
- La clase A usa la clase B.
- La clase A conoce la existencia de la clase B, pero la clase B no conoce la existencia de la clase A, (uso de la flecha).
- Todo cambio que se haga en la clase B, por la relación que hay con la clase A, podrá afectar a la clase A.

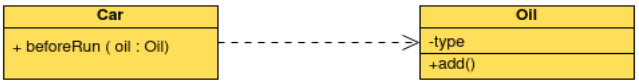
También se le conoce como relación de uso, porque la clase A usa a la clase B.

- Ejemplo 1: Resolución de una ecuación de segundo grado

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Para resolver una ecuación de segundo grado hemos de recurrir a la función `sqrt` de la clase `Math` para calcular una raíz cuadrada.

- Ejemplo 2: Una relación de dependencia es una relación de “uso”. Un cambio en una cosa en particular puede afectar a otras cosas que la usan, y usar una dependencia cuando es necesario indica que una cosa usa otra. En este ejemplo vemos que el auto depende de la gasolina. Si no hay gasolina, el automóvil no podrá conducir.



3.- RESUMEN DE LA RELACIÓN ENTRE CLASES Y OBJETOS