7.A. Relación entre clases.

1. Relaciones entre clases.

Cuando estudiaste el concepto de clase, ésta fue descrita como una especie de mecanismo de definición (plantillas), en el que se basaría el entorno de ejecución a la hora de construir un objeto: un **mecanismo de definición de objetos**.

Por tanto, a la hora de diseñar un conjunto de clases para modelar el conjunto de información cuyo tratamiento se desea automatizar, es importante establecer apropiadamente las posibles relaciones que puedan existir entre unas clases y otras.

En algunos casos es posible que no exista relación alguna entre unas clases y otras, pero lo más habitual es que sí exista: una clase puede ser una **especialización** de otra, o bien una **generalización**, o una clase contiene en su interior objetos de otra, o una clase utiliza a otra, etc. Es decir, que entre unas clases y otras habrá que definir cuál es su relación (si es que existe alguna).

Se pueden distinguir diversos tipos de relaciones entre clases:

- **Clientela**. Cuando una clase utiliza objetos de otra clase (por ejemplo al pasarlos como parámetros a través de un método).

- **Composición**. Cuando alguno de los atributos de una clase es un objeto de otra clase.

- **Anidamiento**. Cuando se definen clases en el interior de otra clase.

- **Herencia**. Cuando una clase comparte determinadas características con otra (clase base), añadiéndole alguna funcionalidad específica (especialización).

La relación de **clientela** la llevas utilizando desde que has empezado a programar en Java, pues desde tu clase principal (clase con método `main`) has estado declarando, creando y utilizando objetos de otras clases. Por ejemplo: si utilizas un objeto `String` dentro de la clase principal de tu programa, éste será cliente de la clase `String` (como sucederá con prácticamente cualquier programa que se escriba en Java). Es la relación fundamental y más habitual entre clases (la utilización de unas clases por parte de otras) y, por supuesto, la que más vas a utilizar tú también, de hecho, ya la has estado utilizando y lo seguirás haciendo.

La relación de **composición** es posible que ya la hayas tenido en cuenta si has definido clases que contenían (tenían como atributos) otros objetos en su interior, lo cual es bastante habitual. Por ejemplo, si escribes una clase donde alguno de sus atributos es un objeto de tipo `String`, ya se está produciendo una relación de tipo **composición** (tu clase "tiene" un `String`, es decir, está compuesta por un objeto `String` y por algunos elementos más).

La relación de **anidamiento** (o **anidación**) es quizá menos habitual, pues implica declarar unas clases dentro de otras (**clases internas** o **anidadas**). En algunos casos puede resultar útil para tener un nivel más de **encapsulamiento** y **ocultación** de información.

En el caso de la relación de **herencia** también la has visto ya, pues seguro que has utilizado unas clases que derivaban de otras, sobre todo, en el caso de los objetos que forman parte de las **interfaces gráficas**. Lo más probable es que hayas tenido que declarar clases que derivaban de algún componente gráfico (`JFrame`, `JDialog`, etc.).

Podría decirse que tanto la **composición** como la **anidación** son casos particulares de **clientela**, pues en realidad en todos esos casos una clase está haciendo uso de otra (al contener atributos que son objetos de la otra clase, al definir clases dentro de otras clases, al utilizar objetos en el paso de parámetros, al declarar variables locales utilizando otras clases, etc.).

A lo largo de la unidad, irás viendo distintas posibilidades de implementación de clases haciendo uso de todas estas relaciones, centrándonos especialmente en el caso de la **herencia**, que es la que permite establecer las relaciones más complejas.

Autoevaluación

¿Cuál crees que será la relación entre clases más habitual?

- ○ Clientela.
- ○ Anidación o anidamiento.
- ○ Herencia.

Entre las clases no existen relaciones. Son entidades aisladas en el sistema y sin relaciones con el exterior.