Herencia

DigitalHouse >
Coding School

Certified Tech Developer
The Ultimate Degree

Índice

1 Relación “es un”

“La herencia es uno de los pilares del paradigma orientado a objetos, también conocida como una relación del tipo “es un”.”

“En un” entre clases

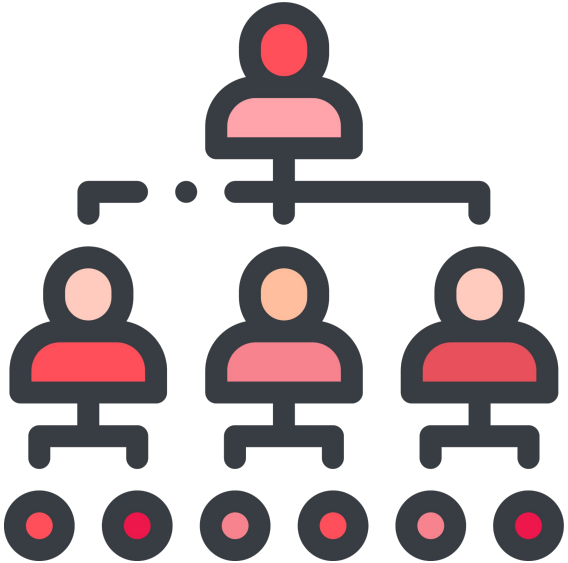
Todos los perros tienen un nombre, una edad y todos ladran y juegan. Cuando nos nombran estas características rápidamente reconocemos que se trata de un perro.

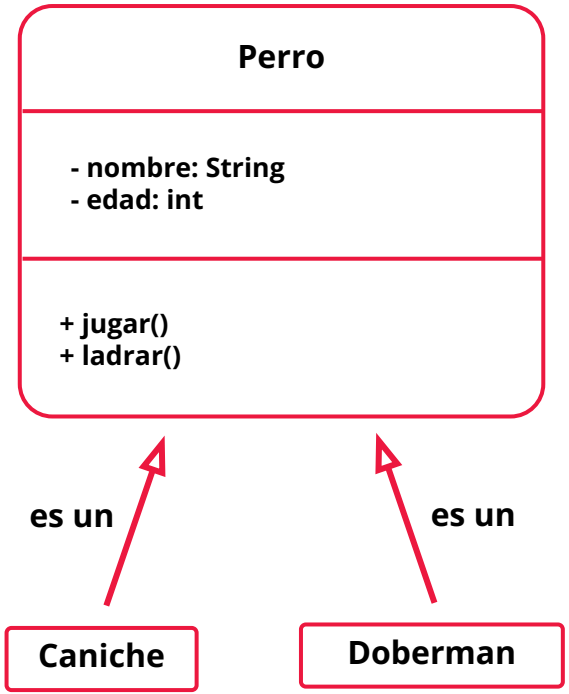
Si analizamos un caniche, veremos como **juega** y **ladra**; si analizamos un doberman, también veremos como **juega** y **ladra**, aunque lo hace muy distinto al caniche.

Entonces, tanto el caniche como el doberman juegan, ladran y ambos tienen nombre y edad: sería razonable asumir que si tienen y hacen todo lo que hace un perro, entonces, **es un** perro.

Por todo esto, podemos decir que un caniche **es un** Perro. De la misma manera que podríamos decir que Profesor **es un** Empleado, y más aún: un Empleado **es una** Persona, por lo tanto, un Profesor **es una** Persona.

Nuevamente, al observar la realidad y pasar por el proceso de abstracción, obtuvimos una serie de entidades que se ordenan naturalmente, y la herencia responde a ello.





Podemos decir entonces que la **herencia** es un **ordenamiento entre clases** que define **una relación “es un”**. Entonces decimos que un caniche y un dóberman **es un** perro, porque tiene y hace todo lo que hace un perro.

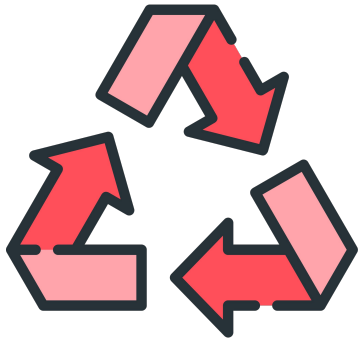
2 ¿Para qué sirve la herencia?

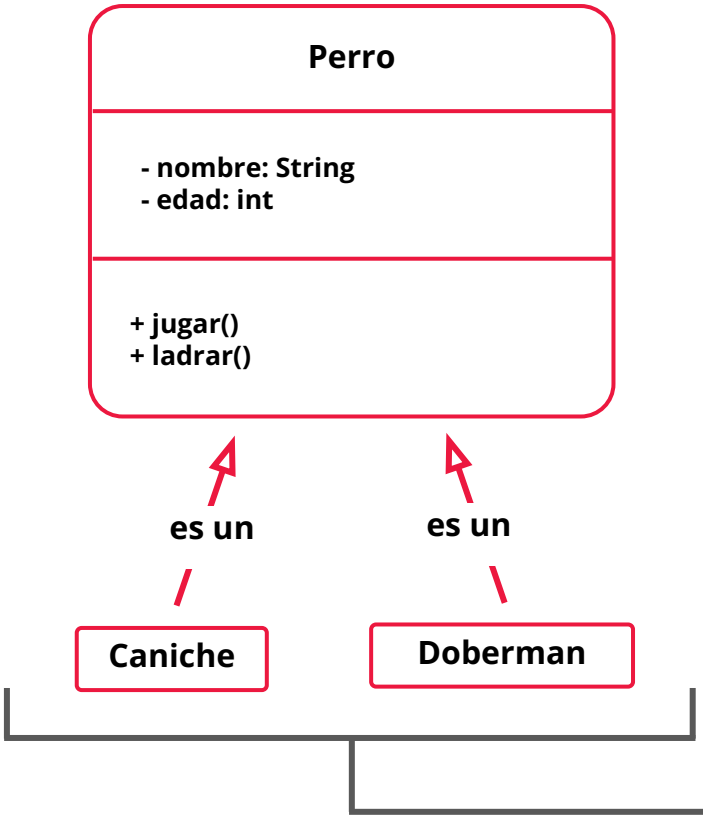
Utilidad de la herencia

Esta es una pregunta interesante, ya que la herencia es uno de los pilares de la orientación a objetos. Si analizamos el esquema anterior, tanto Caniche como Doberman hacen lo mismo que hace el perro. ¿Si hacen lo mismo que el perro, para qué escribir el código de lo que hacen? ¿No sería más conveniente escribirlo una sola vez en la clase Perro y que Doberman, Caniche, “obtengan” este comportamiento desde Perro?

De hacer esto, decimos que Caniche y Dóberman “heredan” el comportamiento de un perro, es decir, la clase Dóberman hereda de la clase Perro, todos sus atributos y responsabilidades favoreciendo la reutilización.

La Herencia favorece la reutilización de código





Tanto Caniche como Doberman tienen los atributos nombre y edad, como así también las responsabilidades jugar() y ladrar() porque las hereda de Perro.

3 | Ejemplos

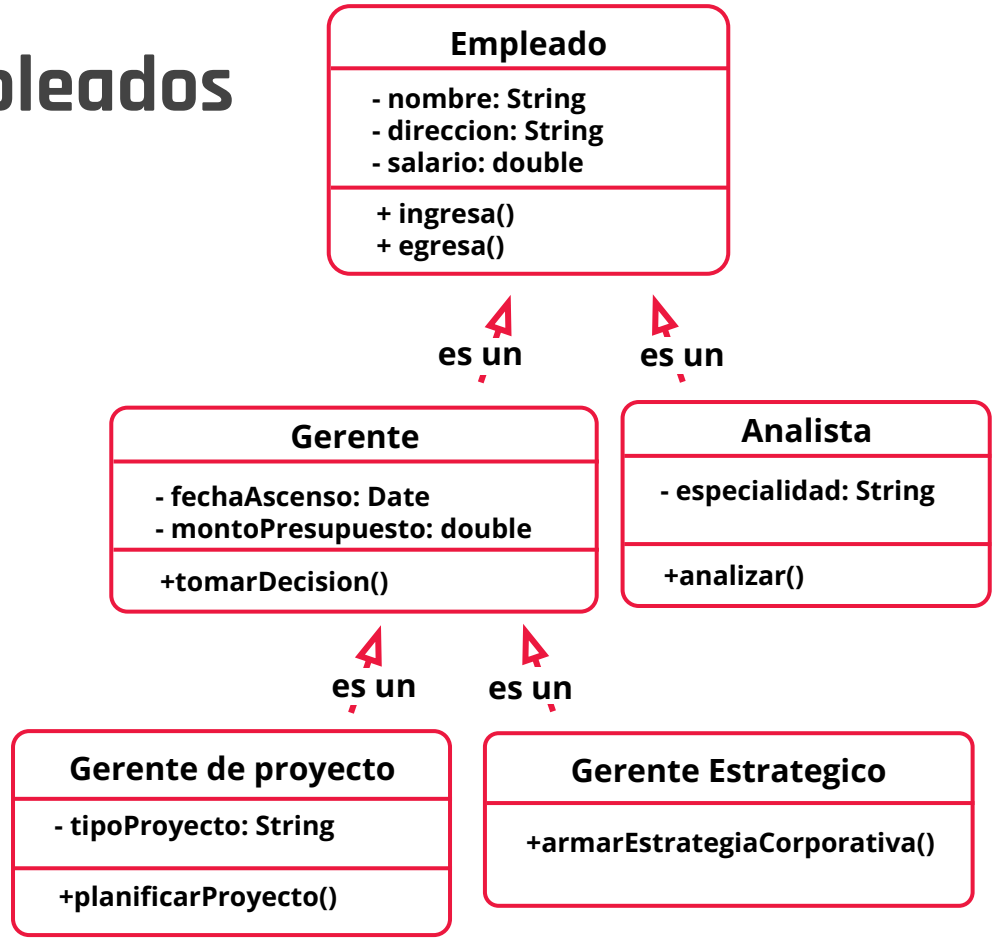
Ejemplo Jerarquía de empleados

Gerente

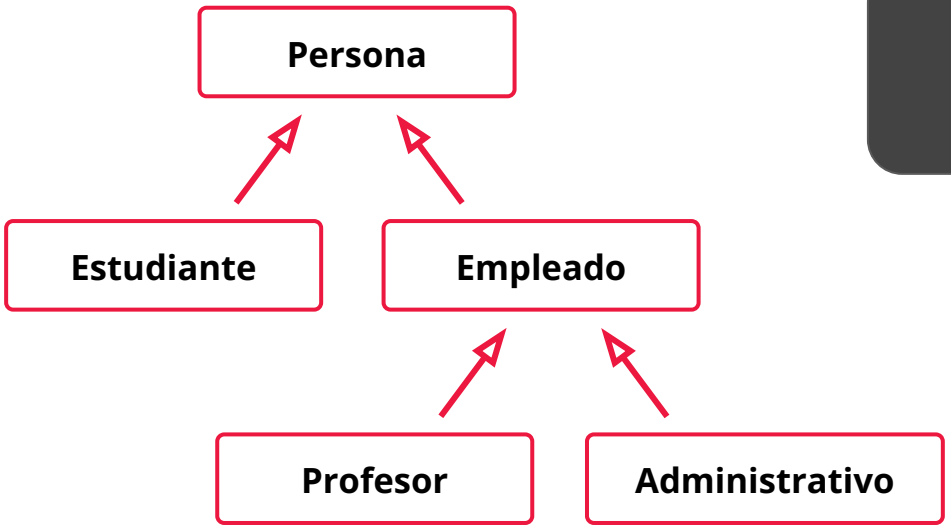
Además de los atributos fechaAscenso y montoPresupuesto, tiene los atributos nombre, direccion y salario que hereda de Empleado como así también las responsabilidades ingresa() y egresa().

Gerente de proyecto

Además del atributo tipoProyecto y la responsabilidad de planificarProyecto() tiene todos los atributos y responsabilidades de Gerente y Empleado



Ejemplo Jerarquía de personas

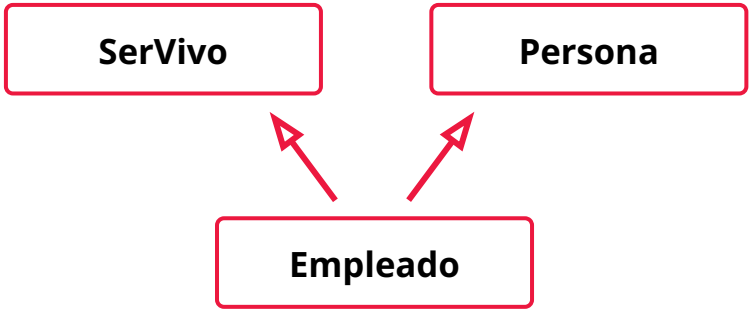


Una clase que hereda de otra, suma a sus propios atributos y responsabilidades los de la clase a la cual hereda.

4 | Herencia múltiple

Herencia múltiple

Se establece cuando una clase hereda de varias otras clases, en este caso, la clase hija hereda atributos y responsabilidades de los diferentes padres.

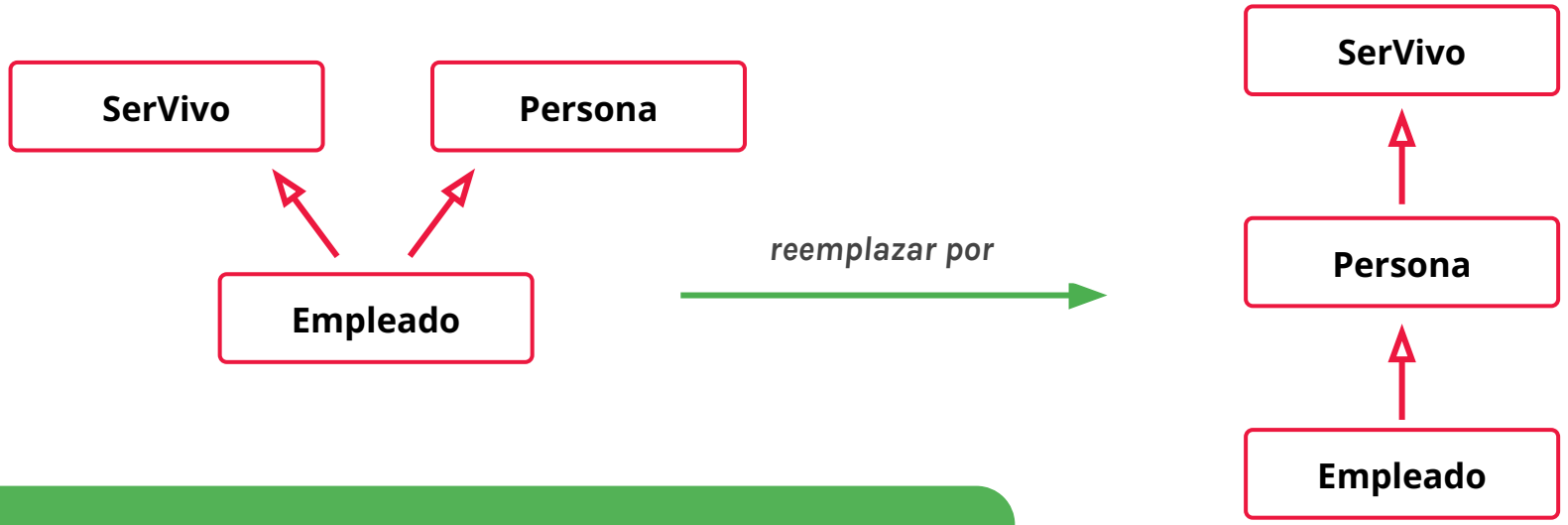


Herencia múltiple

El uso de herencia múltiple requiere una consideración muy atenta para evitar la superposición funcional de atributos y responsabilidades.

Es por ello que en Java no está permitida la herencia múltiple y al no ser considerada una buena práctica de diseño no la utilizaremos en esta materia.

Herencia múltiple



Como el uso de la herencia lo tendremos prohibido, buscaremos la manera siempre de mantener una línea de herencia.


DigitalHouse>
Coding School