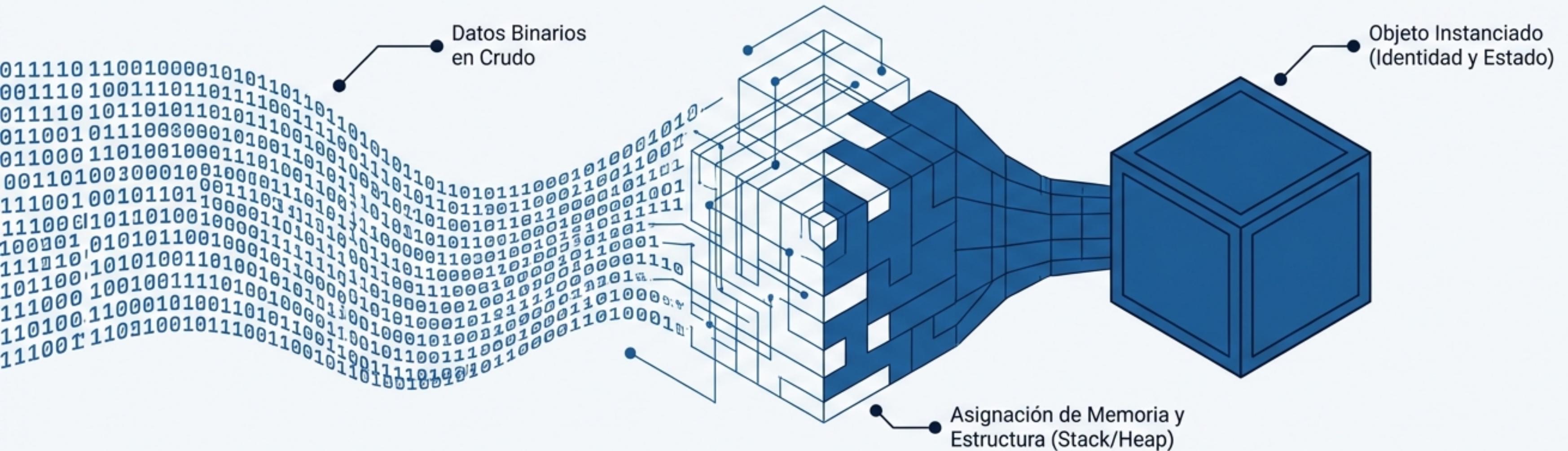
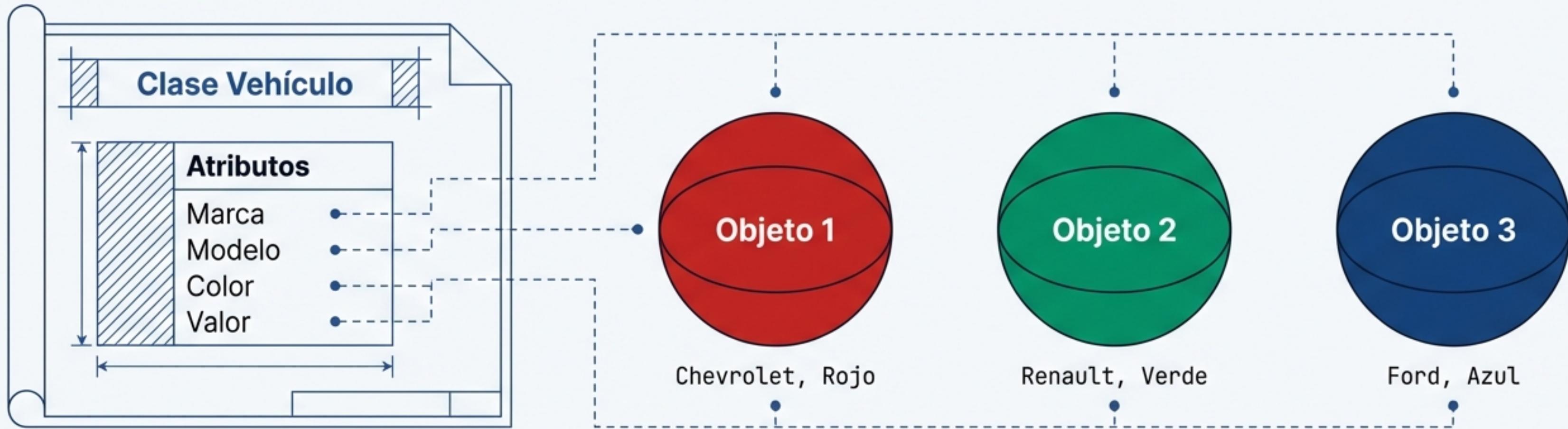


Programación Orientada a Objetos: Gestión de Clases y Objetos

Tema 8 | Identidad, Memoria y Ciclo de Vida



Anatomía de un Objeto: Identidad, Estado y Comportamiento



Identidad

Distingue un objeto de otro, incluso si sus atributos son idénticos. En Java, esto se implementa mediante direcciones de memoria.

Estado

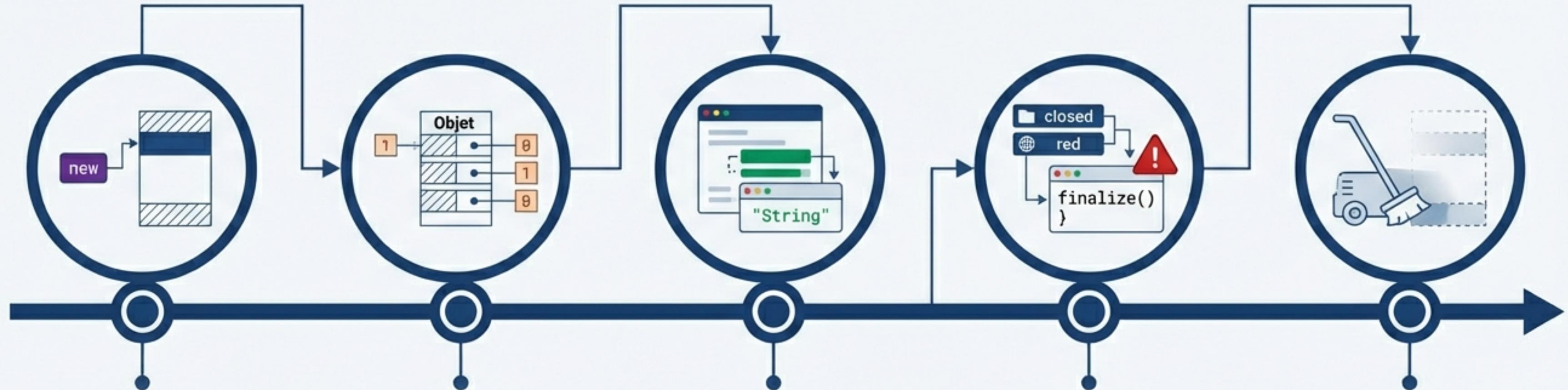
Definido por los atributos (variables miembro). Pueden ser tipos primitivos (int, double) u otros objetos.

Comportamiento

Las acciones que el objeto puede realizar (métodos). Desde constructores hasta lógica de negocio.

Encapsulamiento: Los datos suelen ser privados y solo se modifican a través de métodos.

El Ciclo de Vida Biológico del Código



Creación

Uso de "new".
Asignación de
referencia y reserva
de memoria.

Inicialización

Constructores
asignan valores
predeterminados.

Uso

Estado activo.
Ejecución de
métodos.

Finalización

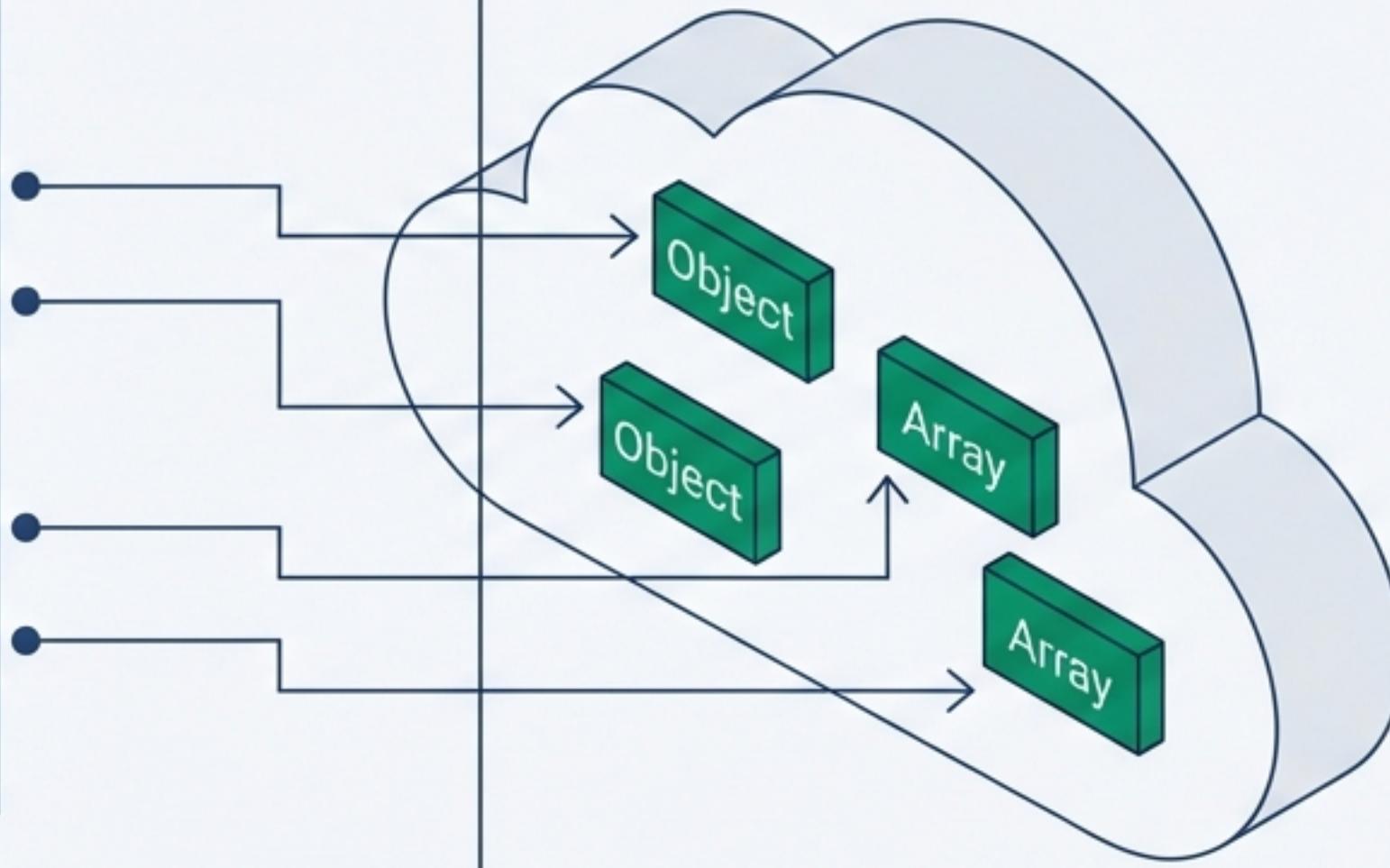
Limpieza de
recursos. Método
finalize().

Destrucción

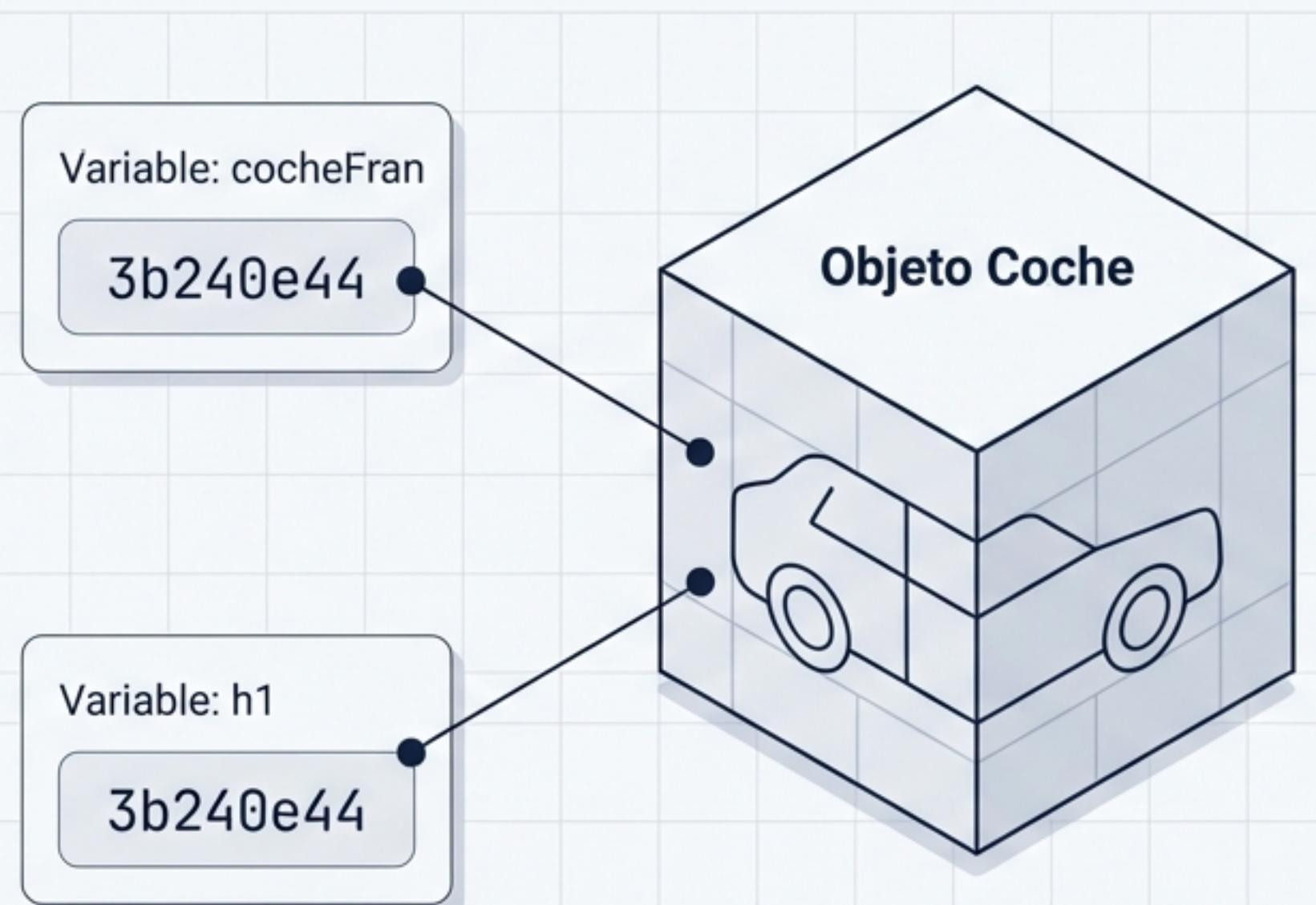
Eliminación real
por el Recolector
de Basura.

Pro-Tip: Nota: La finalización prepara el objeto para morir; la destrucción es cuando realmente desaparece de la memoria.

Arquitectura de Memoria: Stack vs. Heap

The Stack (Pila)	The Heap (Montículo)
<p>LIFO (Last In, First Out). Almacena llamadas a métodos y variables locales. Estructura ordenada.</p> 	 <p>Área dinámica donde viven los Objetos creados con "new". También almacena variables globales y estáticas.</p>
<p>El sistema operativo gestiona esto de forma transparente usando la RAM.</p>	

El Mapa no es el Territorio: Referencias



Una **referencia** en Java es un puntero, un **alias** o un **nombre** que apunta a una dirección de memoria donde vive el objeto.

```
int edad = 32; // La variable contiene  
el valor.  
Coche miCoche; // La variable almacena  
una dirección, no el coche.
```

Instanciación: La Sentencia 'new'

Coche miCoche = new Coche();

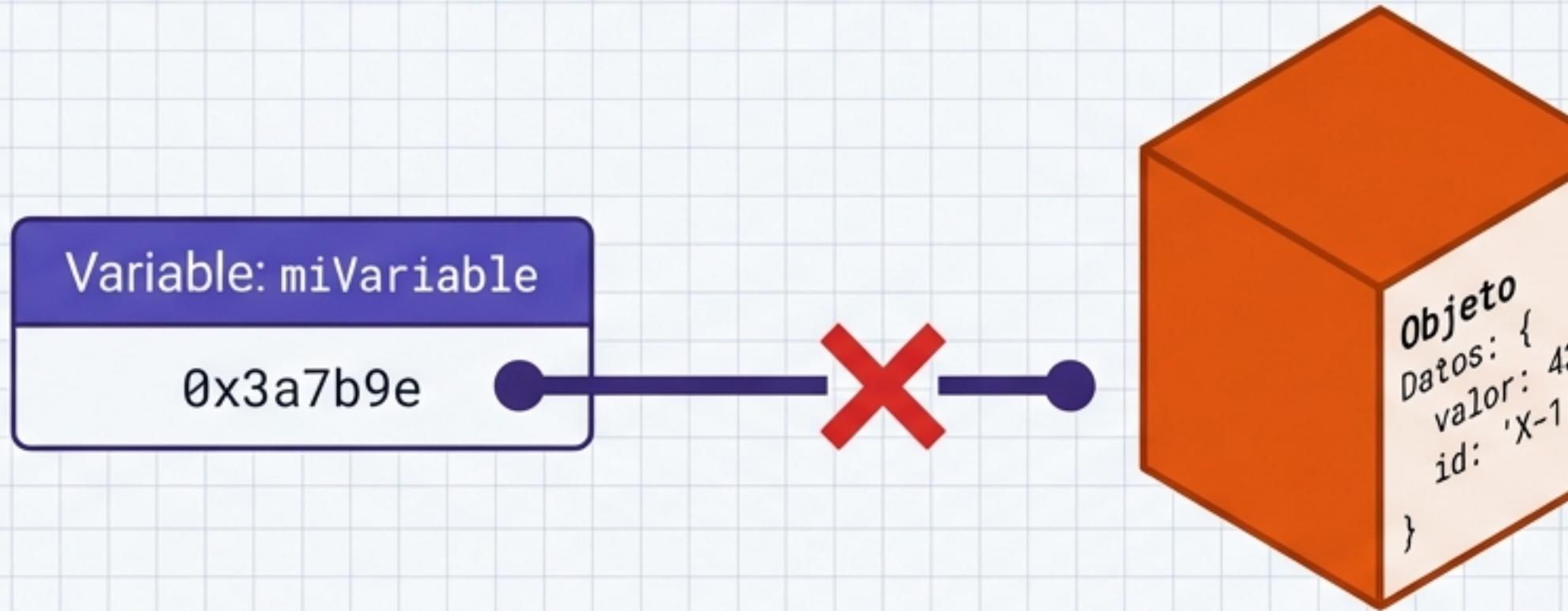


¡Peligro! Referencia Nula (null).

Indica que la variable no apunta a ningún objeto. Intentar usarla provoca el error NullPointerException.

Automatización: El Recolector de Basura (Garbage Collector)

A diferencia de C++, Java limpia su propia memoria.



Objeto Inaccesible
(Candidato a eliminación)



El GC monitorea el Heap continuamente. Si un objeto tiene 0 referencias apuntándole, es eliminado.

• Métodos: Dando Vida al Comportamiento



Constructores

Inicializan el objeto (con o sin parámetros).



Observadores (Getters)

Consultan valores de atributos.



Modificadores (Setters)

Cambian valores de atributos.



Personalizados

Lógica específica del negocio (ej. acelerar()).

Convención CamelCase: calcularPrecioTotal()
Retorno: void (nada) o return (valor único).

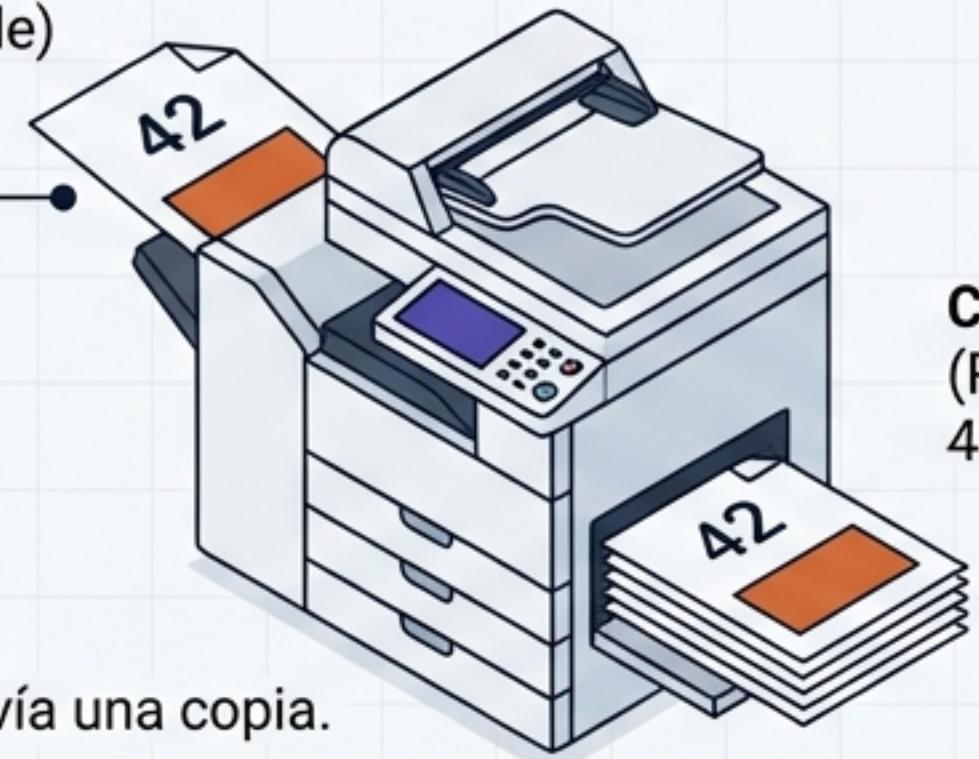
Ejemplo Personalizado:

```
public void saludo(String nombre){  
    System.out.println("Saludos "+nombre);  
}
```

Paso de Parámetros: ¿Copia o Vínculo?

Paso por Valor (Primitivos)

ORIGINAL
(Variable)
42



Se envía una copia.

COPIA
(Parámetro)
42

Paso por Referencia (Objetos)

REFERENCIA 1
(Variable)



OBJETO (Memoria)
{id: 'X-1', valor: 100}

Se envía la dirección de memoria.

REFERENCIA 2
(Parámetro)

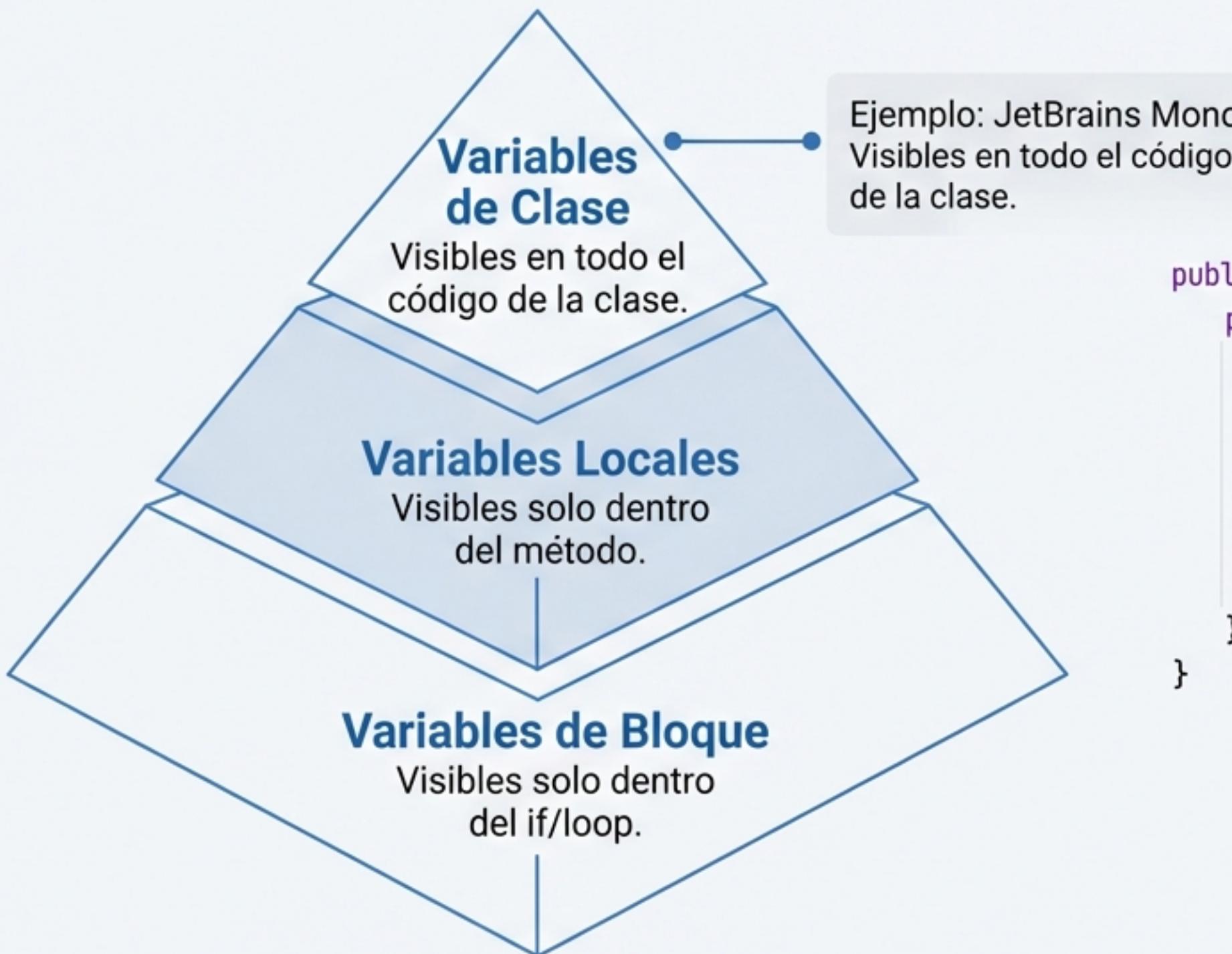


Se envía una copia. Modificar el parámetro **NO** afecta a la variable original.

Se envía la dirección de memoria. Modificar el objeto dentro del método **SÍ** afecta al original.

En Java, los objetos siempre se pasan por referencia.

Jerarquía de Visibilidad: El Ámbito (Scope)



```
public class EjemploScope {  
    public static void main(String[] args) {  
        for (int varBloque = 0; varBloque < 5; varBloque++) {  
            System.out.println("Dentro del bucle: " + varBloque);  
        }  
  
        // Error: varBloque no es accesible aquí  
        // System.out.println("Fuera del bucle: " + varBloque);  
    }  
}
```

Conflicto de Nombres: Shadowing y 'this'

Ocultación (Shadowing): Cuando una variable local tiene el mismo nombre que un atributo de clase. La local 'gana' y oculta a la externa.

El Problema (Shadowing)

```
12 public class Persona {  
13     String nombre;  
14     int edad;  
15     String apellido;  
16  
17     public void cumplirAños(){  
18         double edad = 3; // Local variable hides a field  
19         System.out.println("Tienes " + edad + " años");  
20         // ...  
21     }  
22 }  
23 }  
24 }  
25 }  
26 }
```

Variable Local oculta Atributo

La Solución ('this')

```
12 public class Persona {  
13     String nombre;  
14     int edad;  
15     String apellido;  
16  
17     public void cumplirAños(){  
18         double edad = 3; // Variable local  
19         System.out.println("Tienes " + edad + " años");  
20         // ...  
21         this.edad = 32; // Acceso explícito al atributo de la  
22     }  
23 }  
24 }  
25 }
```

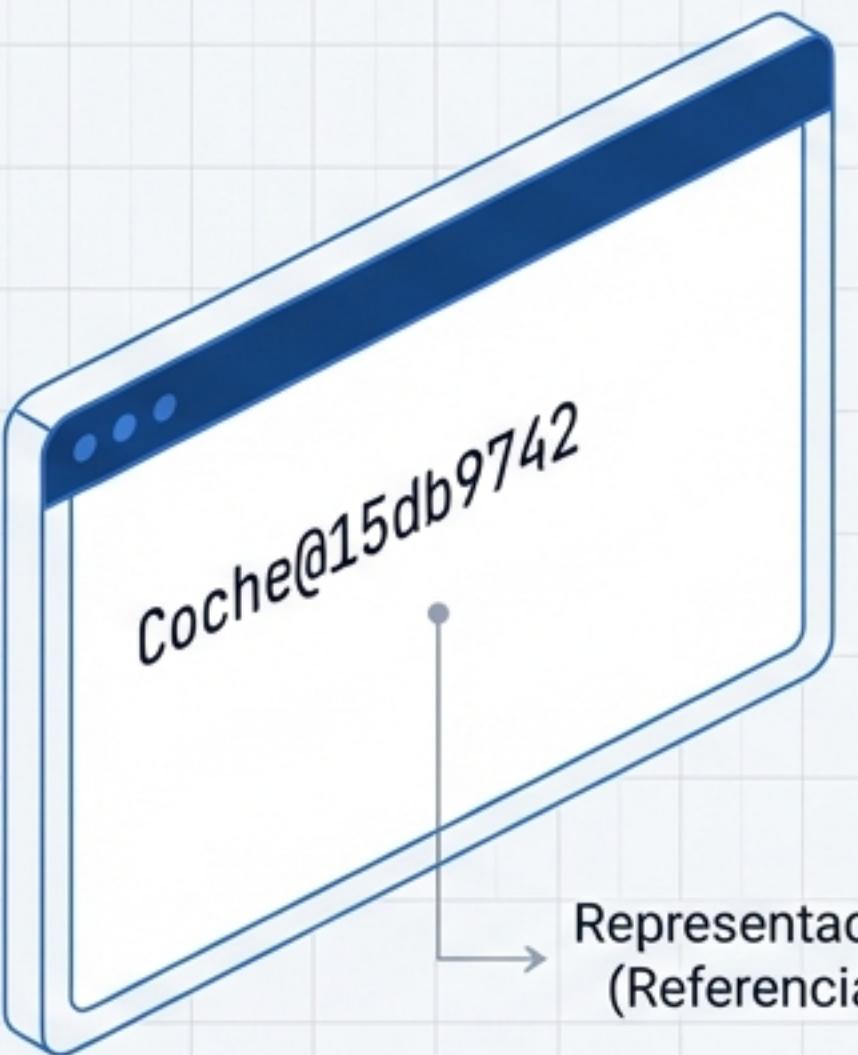
Acceso explícito al atributo de la clase.

La palabra reservada 'this' hace referencia al objeto actual, permitiendo 'saltar' la variable local.

La Presentación del Objeto: `toString()`

Convierte el estado complejo del objeto en texto legible.

Sin `toString()`



Con Override de `toString()`



Método crucial para depuración (debugging) y logs.

Documentación Profesional con Doxygen

"El código se escribe para máquinas, pero se documenta para humanos."

```
ooo

/**  
 * Realiza una operación compleja de procesamiento de datos.  
 * Esta función toma los datos de entrada y aplica una serie de transformaciones  
 * antes de devolver el resultado final.  
 *  
 * @param datosEntrada Una lista de objetos de datos sin procesar.  
 * @param opciones Configuración adicional para el procesamiento.  
 * @return Un objeto ResultadoProcesamiento con los datos transformados.  
 * @throws ExpcionProcesamiento Si ocurre un error durante la transformación de los datos.  
 * @see #validarDatos(List)  
 * @see ResultadoProcesamiento  
 */  
public ResultadoProcesamiento procesarDatos(List<Dato> datosEntrada, Opciones opciones)  
    throws ExpcionProcesamiento {  
    // Lógica de implementación...  
}
```

Describe qué recibe el método.

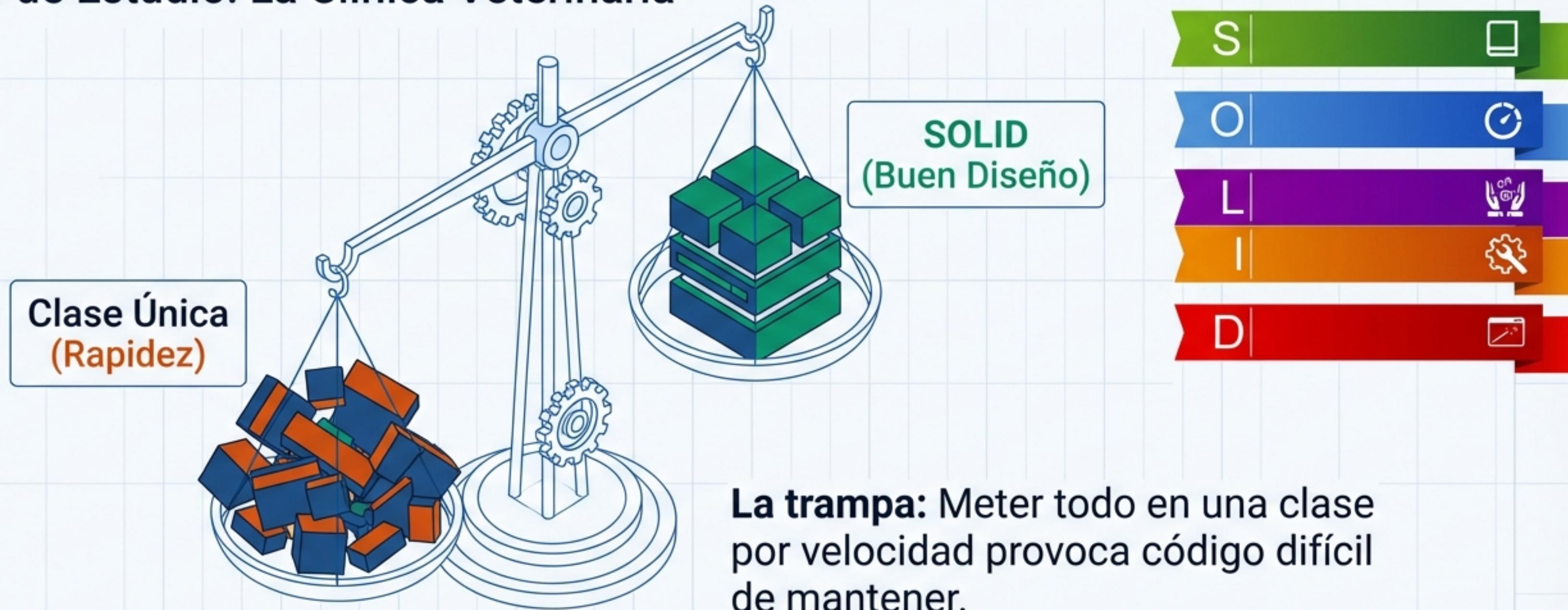
Describe qué devuelve.

Referencia a otros métodos.

Explica excepciones posibles.

Filosofía de Diseño: Rapidez vs. Arquitectura

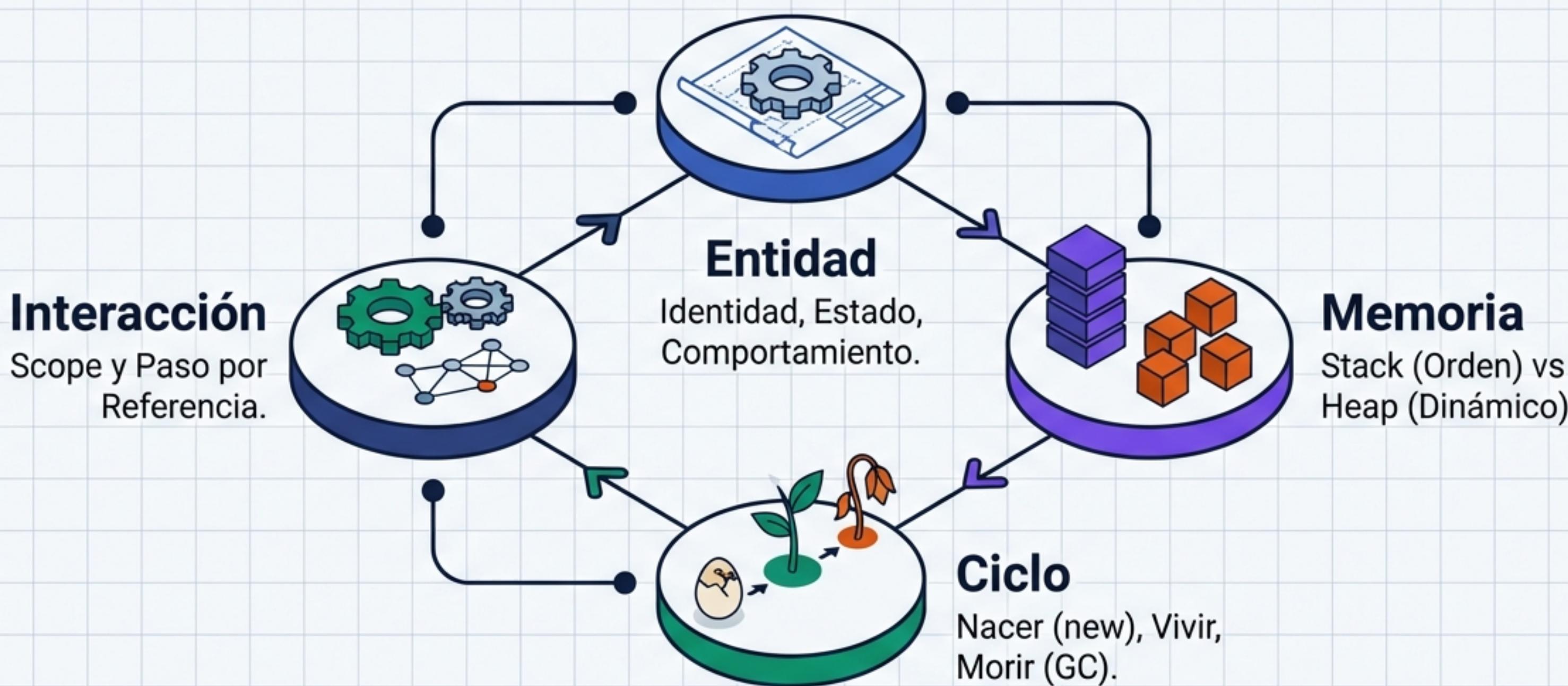
Caso de Estudio: La Clínica Veterinaria



La trampa: Meter todo en una clase por velocidad provoca código difícil de mantener.

Divide y vencerás: Un buen diseño separa responsabilidades.

Resumen: El Dominio del Objeto



Modularizar es dividir un problema complejo en subproblemas resolubles.
Un main limpio es señal de un buen diseño.