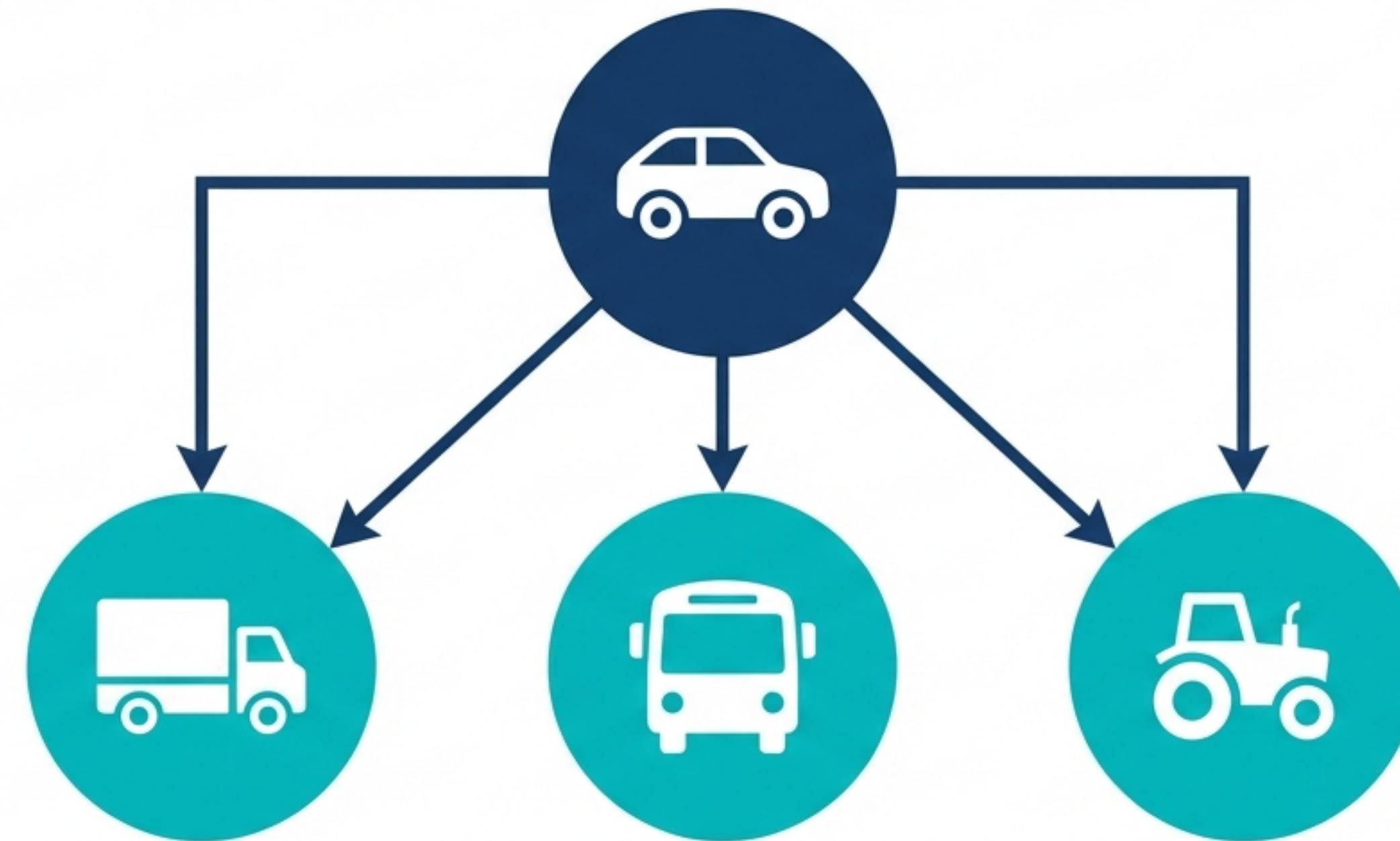


# Programación Orientada a Objetos: Dominando la Herencia y el Polimorfismo

Guía completa para la reutilización de código y jerarquías de clases en Java.

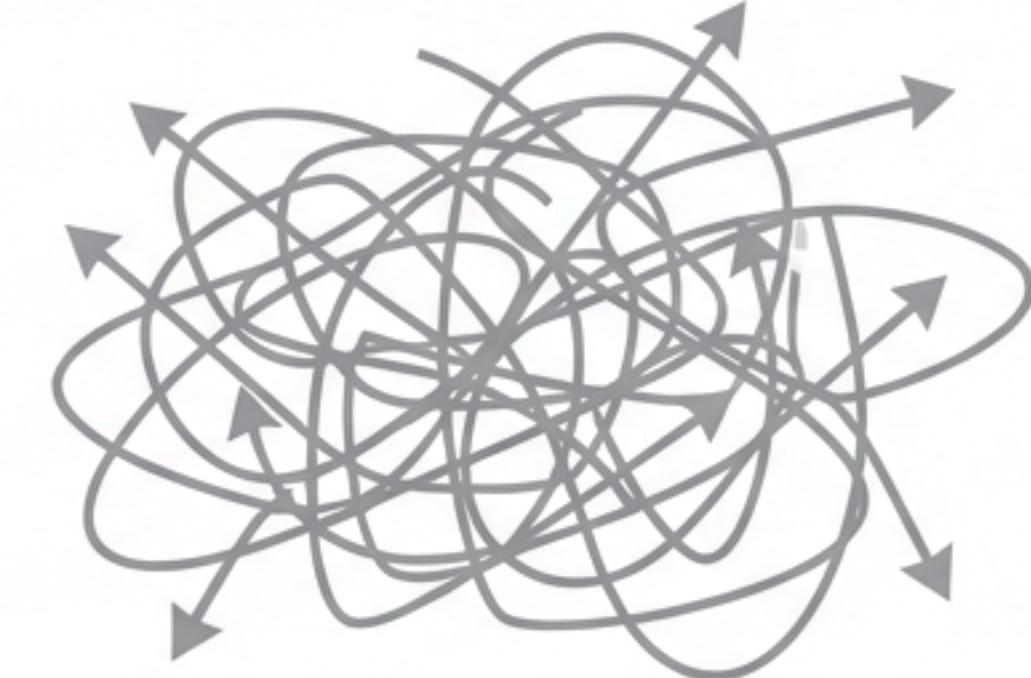


# Introducción: La necesidad de reutilizar código

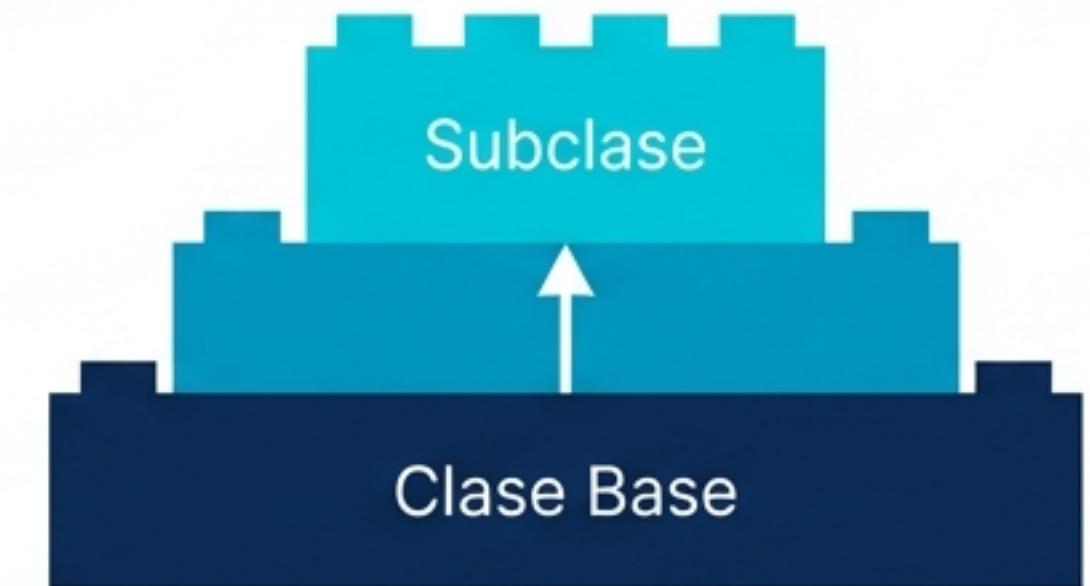
El concepto de herencia surge de la necesidad de **reutilizar y extender** el código de manera eficiente.

Permite que nuevas clases adopten propiedades y comportamientos de clases existentes sin necesidad de recodificar ni volver a testear métodos ya probados.

- ✓ Reutilización de código.
- ✓ Creación de jerarquías lógicas.
- ✓ Facilitación del mantenimiento y la extensibilidad.



**Código Redundante (Sin Herencia)**

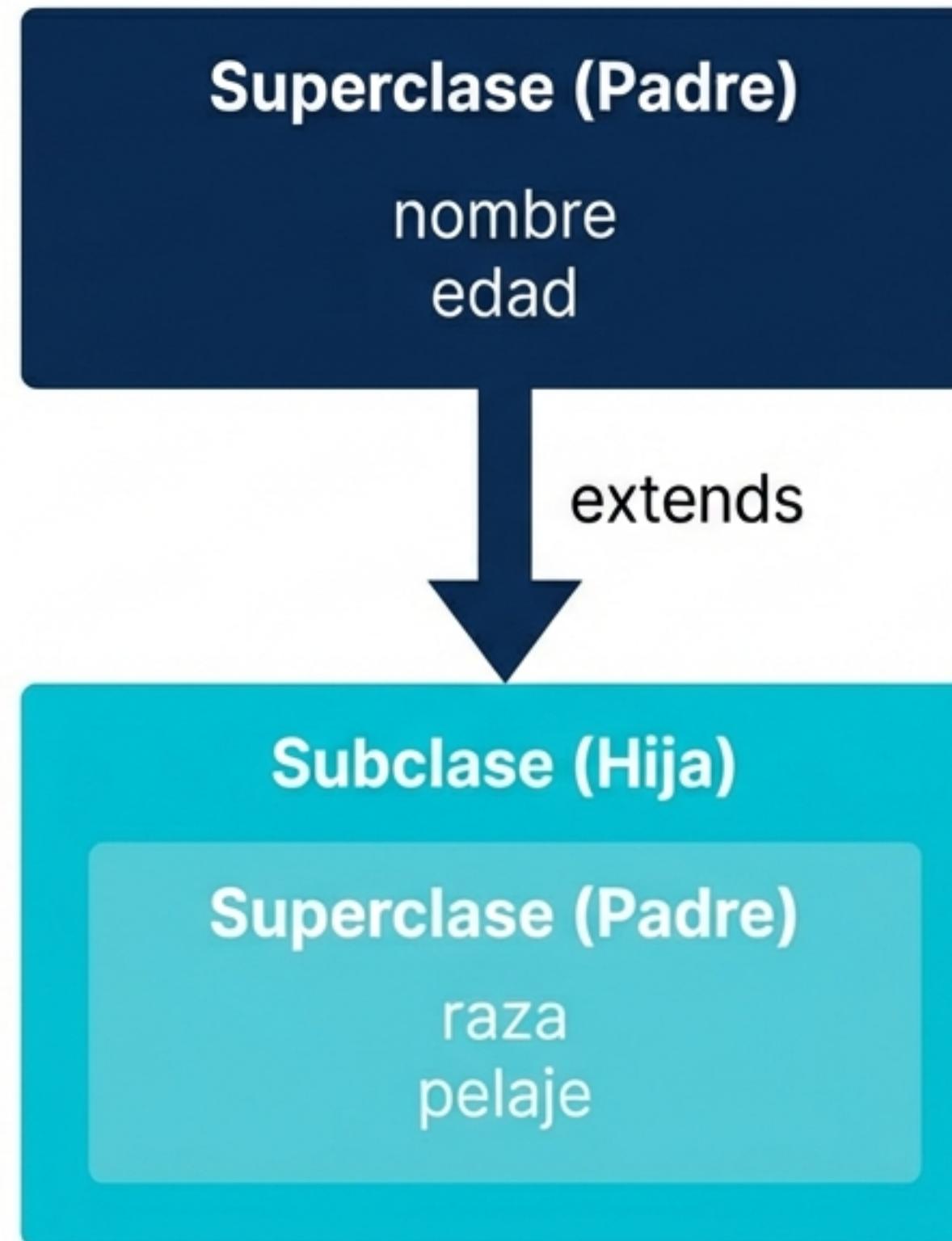


**Código Jerárquico (Con Herencia)**



**Sabías que...** En la década de 1960, lenguajes como Simula y Smalltalk introdujeron la idea de clases y objetos, conduciendo a la conceptualización de la herencia.

# Conceptos Clave: Padre e Hija



**Superclase:** La clase de la cual se hereda.

**Subclase:** La clase que hereda.  
Adquiere todos los atributos y métodos (getters, setters, tostring), excepto constructores y miembros privados.

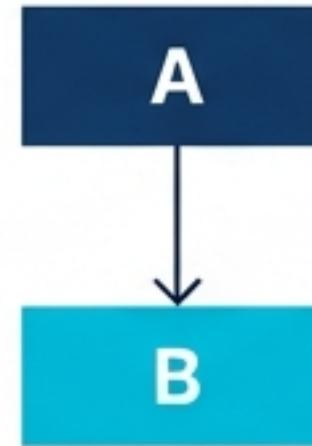
## La Regla 'Is-A' (Es un)

Cuando una clase hereda de otra, se establece una relación de pertenencia.

Ejemplo: La clase 'Perro' **es un** 'Animal'.

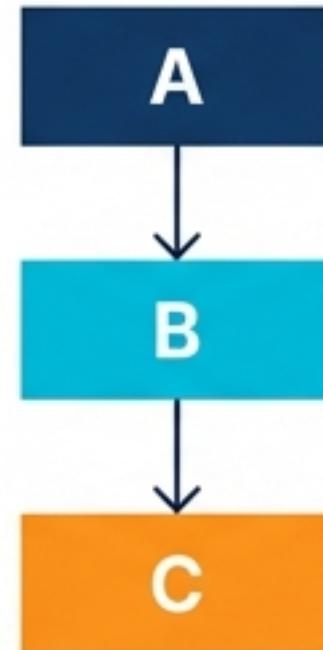
# Tipos de Herencia en Java

## Herencia Simple (Soportada)



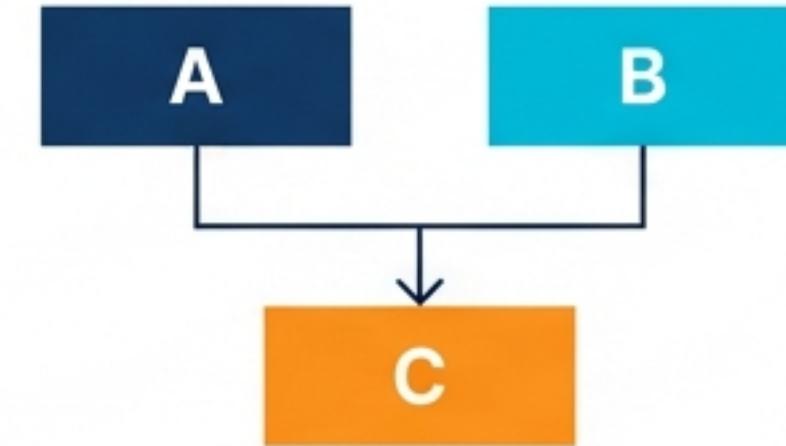
Una clase hija tiene una única clase padre. Se usa la palabra clave '**extends**'.

## Herencia Multinivel (Soportada)



Cadena de herencia permitida.

## Herencia Múltiple (No en Java)



Java no soporta herencia de múltiples clases para evitar ambigüedades (El problema del "Diamante de la Muerte").

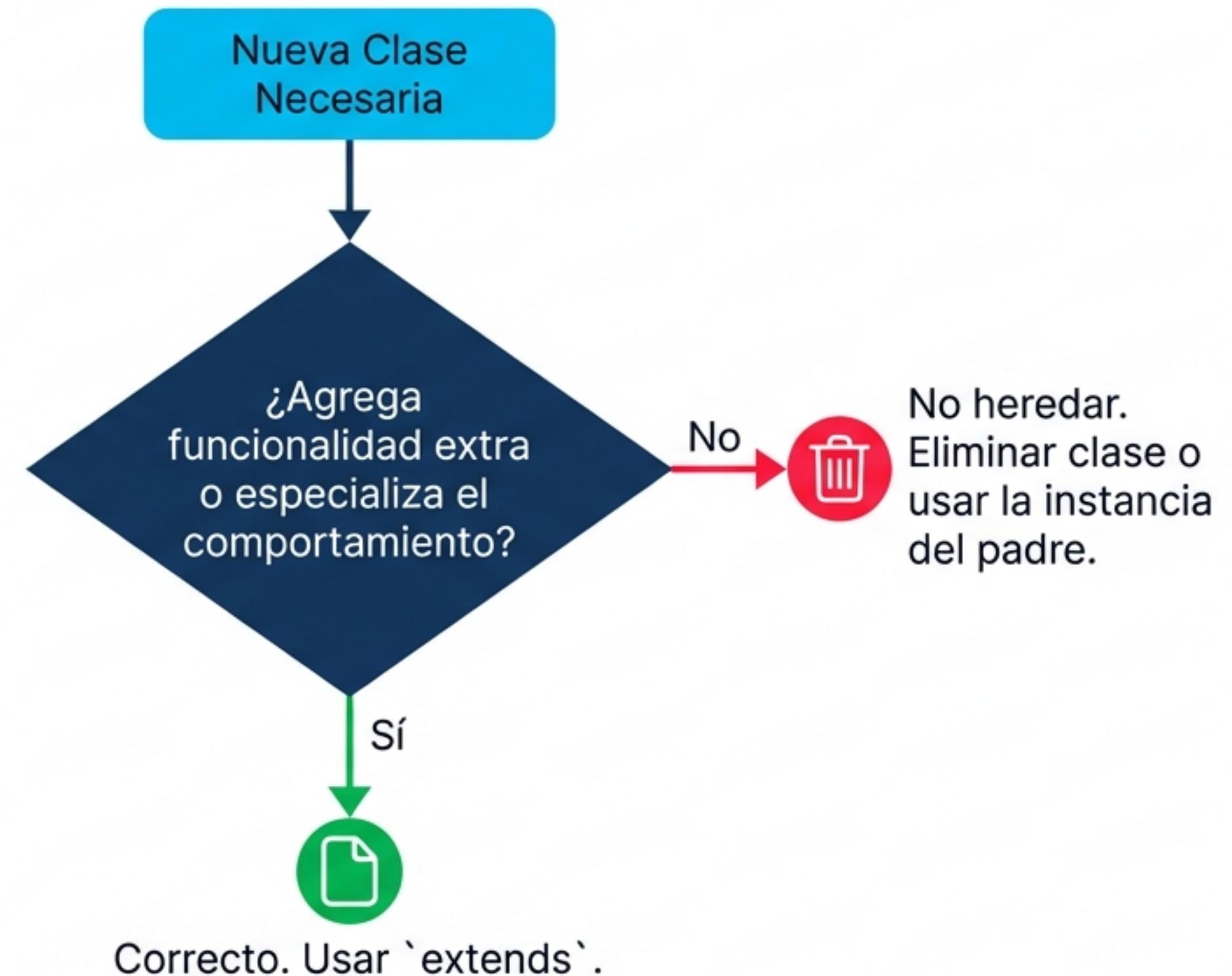
Nota: Java simula herencia múltiple mediante el uso de Interfaces.

# Caso Práctico: ¿Cuándo debemos heredar?

**El Problema:** Crear una subclase que no añade atributos ni métodos nuevos es un error de diseño.

**La Regla:** La herencia solo tiene sentido si la subclase agrega valor a la superclase.

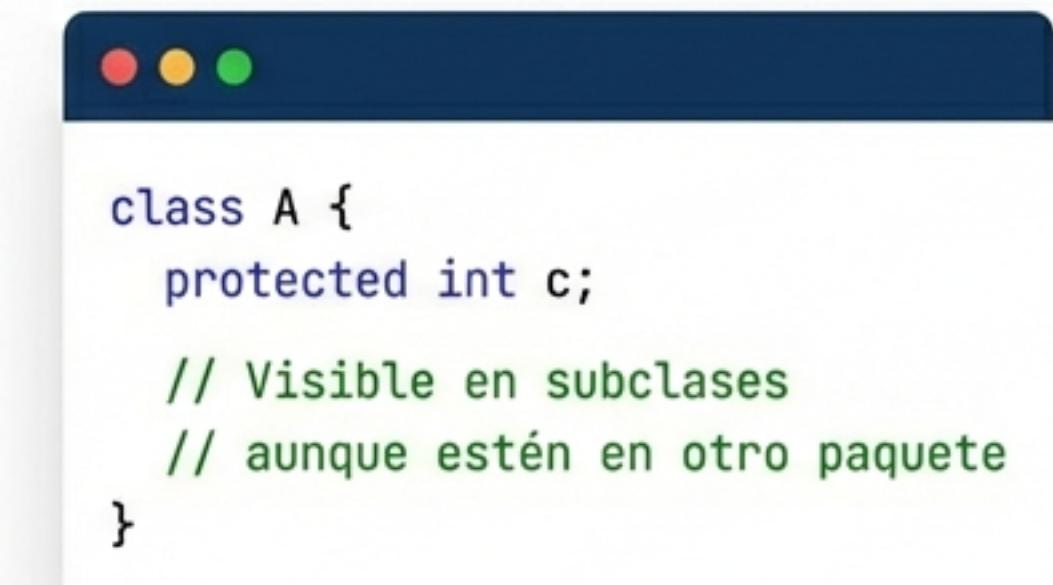
Clase A
- a
b
# c
+ d



# Control de Acceso: El modificador `protected`

El puente entre `private` y `public` para la herencia.

Modificador	Misma Clase	Mismo Paquete	Subclase (Otro Paquete)	Mundo (Otro Paquete)
public	Sí	Sí	Sí	Sí
protected	Sí	Sí	Sí	No
default	Sí	Sí	No	No
private	Sí	No	No	No



```
class A {  
    protected int c;  
  
    // Visible en subclases  
    // aunque estén en otro paquete  
}
```

# Sobreescritura de Métodos (`@Override`)

## Clase Padre: Animal

```
1 class Animal {  
2     public void hacerSonido() {  
3         System.out.println("Haciendo un sonido genérico.");  
4     }  
5 }
```



La hija modifica el comportamiento

## Clase Hija: Perro

```
1 class Perro extends Animal {  
2     @Override  
3     public void hacerSonido() {  
4         System.out.println("Ladrando...");  
5     }  
6 }
```

## Concepto:

La clase hija redefine un método heredado.

## Requisitos:

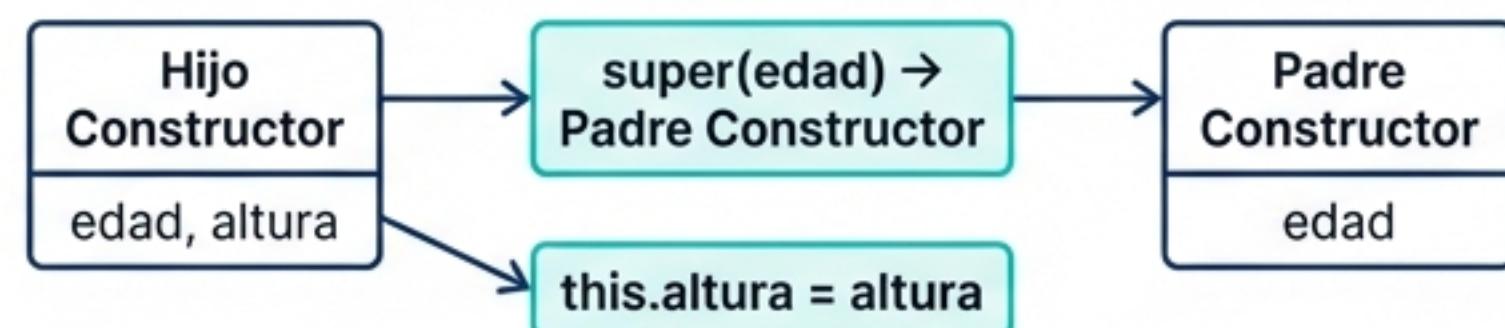
1. Mismo nombre.
2. Mismos parámetros.
3. Etiqueta **@Override**  
(Buenas prácticas).

# La instrucción `super`

## 1. Invocar al Constructor Padre

`super()` debe ser la **primera línea** del constructor de la hija.

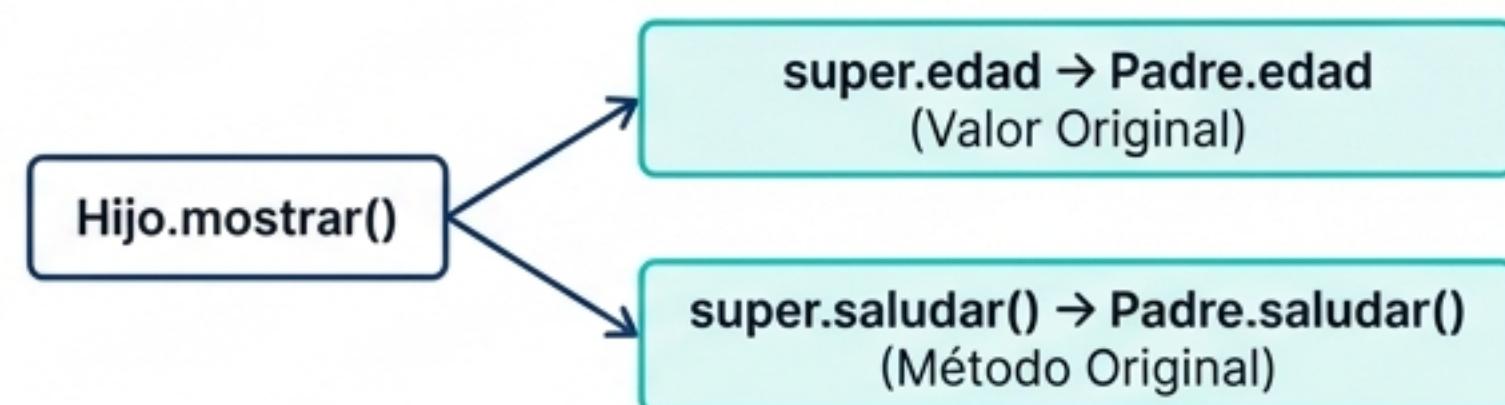
```
Mac OS X icon  
Hijo(int edad, int altura) {  
    super(edad); // Llama al constructor Padre  
    this.altura = altura;  
}
```



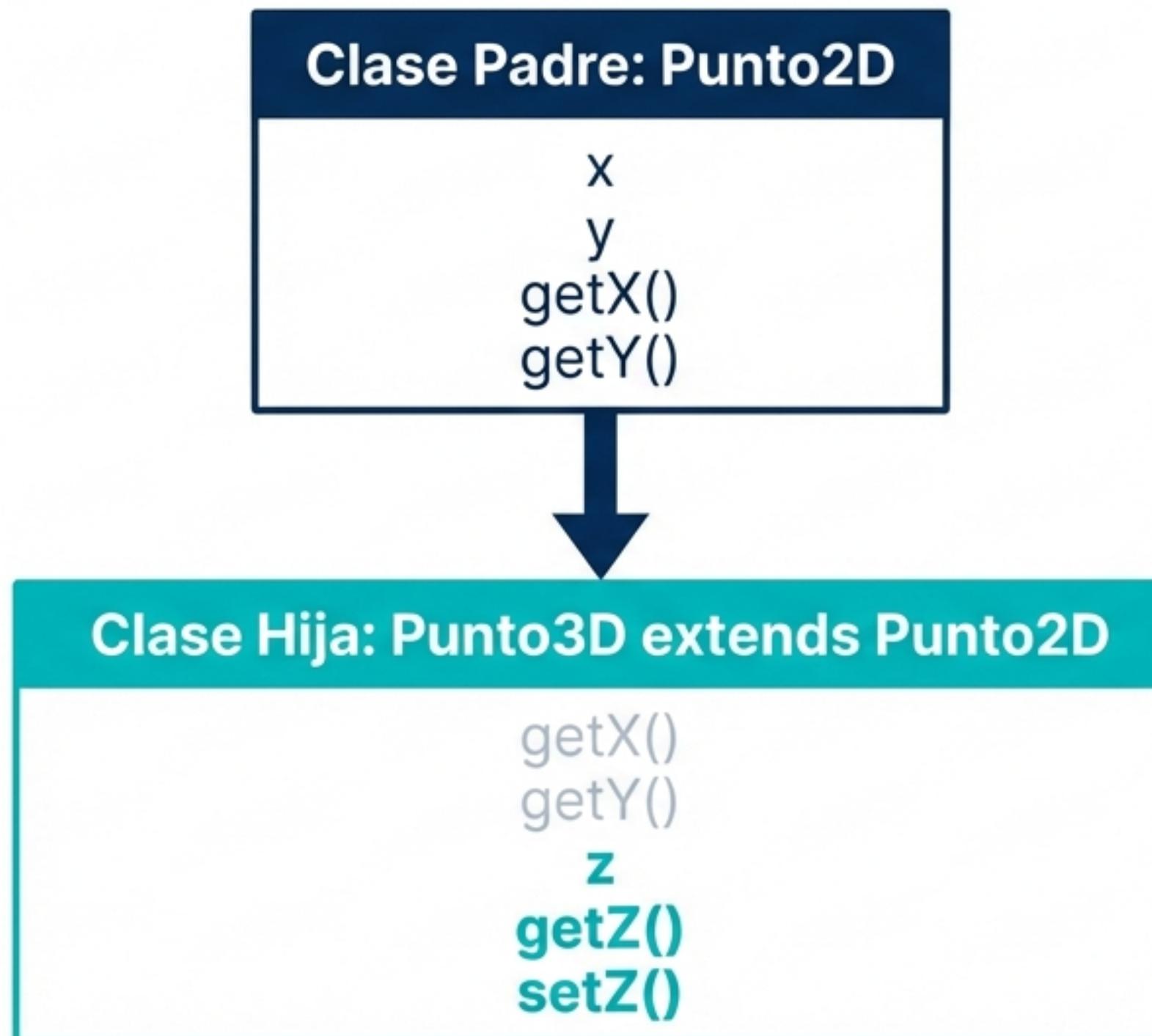
## 2. Acceso a Miembros Ocultos

Permite acceder a métodos o atributos de la clase padre que han sido sobreescritos.

```
Mac OS X icon  
void mostrar() {  
    System.out.println(super.edad);  
    super.saludar(); // Versión original del padre  
}
```



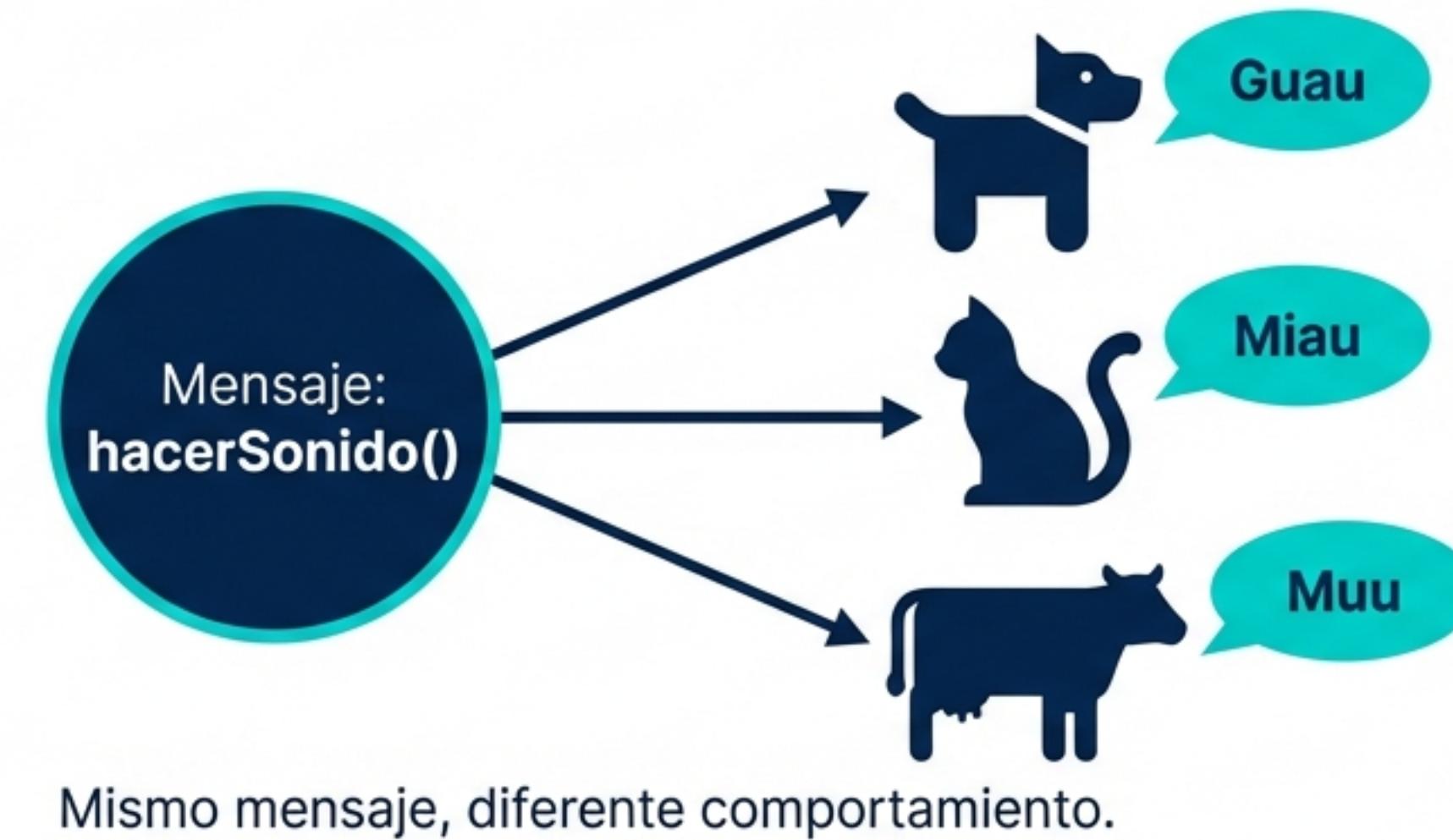
# Eficiencia en Métodos Heredados



## Explicación

- **Herencia Automática:** Los Getters y Setters se heredan. No es necesario reescribirlos.
  - **Eficiencia:** Evita el código repetitivo (Boilerplate). Solo implementa lo nuevo.

# Polimorfismo: Un nombre, muchas formas



POLIMORFISMO  
DINÁMICO



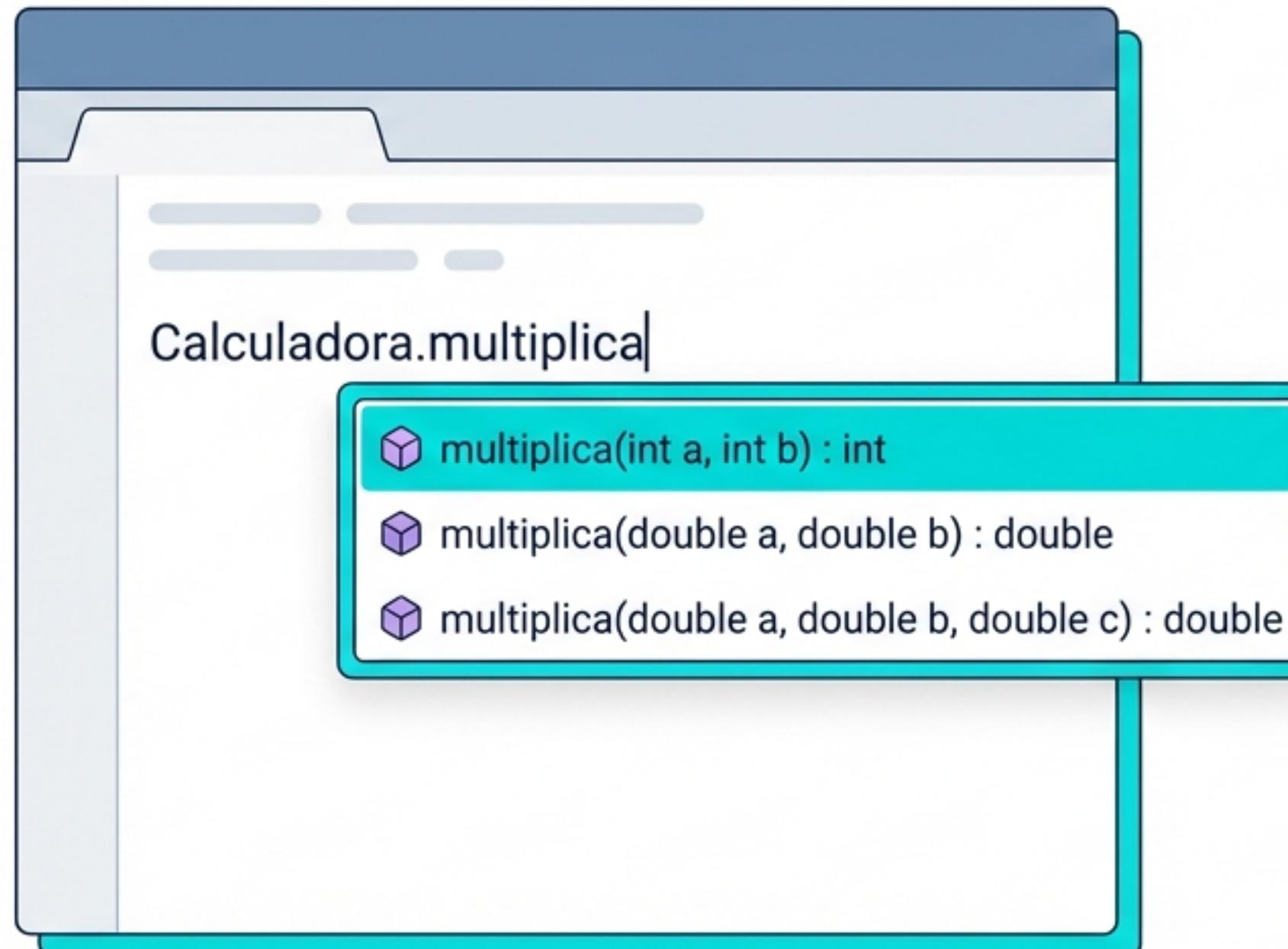
POLIMORFISMO  
ESTÁTICO

**Definición:** Capacidad de enviar el mismo mensaje a objetos de clases diferentes que comparten una raíz común.

**Dinámico:** Resuelto en ejecución (Sobreescritura).

**Estático:** Resuelto en compilación (Sobrecarga).

# Polimorfismo Estático (Sobrecarga)



## Explicación

- **Mecanismo:** Múltiples métodos con el mismo nombre pero diferentes parámetros.
- **Resolución:** El compilador decide qué método usar basándose en los argumentos (Tiempo de Compilación).
- **Ejemplo:** Clase Math o Calculadora.

# Polimorfismo Dinámico (Tiempo de Ejecución)

```
Animal animal1 = new Perro();
Animal animal2 = new Perro();
Animal animal2 = new Gato();

animal1.hacerSonido();
// Output: El perro ladra

animal2.hacerSonido();
// Output: El gato maulla
```

## Explicación Visual

Variable Reference  
(Stack)

animal1: Animal

**\*\*Binding Dinámico\*\*:** La JVM decide en tiempo de ejecución usar el método de la instancia real ('Perro'), no el de la variable ('Animal').

Object in Memory  
(Heap)

Perro Instance

hacerSonido()

**Beneficio:** Permite escribir código genérico escalable.

# Restricciones de Herencia: La palabra clave `final`

## Clases Finales

```
final class ClaseFinal
```



```
class Hija extends ClaseFinal
```

No puede tener subclases.  
Garantiza seguridad e inmutabilidad.

```
final class ClaseFinal {  
    // Contenido de la clase final  
}
```

## Métodos Finales

```
class Padre
```

```
final void metodoFinal() {  
    // Contenido del método final  
}
```

```
class Hijo extends Padre
```

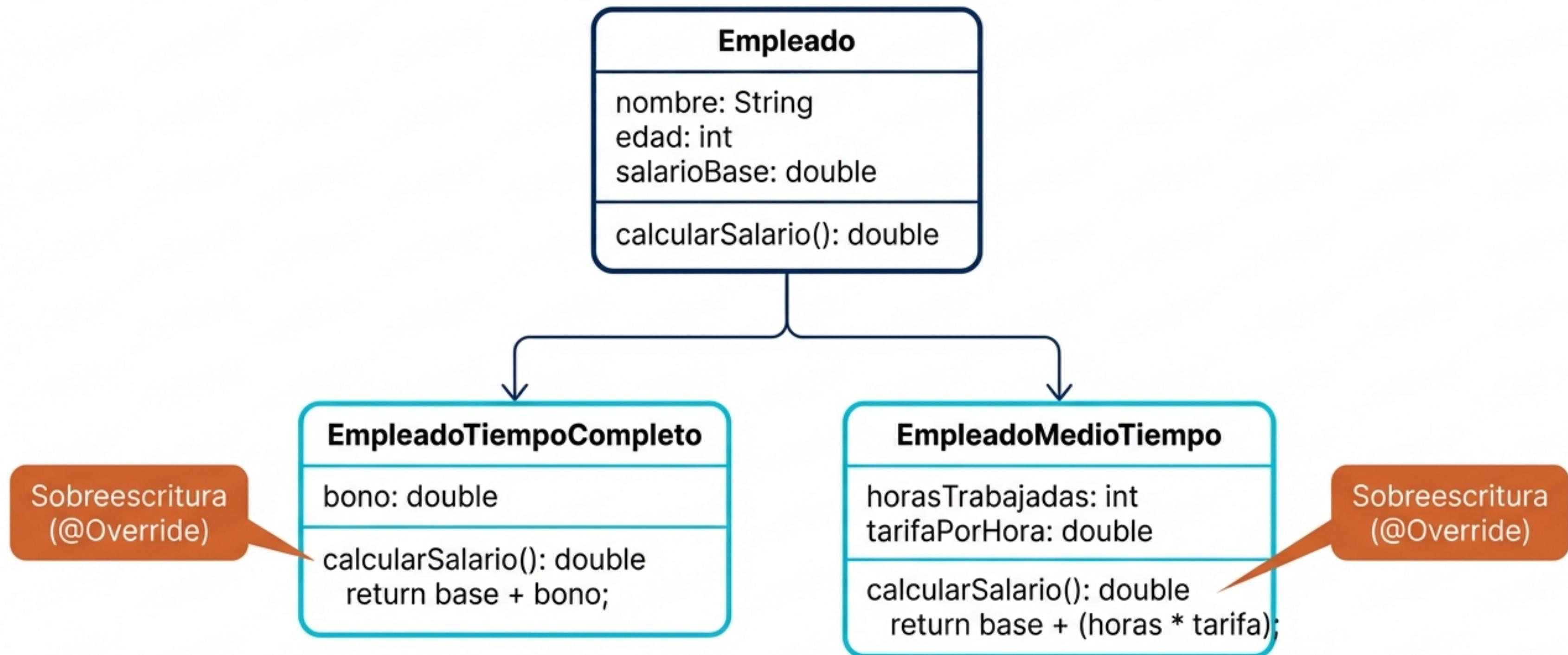
```
@Override  
void metodoFinal() { ... }
```

A large red X is positioned over the code block for final methods in subclasses, indicating that it is incorrect or prohibited.

No puede ser sobreescrito.  
Garantiza comportamiento fijo.

```
class Padre {  
    final void metodoFinal() { }  
}  
  
class Hijo extends Padre {  
    // @Override void metodoFinal() { } // Error de compilación  
}
```

# Caso Práctico Integrador: Gestión de Empleados



# Resumen y Conclusiones



La herencia aumenta la complejidad inicial pero es indispensable para crear aplicaciones escalables y fáciles de mantener.