7.A. Relación entre clases.

Sitio: Formación Profesional a Distancia
Curso: Programación
Libro: 7.A. Relación entre clases.
Imprimido por: Iván Jiménez Utiel
Día: lunes, 10 de febrero de 2020, 15:47

Tabla de contenidos

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.

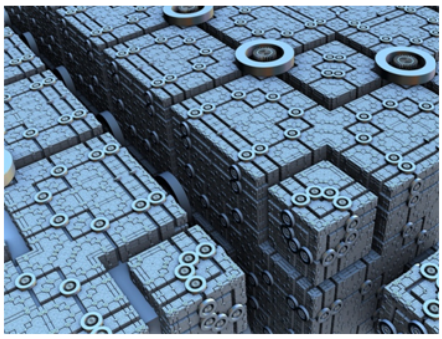
Autoevaluación

¿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.

1.1. Composición.

Cuando en un sistema de información, una determinada entidad A contiene a otra B como una de sus partes, se suele decir que se está produciendo una relación de **composición**. Es decir, el objeto de la clase A contiene a uno o varios objetos de la clase B.



Por ejemplo, si describes una entidad **País** compuesta por una serie de atributos, entre los cuales se encuentra una lista de comunidades autónomas, podrías decir que los objetos de la clase **País** contienen varios objetos de la clase **ComunidadAutonoma**. Por otro lado, los objetos de la clase **ComunidadAutonoma** podrían contener como atributos objetos de la clase **Provincia**, la cual a su vez también podría contener objetos de la clase **Municipio**.

Como puedes observar, la **composición** puede encadenarse todas las veces que sea necesario hasta llegar a objetos básicos del lenguaje o hasta tipos primitivos que ya no contendrán otros objetos en su interior. Ésta es la forma más habitual de definir clases: mediante otras clases ya definidas anteriormente. Es una manera eficiente y sencilla de gestionar la reutilización de todo el código ya escrito. Si se definen clases que describen entidades distinguibles y con funciones claramente definidas, podrán utilizarse cada vez que haya que representar objetos similares dentro de otras clases.

La **composición** se da cuando una clase contiene algún atributo que es una referencia a un objeto de otra clase.

Una forma sencilla de plantearte si la relación que existe entre dos clases A y B es de **composición** podría ser mediante la expresión idiomática "**tiene un**": "la clase A tiene uno o varios objetos de la clase B", o visto de otro modo: "Objetos de la clase B pueden formar parte de la clase A".

Algunos ejemplos de composición podrían ser:

- Un **coche** tiene un **motor** y tiene cuatro **ruedas**.
- Una **persona** tiene un **nombre**, una **fecha de nacimiento**, una **cuenta bancaria** asociada para ingresar la nómina, etc.
- Un **cocodrilo** bajo investigación científica que tiene un número de **dientes** determinado, una **edad**, unas **coordenadas** de ubicación geográfica (medidas con GPS), etc.

Recuperando algunos de los ejemplos de clases que has utilizado en otras unidades:

- Una clase `Rectangulo` podría contener en su interior dos objetos de la clase `Punto` para almacenar los vértices inferior izquierdo y superior derecho.
- Una clase `Empleado` podría contener en su interior un objeto de la clase **DNI** para almacenar su DNI/NIF, y otro objeto de la clase `CuentaBancaria` para guardar la cuenta en la que se realizan los ingresos en nómina.

Ejercicio resuelto

¿Podría decirse que la relación que existe entre la clase Ave y la clase Loro es una relación de composición?

Solución:

No. Aunque claramente existe algún tipo de relación entre ambas, no parece que sea la de composición. No parece que se cumpla la expresión "**tiene un**": "Un loro tiene un ave". Se cumpliría más bien una expresión del tipo "**es un**": "Un loro es un ave". Algunos objetos que cumplirían la relación de composición podrían ser **Pico** o **Alas**, pues "un loro tiene un pico y dos alas", del mismo modo que "un ave tiene pico y dos alas". Este tipo de relación parece más de **herencia** (un loro es un tipo de ave).

1.2. Herencia.

El mecanismo que permite crear clases basándose en otras que ya existen es conocido como **herencia**. Como ya has visto en unidades anteriores, Java implementa la herencia mediante la utilización de la palabra reservada `extends`.

El concepto de **herencia** es algo bastante simple y sin embargo muy potente: cuando se desea definir una nueva clase y ya existen clases que, de alguna manera, implementan parte de la funcionalidad que se necesita, es posible crear una nueva **clase derivada** de la que ya tienes. Al hacer esto se posibilita la reutilización de todos los atributos y métodos de la clase que se ha utilizado como **base** (**clase padre** o **superclase**), sin la necesidad de tener que escribirlos de nuevo.

Una **subclase** hereda todos los miembros de su **clase padre** (atributos, métodos y clases internas). Los **constructores** no se heredan, aunque se pueden invocar desde la **subclase**.

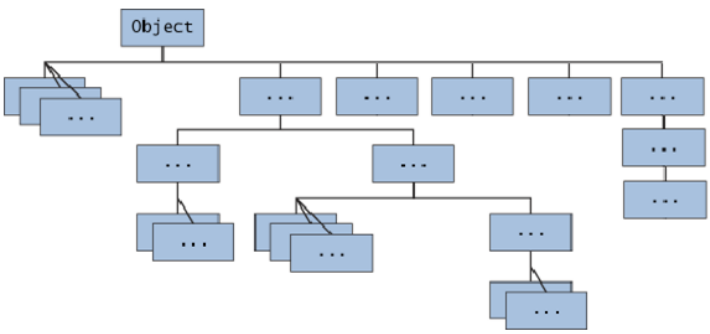
Algunos ejemplos de herencia podrían ser:

- Un **coche** es un **vehículo** (heredará atributos como la **velocidad máxima** o métodos como **parar** y **arrancar**).
- Un **empleado** es una **persona** (heredará atributos como el **nombre** o la **fecha de nacimiento**).
- Un **rectángulo** es una **figura geométrica** en el plano (heredará métodos como el cálculo de la **superficie** o de su **perímetro**).
- Un **cocodrilo** es un **reptil** (heredará atributos como por ejemplo el **número de dientes**).

En este caso la expresión idiomática que puedes usar para plantearte si el tipo de relación entre dos clases A y B es de herencia podría ser "**es un**": "la clase A es un tipo específico de la clase B" (**especialización**), o visto de otro modo: "la clase B es un caso general de la clase A" (**generalización**).

En Java, la clase `Object` (dentro del paquete `java.lang`) define e implementa el comportamiento común a todas las clases (incluidas aquellas que tú escribas). Como recordarás, ya se dijo que en Java cualquier clase deriva en última instancia de la clase `Object`.

Todas las clases tienen una **clase padre**, que a su vez también posee una **superclase**, y así sucesivamente hasta llegar a la clase `Object`. De esta manera, se construye lo que habitualmente se conoce como una **jerarquía de clases**, que en el caso de Java tendría a la clase `Object` en la raíz.



Ejercicio resuelto

Cuando escribas una clase en Java, puedes hacer que herede de una determinada clase padre (mediante el uso de `extends`) o bien no indicar ninguna herencia. En tal caso tu clase no heredará de ninguna otra clase Java. ¿Verdadero o Falso?

Solución:

No es cierto. Aunque no indiques explícitamente ningún tipo de herencia, el compilador asumirá entonces de manera implícita que tu clase hereda de la clase `Object`, que define e implementa el comportamiento común a todas las clases.

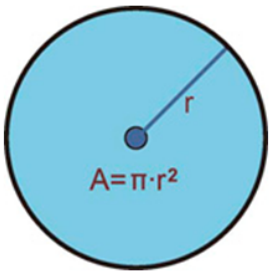
1.3. ¿Herencia o composición?.

Cuando escribas tus propias clases, debes intentar tener claro en qué casos utilizar la **composición** y cuándo la **herencia**:

- **Composición**: cuando una clase está formada por objetos de otras clases. En estos casos se incluyen objetos de esas clases, pero no necesariamente se comparten características con ellos (no se heredan características de esos objetos, sino que directamente se utilizarán sus atributos y sus métodos). Esos objetos incluidos no son más que atributos miembros de la clase que se está definiendo.

- **Herencia**: cuando una clase cumple todas las características de otra. En estos casos la clase derivada es una **especialización** (**particularización**, **extensión** o **restricción**) de la clase base. Desde otro punto de vista se diría que la clase base es una **generalización** de las clases derivadas.

Por ejemplo, imagina que dispones de una clase **Punto** (ya la has utilizado en otras ocasiones) y decides definir una nueva clase llamada **Círculo**. Dado que un punto tiene como atributos sus coordenadas en plano (**x1**, **y1**), decides que es buena idea aprovechar esa información e incorporarla en la clase **Circulo** que estás escribiendo. Para ello utilizas la **herencia**, de manera que al derivar la clase **Círculo** de la clase **Punto**, tendrás disponibles los atributos **x1** e **y1**. Ahora solo faltaría añadirle algunos atributos y métodos más como por ejemplo el **radio** del círculo, el cálculo de su **área** y su **perímetro**, etc.



En principio parece que la idea pueda funcionar pero es posible que más adelante, si continúas construyendo una **jerarquía de clases**, observes que puedas llegar a conclusiones incongruentes al suponer que un círculo es una especialización de un punto (un tipo de punto). ¿Todas aquellas figuras que contengan uno o varios puntos deberían ser tipos de punto? ¿Y si tienes varios puntos? ¿Cómo accedes a ellos? ¿Un rectángulo también tiene sentido que herede de un punto? No parece muy buena idea.

Parece que en este caso habría resultado mejor establecer una relación de **composición**. Analízalo detenidamente: ¿cuál de estas dos situaciones te suena mejor?

1. "**Un círculo es un punto** (su centro)", y por tanto heredará las coordenadas **x1** e **y1** que tiene todo punto. Además tendrá otras características específicas como el **radio** o métodos como el cálculo de la **longitud** de su perímetro o de su **área**.
2. "**Un círculo tiene un punto** (su centro)", junto con algunos atributos más como por ejemplo el **radio**. También tendrá métodos para el cálculo de su **área** o de la longitud de su **perímetro**.

Parece que en este caso la **composición** refleja con mayor fidelidad la relación que existe entre ambas clases. Normalmente suele ser suficiente con plantearse las preguntas "**¿A es un tipo de B?**" o "¿**A contiene elementos de tipo B?**".