Polimorfismo

DigitalHouse >
Coding School

Certified Tech Developer
The Ultimate Degree

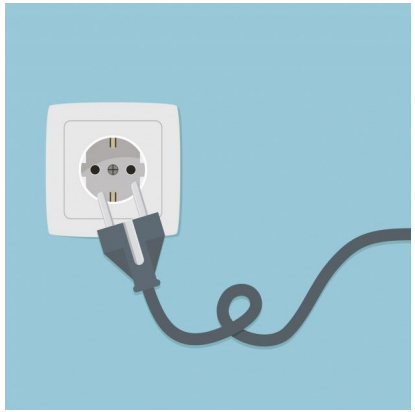
Índice

Antes de ver qué es el polimorfismo, debemos entender **qué es la vinculación dinámica**.

1 Vinculación dinámica (dynamic binding)

Vinculación dinámica (dynamic binding)

La vinculación dinámica de una referencia funciona es igual que un enchufe. En un enchufe se puede conectar diferentes cosas: un TV, una heladera, una notebook.



Veremos que en una referencia podremos apuntar a diferentes tipos de objetos.

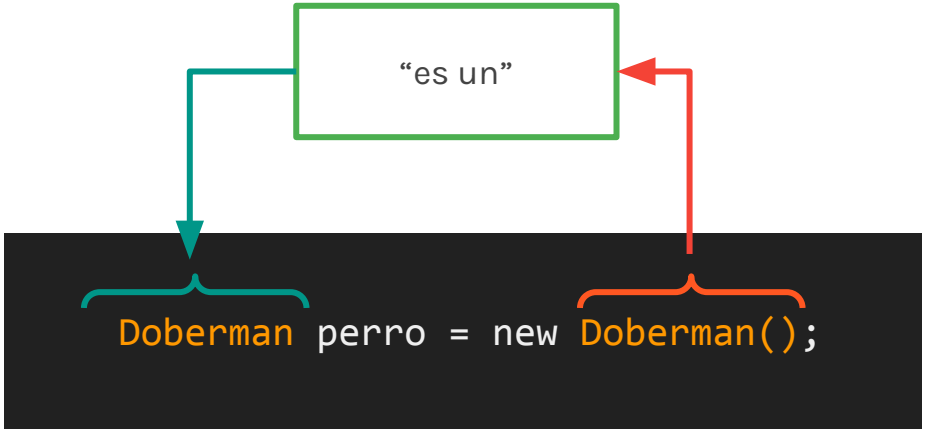
Aquí, tanto la **referencia** como el **objeto** referenciado son del mismo tipo: Doberman. Sin embargo, es posible que la referencia y el objeto referenciado sean de distinto tipo.

Objeto

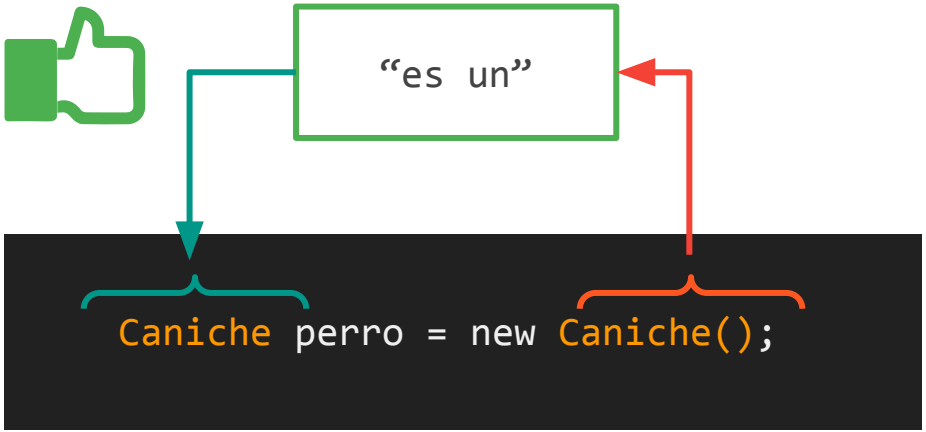
```
Doberman perro = new Doberman();
```

Referencia/Variable

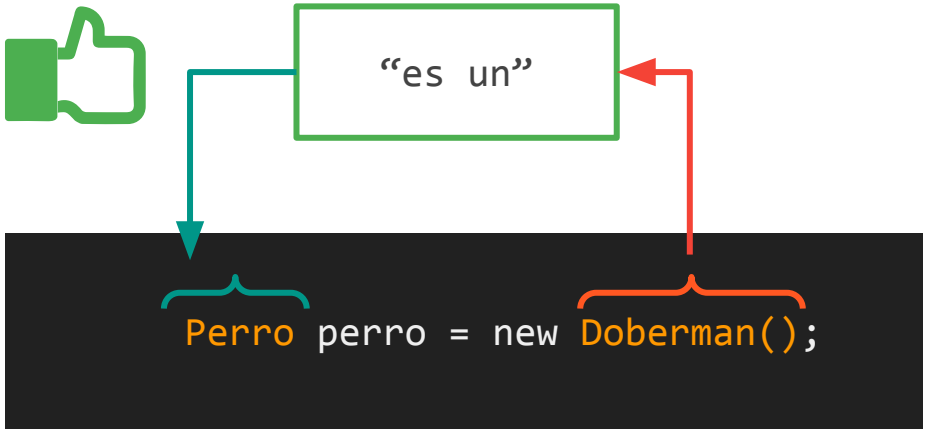
En los lenguajes que no son tipados la **referencia** y el **objeto** pueden ser de cualquier tipo, en los **fuertemente tipados** como Java el **objeto** debe ser de una clase que tenga una **relación del tipo "es un"** respecto de la **referencia**.



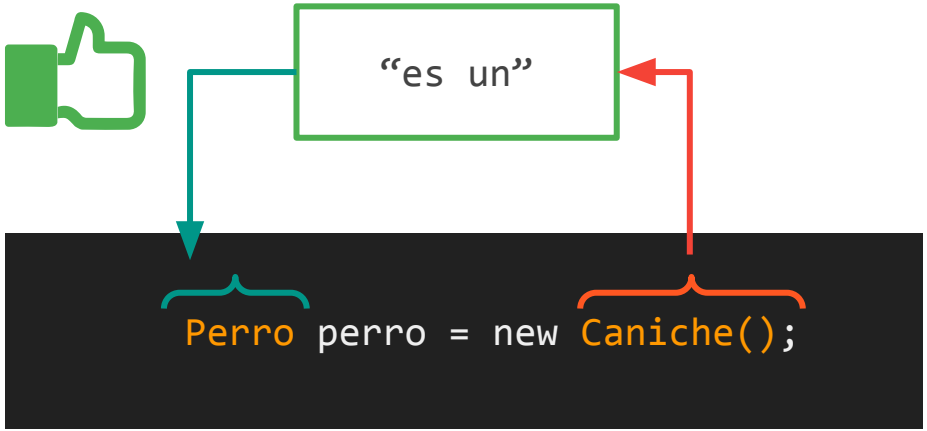
Veamos el siguiente ejemplo:



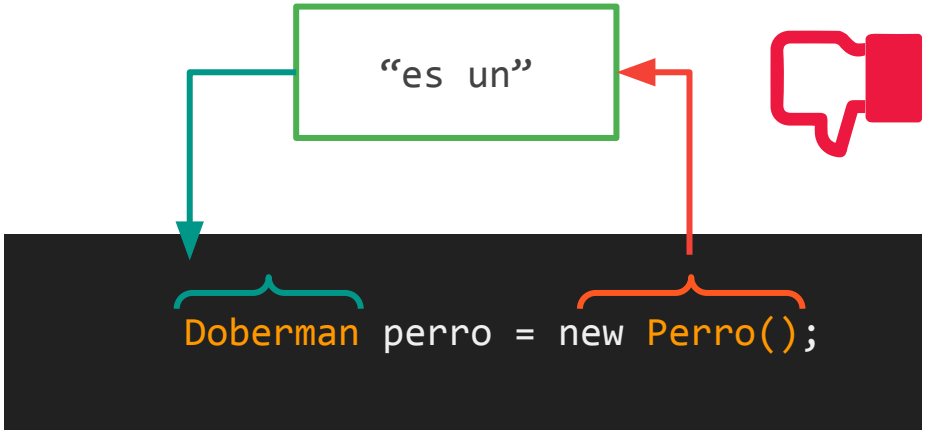
Vemos que en este caso la **referencia** tiene un tipo diferente al **objeto** referenciado, pero cumple con la condición “es un”.



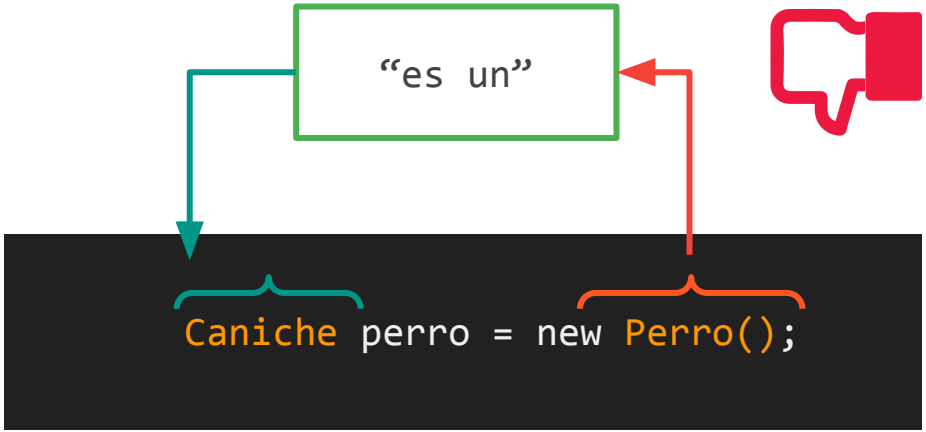
Vemos que en este caso la **referencia** tambien tiene un tipo diferente al **objeto** referenciado pero cumple con la condición “es un”.



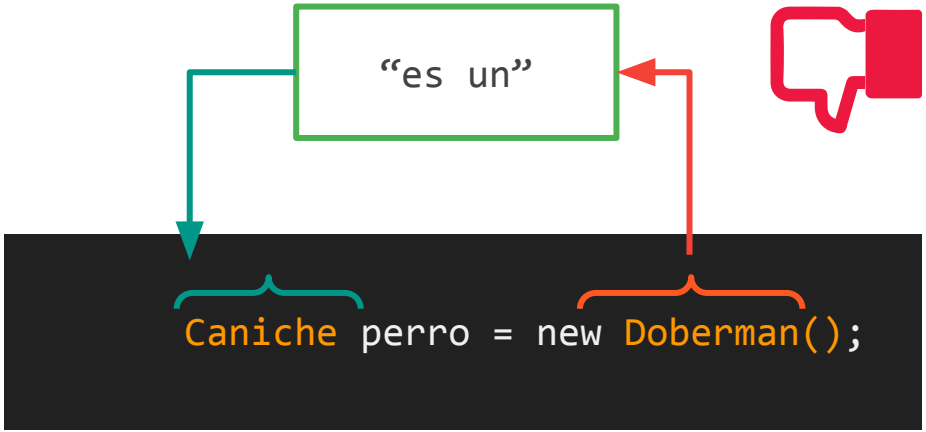
Un perro no necesariamente siempre es un Doberman. Esto no está permitido.



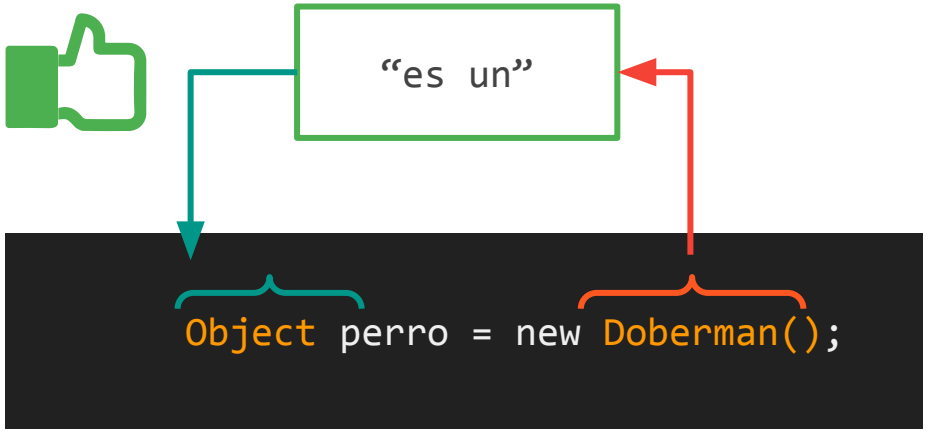
De la misma manera, un perro no necesariamente siempre es un Caniche. Esto no está permitido.



Un Doberman puede tener cosas parecidas a un Caniche, pero no lo es.



En Java todas las clases por definición heredan de Object con lo cual un Doberman es un Object.



2 | Polimorfismo

Polimorfismo

Es la capacidad de un mismo objeto de comportarse como otro. En otras palabras, es la capacidad de un objeto de funcionar de diversas formas.
Veamos con lo ejemplos anteriores:

```
Perro p;

p = new Doberman();
p.ladrar();

p = new Caniche();
p.ladrar();
```

La referencia p se puede comportar y `ladrar` como un `Doberman`.

Y la misma referencia p al vincularse dinámicamente (dynamic binding) con un `Caniche`.

Se comporta de forma diferente y `ladra` como un `Caniche`.

Si utilizamos polimorfismo, podemos estar seguros de que modificaciones futuras, que agreguen nuevas subclases, no deberían afectar el código ni su funcionamiento.

Si el código usa Perros (es decir, cualquier objeto que “es un” Perro) siempre que nuevas razas de perros introducidas al sistema hereden de Perro funcionaran correctamente.

3 | Casting

Casting

Supongamos que **Doberman** tiene un método llamado **morderComoDoberman()**, pero la referencia o sea la variable es del tipo Perro. Para forzar a un perro a que sea un Doberman utilizamos el **casteo**. De esta manera podremos invocar los métodos propios de Doberman.

```
Perro perro = new Doberman();
        perro.ladrar();

((Doberman)perro).morderComoDoberman()
```

Casteo

Lo mismo sucede si nuestro objeto referencia es del tipo Object. En este caso como la clase Object no tiene tampoco el método ladrar() debemos castearlo ya sea a Perro que tiene dicho método o a Doberman.

```
Object perro = new Doberman();
((Perro)perro).ladrar()

((Doberman)perro).morderComoDoberman()
```

Casteo


DigitalHouse>
Coding School