Asociación Agregación Composición

Introducción

Además de la relación de herencia las clases empleadas dentro de una aplicación Java, los objetos pueden estar conectados dentro de un programa con otros tipos de relaciones.
Estas relaciones puede ser persistentes si establecen si la comunicación entre objetos se registra de algún modo y por tanto puede ser utilizada en cualquier momento.
En el caso de relaciones no persistentes entonces el vínculo entre objetos desaparece tras ser empleado. En cualquier caso, en la mayor parte de los casos la resolución de un problema más o menos complejo exige la colaboración entre objetos.
Esta colaboración se puede llevar a cabo mediante el establecimiento de relaciones entre clases (relación de herencia o generalización) o entre instancias (relación de asociación y relación todo-parte: agregación y composición).

Asociación

En una asociación, dos instancias A y B relacionadas entre sí existen de forma independiente. No hay una relación fuerte. La creación o desaparición de uno de ellos implica únicamente la creación o destrucción de la relación entre ellos y nunca la creación o destrucción del otro. Por ejemplo, un cliente puede tener varios pedidos de compra o ninguno.

La relación de **asociación** expresa una relación (unidireccional o bidireccional) entre las instancias a partir de las clases conectadas. El sentido en que se recorre la asociación se denomina **navegabilidad** de la asociación. Cada extremo de la asociación se caracteriza por el **rol** o papel que juega en dicha relación el objeto situado en cada extremo. La **cardinalidad** o multiplicidad es el número mínimo y máximo de instancias que pueden relacionarse con la otra instancia del extremo opuesto de la

relación. Por defecto es 1. El formato en el que se específica es (mínima..máxima). Por ejemplo:

- 1: Uno y sólo uno (por defecto)
- 0..1: Cero a uno. También (0,1)
- M..N: Desde M hasta N (enteros naturales)
- 0..*: Cero a muchos
- 1..*: Uno a muchos (al menos uno)
- 1,5,9: Uno o cinco o nueve

La asociación se podría definir como el momento en que dos objetos se unen para trabajar juntos y así, alcanzar una meta.

Un punto a tomar muy en cuenta es que ambos objetos son independientes entre sí, veremos un poco más adelante qué implicación tiene esto. Para validar la asociación, la frase *"Usa un"*, debe tener sentido:

- **El ingeniero** *usa* **una computadora**
- **El cliente** *usa* **tarjeta de crédito**.

Cómo implementar *Asociación*

Representaremos la relación: **El cliente** *usa* **tarjeta de crédito**.

Código :

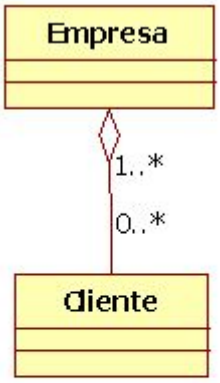
```
public class Customer {

    private int id;
    private String firstName;
    private String lastName;
    private CreditCard creditCard;

    public Customer() {
        //Lo que sea que el construtor haga
    }

    public void setCreditCard(CreditCard creditCard) {
        this.creditCard = creditCard;
    }

    // Más código aquí
}
```

Agregación

La agregación es un tipo de asociación que indica que **una clase es parte de otra clase** (composición débil). Los componentes pueden ser compartidos por varios compuestos (de la misma asociación de agregación o de varias asociaciones de agregación distintas). **La destrucción del compuesto no conlleva la destrucción de los componentes.** Habitualmente se da con mayor frecuencia que la composición.

La agregación se representa en UML mediante un diamante de color blanco colocado en el extremo en el que está la clase que representa el "todo".

Veamos un ejemplo de agregación:



- Tenemos una clase Empresa.
- Tenemos una clase Cliente.
- Una empresa agrupa a varios clientes.

```
public class Cliente {
      …..
}

public class Empresa{
      ...
   ArrayList<Cliente> listaClientes = new ArrayList<Cliente>();
      ...

}
```

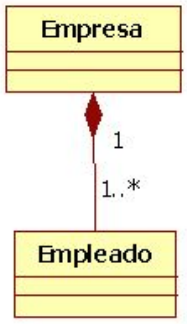
Composición

Composición es una forma fuerte de composición donde la vida de la clase contenida debe coincidir con la vida de la clase contenedor. Los componentes constituyen una parte del objeto compuesto. De esta forma, los componentes no pueden ser compartidos por varios objetos compuestos. La supresión del objeto compuesto conlleva la supresión de los componentes. En esta situación, la frase "*Tiene un*", debe tener sentido:

- **El auto** *tiene* **llantas**
- **La portátil** *tiene* **un teclado**.

El símbolo de composición es un diamante de color negro colocado en el extremo en el que está la clase que representa el "todo" (Compuesto).

Veamos un ejemplo de composición:



• Tenemos una clase Empresa.
• Un objeto Empresa está a su vez compuesto por uno o varios objetos del tipo empleado.
• El tiempo de vida de los objetos Empleado depende del tiempo de vida de Empresa, ya que si no existe una Empresa no pueden existir sus empleados.

La codificación del Diagrama de Clases del ejemplo anterior en Java podría corresponderse con el siguiente código:

```
public class Empleado{
    ...
}
public class Empresa{
    ...
    Empleado[] empleado1= new Empleado[2];
    ...
    Empresa() {
        empleado1[0] = new Empleado();
        empleado1[1] = new Empleado();
        ...
    }
}
```

Diferencias entre Composición y Agregación

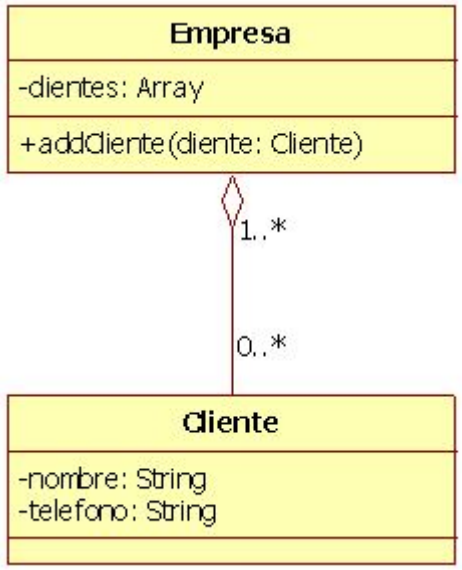
La siguiente tabla intenta resumir algunas diferencias entre agregación y composición.

	Agregación	**Composición**
Varias asociaciones comparten los componentes	Sí	No
Destrucción de los componentes al destruir el compuesto	No	Sí
Cardinalidad a nivel de compuesto	Cualquiera	0..1 ó 1
Representación	Rombo transparente	Rombo negro

Y en código…

Para traducir ambas relaciones a código, podemos utilizar un atributo en la clase contenedora o compuesta donde almacenaremos una colección de los objetos que la componen, y por otro lado declararemos un método para agregar elementos a la colección. Dependiendo del lenguaje de programación empleado, podemos utilizar diferentes estructuras de datos que nos permitan almacenar esa colección de objetos, aunque generalmente se utilizan arrays (arreglos) para este fin.

Veamos un ejemplo:



Como podemos apreciar, es tan simple como crear en la clase Empresa un atributo clientes (coleccion de clientes) que sea un array, luego creamos un método addCliente donde recibiremos objetos de tipo Cliente y los agregaremos dentro del array.

Resumen:

En líneas generales, como hemos visto, se podría decir que **la diferencia entre agregación y composición es conceptual**, no se diferencia por código, o al menos, en el mayor de los casos y en la mayoría de los lenguajes de programación (como Java o PHP). De todas maneras, en el caso de la composición, si quisiéramos ser más estrictos con los diagramas de clases modelados con UML, deberíamos destruir de alguna manera el objeto componente (*empleado*) una vez que se desasociaran del objeto compuesto (*empresa*).

En definitiva, UML nos permite la posibilidad de diferenciar este tipo de asociaciones con el fin de que, aquella persona que le interese, pueda estipular de una u otra manera que se trata de una composición o una agregación, aunque en términos de implementación no se diferencie tan apenas su uso ni tenga tanta relevancia. Pero una vez más, UML propone muchas posibilidades y debe ser el analista y/o desarrollador quien decida y haga un uso correcto del mismo, con el fin de visualizar, especificar, construir y documentar adecuadamente los artefactos (modelos) de un sistema software.

¿Asociación o Composición? … depende

Habrá casos en que será difícil determinar qué tipo de relación usar cuando ambas encajan:

- **Un reloj** *tiene* **manecillas**

- **Un reloj** *usa* **manecillas (Para dar la hora, claro)**.

Así que debes tomar en cuenta qué tanta flexibilidad te daría implementar una u otra.

Desde el punto de vista de fabricante de relojes, necesito tener control sobre cada una de las piezas que conforman mis relojes; así, si alguna pieza sale defectuosa, puedo reemplazarla antes que mi producto llegue al mercado. Me conviene la asociación.