Tema 3

Relaciones entre clases

Índice de contenidos

1. Introducción

Hasta ahora hemos abordado los conceptos de la POO centrándonos en el concepto de clase de forma aislada. Cuando nos enfrentamos al diseño de un sistema complejo lo normal es que aparezcan numerosos conceptos o abstracciones relacionadas entre sí. Una relación entre clases representa una conexión semántica entre objetos que son instancia de esa clase y esta relación podrá tener distintas propiedades.

En este tema veremos cómo abstraer y modelar las relaciones semánticas entre clases, viendo los distintos tipos de relaciones existentes y estudiando cada una de ellas. Como notación emplearemos UML (anexo I) que es un lenguaje de representación independiente del lenguaje de programación que usemos (C++ o Java en nuestro caso).

2. Relaciones /Asociaciones entre clases

Un aspecto muy importante del proceso de diseño es definir las relaciones que se tienen que establecer entre las clases que representan el modelo del dominio del problema que pretendemos resolver de la forma más completa posible. La idea es que las clases(sus objetos) colaboran entre sí para realizar algún tipo de proceso o funcionalidad que es preciso llevar a acbo.

Para representar las clases y sus relaciones utilizaremos como hemos mencionado antes un **diagrama de clases en UML (Unified Modeling Language)**. Un diagrama de clases representa las clases y las asociaciones existentes entre ellas y las relaciones que existirán entre los objetos . Las relaciones entre clases también se denominan *asociaciones*.

2.1. Asociaciones entre clases

Dadas dos clases A y B una asociación representa **algún tipo de relación** existente entre dichas clases en el dominio del problema. La asociación se representa en UML con un segmento que une las dos clases (ver Figura 1).

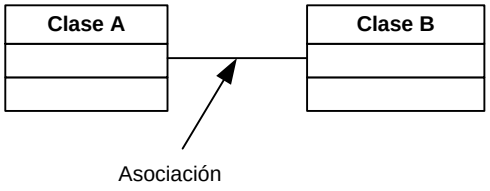
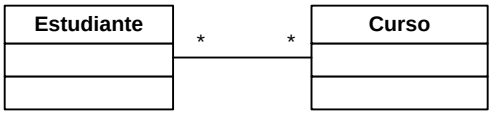


Figura 1 Asociación entre clases

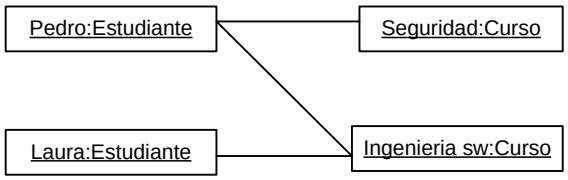
La asociación representa una relación estructural entre objetos normalmente de clases diferentes y **es la más común** de las relaciones. Aunque es frecuente que la mayoría de las asociaciones sean binarias, pueden existir relaciones ternarias o n-arias.

Las conexiones que describe una asociación dan lugar a interacciones entre los objetos que comúnmente se denomina colaboración. En esencia, una asociación es una relación entre clases que indica cómo se pueden enlazar juntas las instancias de las clases, además de que estas van a colaborar para desarrollar alguna funcionalidad.

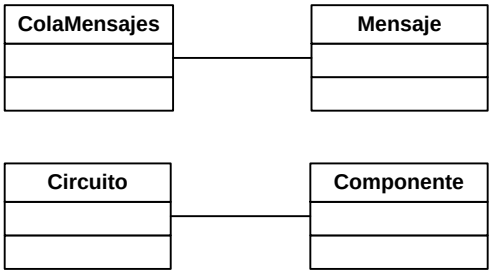
Por ejemplo, dado el siguiente diagrama de clases



Una posible instancia de estos objetos sería la siguiente:



Otros ejemplos de asociaciones entre clases:



Al observar estos ejemplos surgen las siguientes cuestiones. ¿Un objeto de tipo Mensaje con cuantos objetos de tipo ColaMensajes está relacionado y viceversa? O de la misma forma ¿Con cuántos objetos de tipo Circuitos puede estar relacionado un objeto de tipo Componente?.

Al plantearnos las relaciones entre clases surgen cuestiones como la cardinalidad/multiplicidad y la direccionalidad de la relación (sentido de la relación) ,conceptos estos que proporcionan información adicional acerca de la asociación. Más adelante veremos estos importantes conceptos en detalle.

Implementación de la asociación entre clases

Dado el siguiente diagrama de clases (Figura 2

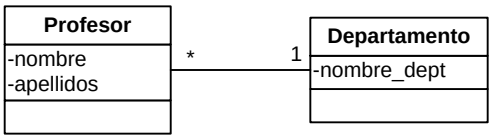


Figura 2 Diagrama de clases

Vamos a ver como se implementaría en un lenguaje de programación Java :

```
public class Profesor {
    String nombre;
    String apellidos;
    Departamento dept;
    Profesor (){}
}
```

```
public class Departamento {
 String nombre_dept;
 List empleados;

 Departamento(){}
 public void setProfesor(Profesor p){
 empleados.add(p);
 }
}
```

2.2. Cardinalidad/Multiplicidad

Dado que una asociación representa una interconexión entre instancias de objetos de dos clases, se podría indicar, cuántos objetos de cada clase pueden estar involucrados en la asociación. Para ello se debe definir la **cardinalidad** (o multiplicidad) de la asociación.

Cardinalidad/Multiplicidad: es el número de objetos de un extremo de la asociación que están enlazados con un objeto del otro extremo. Por ejemplo, un objeto de tipo Departamento puede emplear (relación) a muchos objetos de tipo *Empleado*. La multiplicidad de la relación de *Departamento* con *Empleado* es "una a muchos": 1 .. *.

Podemos indicar el valor máximo y mínimo de la cardialidad , la cardinalidad máxima y mínima se ponen encima o debajo de la línea que representa la asociación y, dependiendo de los valores, tienen un significado u otro.

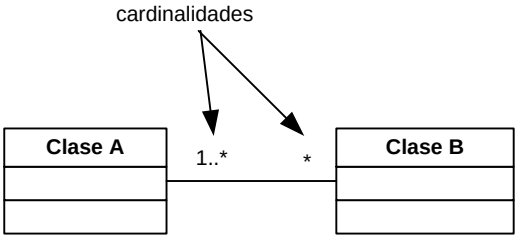


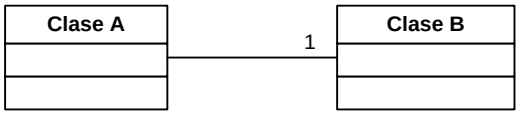
Figura 3 Representación de cardinalidades

La **cardinalidad es una de restricción** introducida en el modelo para representar más fielmente la realidad (Ejemplo un estudiante sólo puede estar matriculado en un único

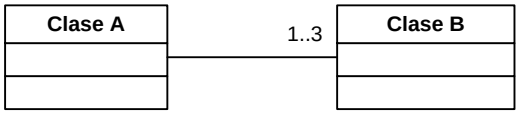
curso). Esto **tendrá su implicación en la implementación** de la asociación en el lenguaje de programación.

La cardinalidad se puede expresar mediante diferentes notaciones:

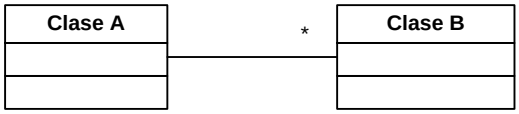
- **Número exacto (1, 3, etc.).** Expresa que un objeto de la clase Clase A tiene que estar relacionado obligatoriamente con el número de objetos que indica la cardinalidad de la clase Clase B .



- **Rango de valores** (0..1, 3..5, etc.). Con esta notación, indicamos el número mínimo y máximo de objetos con los que puede estar relacionado un objeto en el otro extremo de la relación. En el ejemplo un objeto instancia de la ClaseA sólo podrá estar relacionado como mucho con 3 instancias de la clase B y como mínimo lo estará con una.

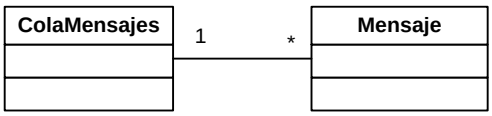


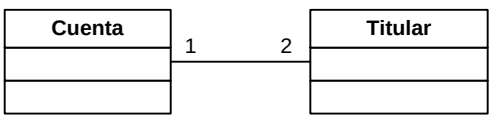
- **Muchos (*)**.Indica que la instancia de la clase Clase A puede estar relacionada con cualquier número de instancias de la clase Clase B (o incluso con ninguna) equivale a escribir 0..*.



- Combinaciones de las anteriores.

A continuación (Figura 4) se muestran algunos ejemplos de cardinalidades.





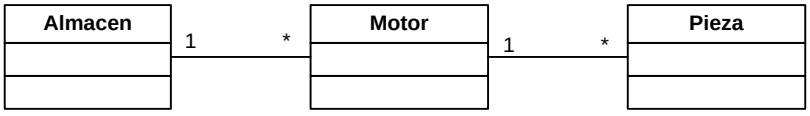


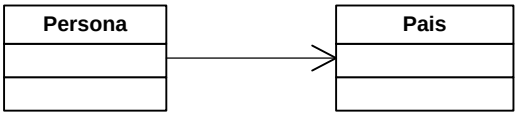
Figura 4 Ejemplo de multiplicidades/cardinalidades

2.3. Navegabilidad de las asociaciones

Otra característica (o restricción) que podemos expresar en una relación de clases dentro de un diagrama UML es la navegabilidad de la relación. Esta **restricción** indica **cómo se debe** interpretar la asociación. La navegación indica el sentido en el cuál es posible recorrer la asociación.

Las dos opciones o tipos de navegabilidad posibles son las siguientes:

Asociación unidireccional: sólo una de las dos clases tiene constancia de la existencia de la otra. Y consecuentemente sólo un objeto tendrá "constancia del otro".



El sentido semántico de esta restricción es el de que sólo un tipo de objeto participante en la relación va a interpretar la relación , en el otro no es necesario.

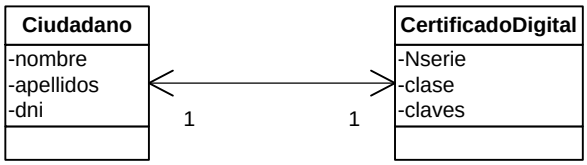
Implementación en Java

```
public class Persona {
    String nombre;
    String apellidos;
    Departamento dept;
    Pais pais;
    Persona(){};
}
```

```
public class Pais {
 String nombre;
 int población;
 String religion_mayoritaria;
}
```

Como se puede observar en la implementación anterior, desde un objeto de tipo *Persona* se tendrá acceso a uno de tipo *País*, pero no al revés.

Asociación Bidireccional: los dos tipos de objetos pueden interpretar la relación existente.



Ejercicio. Plantee como seria la implementación en Java de esta asociación bidireccional

2.4. Tipos de asociaciones entre clases

2.4.1. Asociaciones reflexivas

Las asociaciones reflexivas son un tipo de asociación binaria, en la que las dos clases origen y destino son del mismo tipo. Como muestra este tipo de relación, no hay ninguna restricción que impida que una instancia de una clase se relacione con otras instancias de la misma clase.

Se representan con una relación que entra y sale de la misma clase .Para poder identificar correctamente los roles de los dos extremos de la asociación, hay que definirlos con un texto aclaratorio.

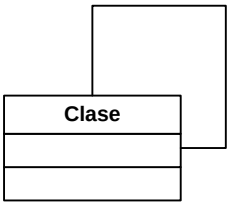


Figura 5 . Asociación reflexiva

Un ejemplo de esta asociación seria una clase que representa un directorio en un sistema de archivos (Figura 6).

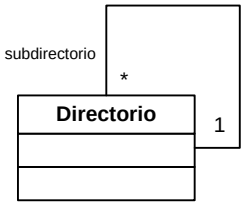


Figura 6.Asociación Reflexiva

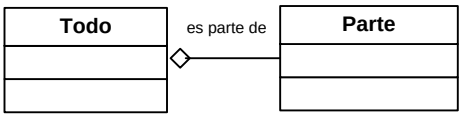
Implementación en C++

La implementación de una clase reflexiva del ejemplo anterior sería de la siguiente forma

```
class Directorio;

class Directorio
{
    char *nombre;
    Directorio raiz;
    List<*Directorio> subdirectorios;

       public:

              Directorio();
              ~Directorio();
};
```

2.4.2. Asociación de agregación compartida

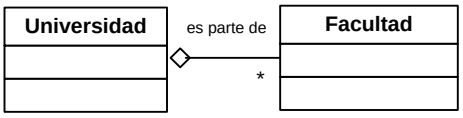
Las asociación de agregación modela la relación semántica "todo/parte" o bien " x tiene un y" o " y es parte de x". A continuación podemos el esquema general de este tipo de asociación.



El rol "Parte" puede pertenecer a más de un agregado (de ahí el nombre de agregación compartida. La existencia del "Todo" está supeditada a la de las *Partes*. y la destrucción del "Todo" no implica la destrucción de las *Partes*.

> Ejemplo : un *Equipo de trabajo* se compone de diferentes *Personas.* Una misma Persona puede ser miembro de más de un equipo de trabajo.

Otro ejemplo podría ser la relación de una Universidad con sus distintas Facultades como se muestra en el siguiente diagrama:



La relación anterior lleva implícito el significado que si una de las partes desaparece la parte *Todo* continua existiendo, de ahí que a este **tipo de agregación se le denomina débil**. En el ejemplo una Facultad podría desaparecer y la Universidad seguiría existiendo.

2.4.3. Agregación de composición

Una **agregación de composición** es aquella que posee *(<<contiene>>)* a sus partes y entraña una **fuerte dependencia de propiedad sobre ellas**. Se dice entonces que la **agregación** es **fuerte** lo que constituye la principal diferencia con la agregación anterior.

La característica de **agregación fuerte** confiere a la asociación la característica de que el *Todo* no puede existir sin las partes ni viceversa. El ciclo de vida de las *Partes* está estrechamente ligado al del *Todo*, de forma que si este se destruye se destruyen también las *Partes*. La cardinalidad en el lado T*odo* tiene que ser uno, mientras que en el lado *Parte* puede ser un intervalo.

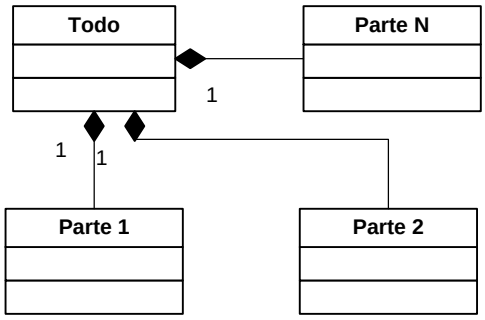


Figura 7 Relación de composición

A diferencia de la agregación simple la agregación de composición requiere que las partes no puedan tener una relación de agregación con a otra clase (no pueden estar compartidas).

Ejemplos de agregación de composición pueden ser los siguientes diagramas de clases:

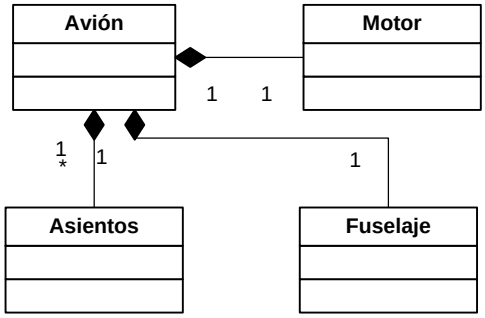


Figura 8 Ejemplos de asociación por composición

2.4.4. La clase asociación

En muchas ocasiones en el diseño, surge la necesidad de representar explícitamente una relación entre clases como otra clase más debido a su relevancia y por que la propia relación tiene características que deben ser reflejadas.

Una **clase asociación** es una clase que **representa una relación entre clases** y tiene la suficiente relevancia en el problema para tener su propia representación. Es decir la clase asociación nos permite caracterizar la propia relación existente entre clases.

Para ver esto considere el siguiente ejemplo (Figura 9):

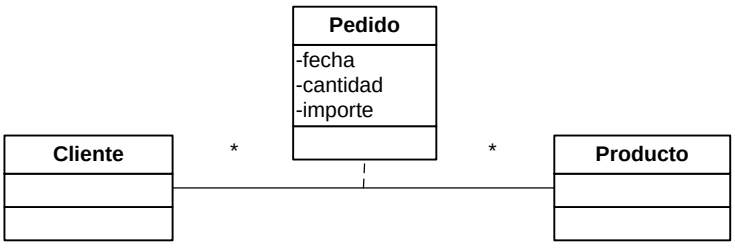


Figura 9 . Representación de una clase asociación

En este ejemplo la asociación entre la clase *Cliente y Producto* tiene tanta relevancia en el dominio que se opta por modelarla como una clase más. Esto tiene la ventaja de que se pueden especificar atributos y métodos que describen la propia relación.