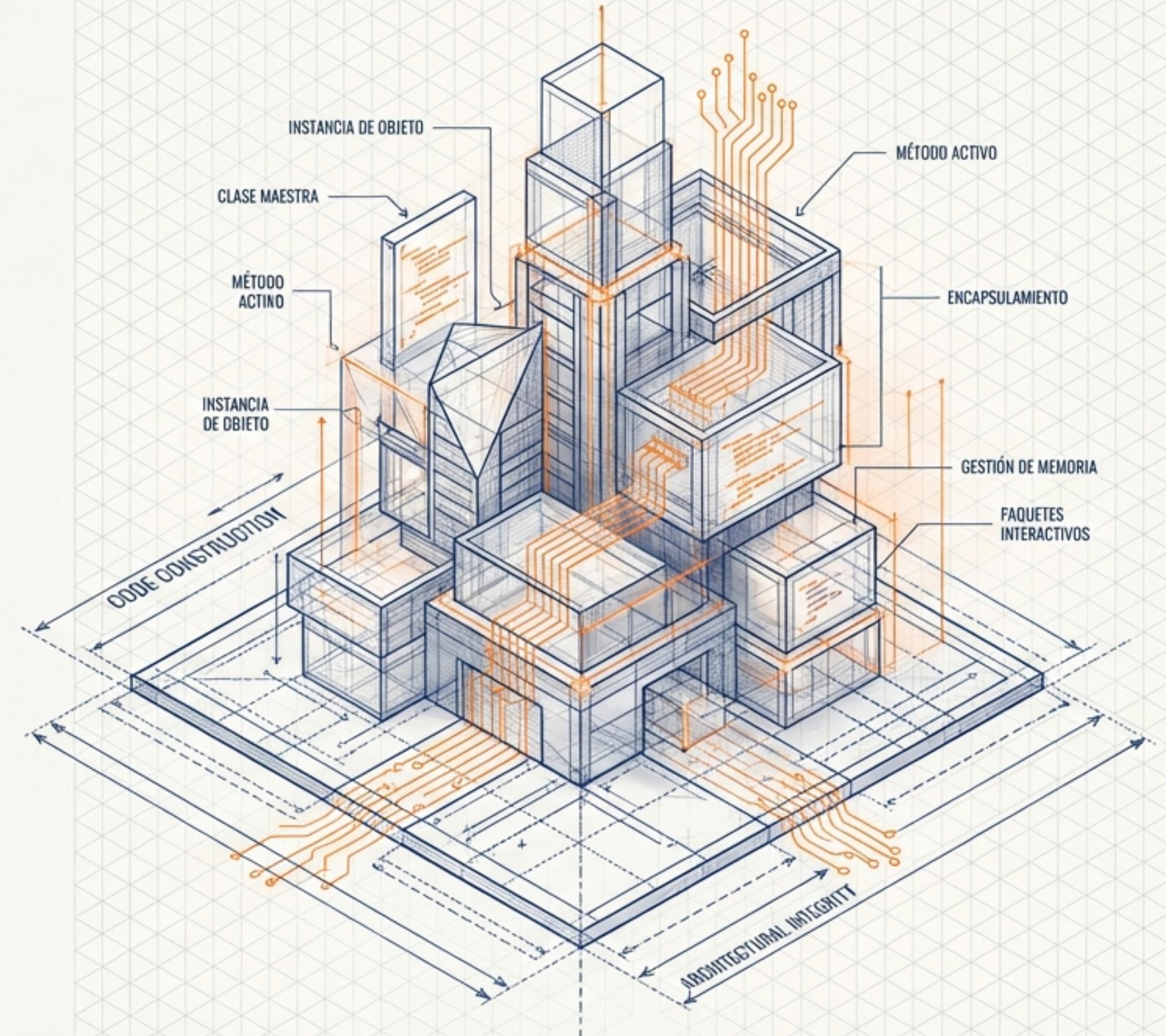


# MANIPULACIÓN DE CLASES Y OBJETOS

## GUÍA DE MAESTRÍA EN JAVA

Del concepto a la construcción:  
Una arquitectura de software robusta.



# LA DINÁMICA DE LA INTERACCIÓN

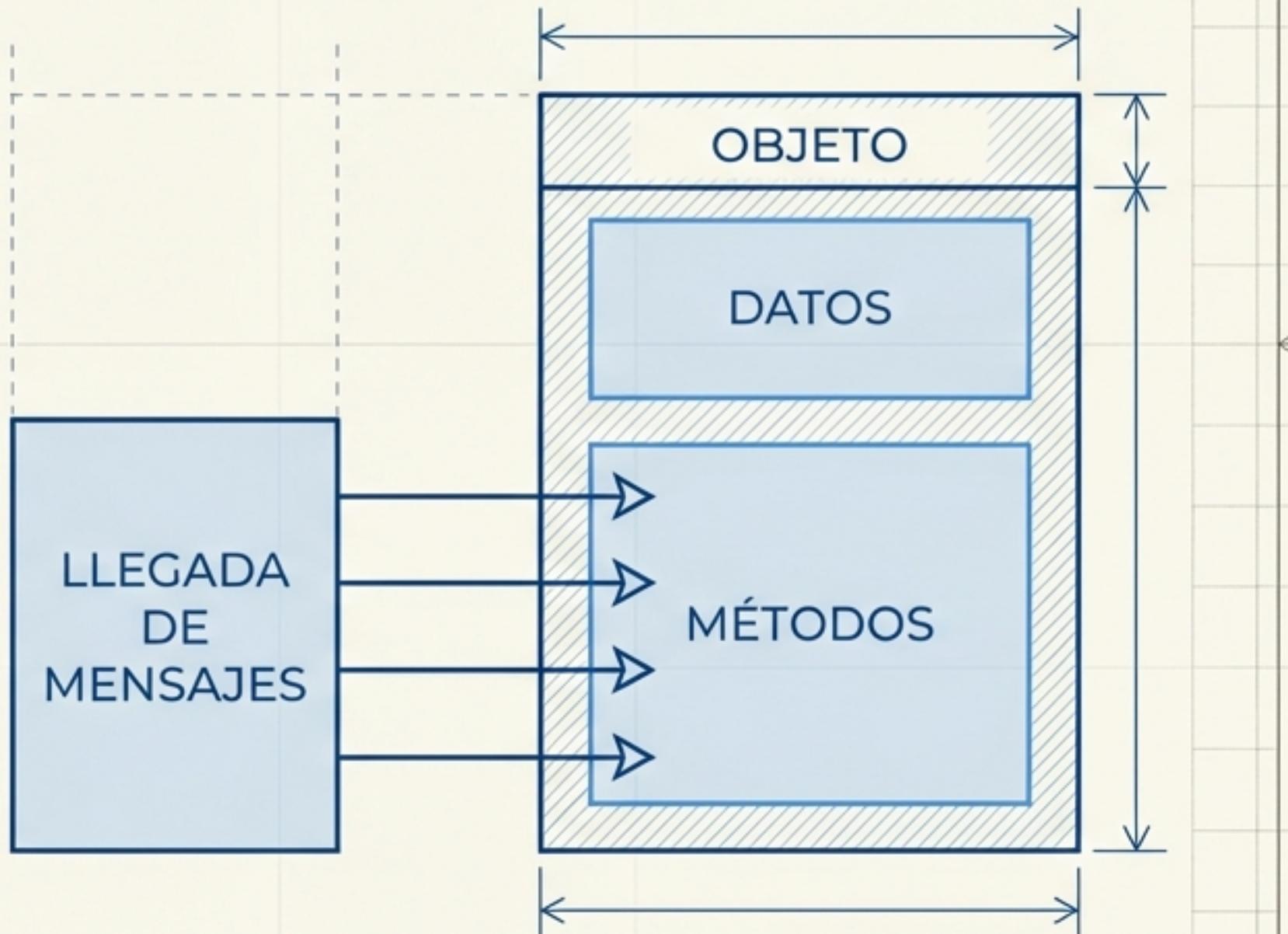
## EL PROTOCOLO DE MENSAJES

### COMUNICACIÓN:

Los objetos interactúan enviándose "mensajes" (peticiones) a través de sus métodos. Este protocolo define la interfaz pública del objeto.

### CICLO DE VIDA EN RAM:

- Creación:** Instanciación en memoria (*new*).
- Interacción:** Procesamiento de datos mediante envío/recepción de mensajes.
- Eliminación:** Liberación automática de espacio por el *Garbage Collector*.



# LO INDIVIDUAL VS. LO COLECTIVO

## ATRIBUTOS Y MÉTODOS ESTÁTICOS

### INSTANCIA (INDIVIDUAL)

Pertenecen a un objeto específico.  
Cada objeto tiene su propia copia.

Ejemplo: Matrícula de  
un coche.

### ESTÁTICO (DE CLASE)

Compartidos por *todos* los objetos.  
Si cambia uno, cambia para todos.  
No requiere `new`.

Ejemplo: Descuento  
general de la marca.

```
class Coche {  
    private String matricula;      // Individual  
    static double descuento = 1500; // Colectivo (Static)  
}  
  
// Acceso directo sin instanciar objeto  
System.out.println(Coche.descuento);
```

Compartido en  
memoria por todos  
los coches

**\*\*Bloque Estático:** Código ejecutado una sola vez al cargar la clase en memoria.

# CASO DE ESTUDIO: LA LÓGICA DEL INTERRUPTOR

GESTIÓN DE ESTADO GLOBAL VS. LOCAL

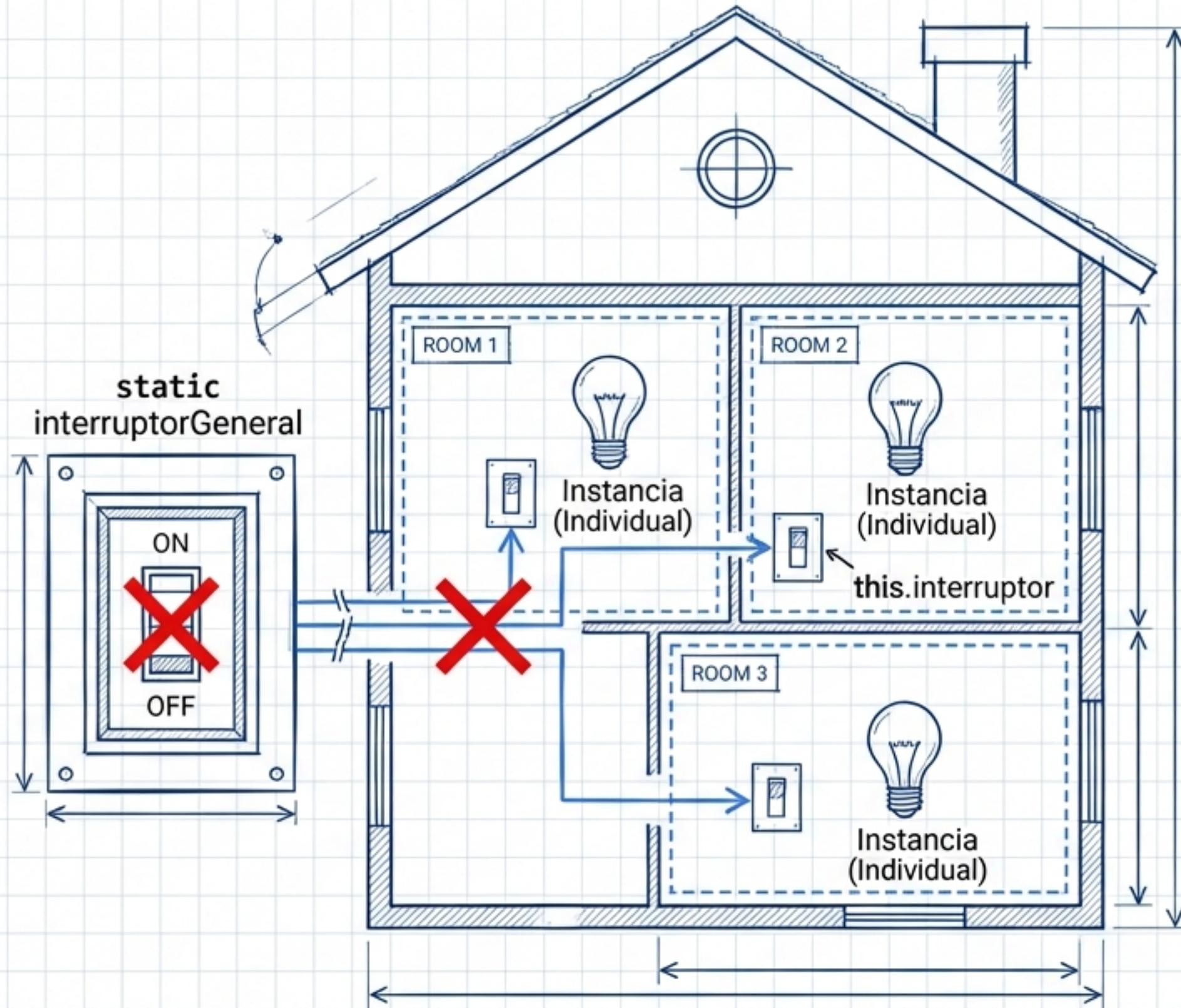
## ESCENARIO:

Modelando bombillas donde un corte general anula los interruptores individuales.

## LÓGICA DE CÓDIGO:

```
public class Bombilla {  
    public static boolean interruptorGeneral = true;  
    private boolean interruptor;  
  
    public boolean estado() {  
        // El estado real depende de AMBOS  
        return interruptorGeneral && interruptor;  
    }  
}
```

`static` acts it a master constraint



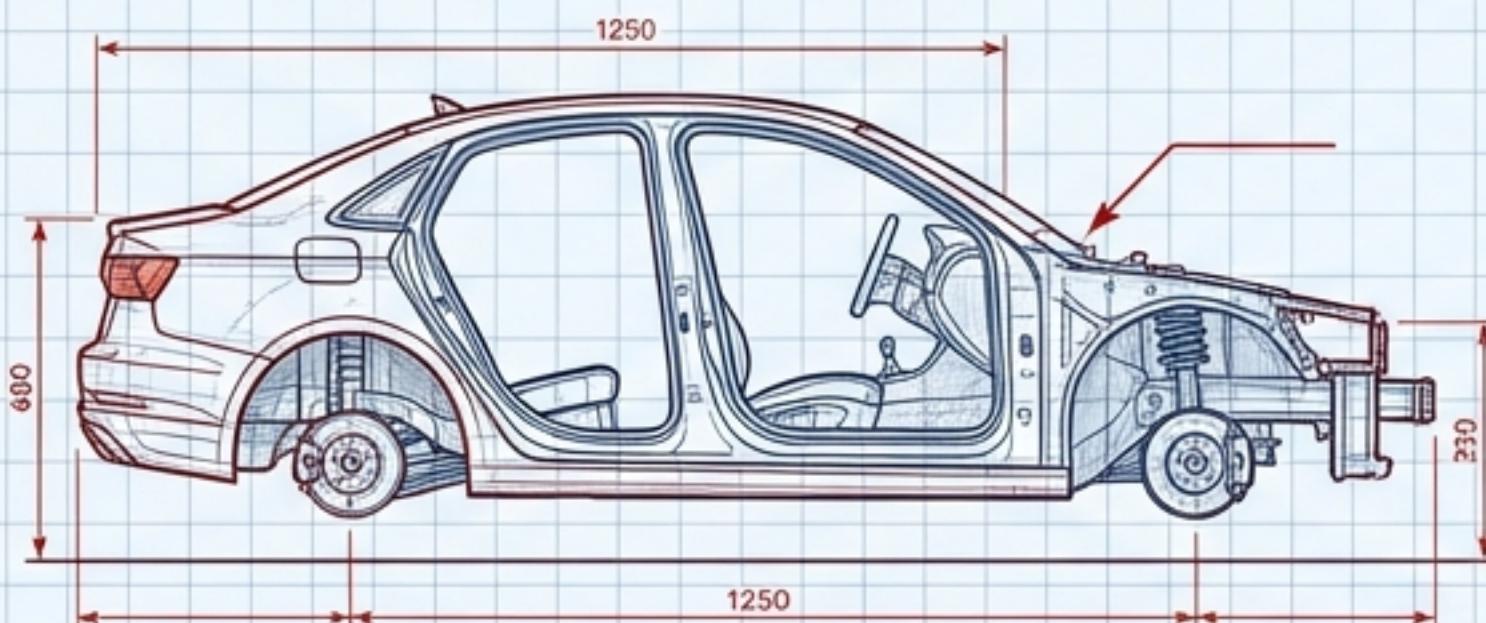
# EL ARTE DE LA CREACIÓN: CONSTRUCTORES

## INICIALIZACIÓN Y EL OPERADOR NEW

**PROPÓSITO:** Asegurar un estado válido desde el nacimiento del objeto.

**REGLA:** Mismo nombre que la clase, sin tipo de retorno.

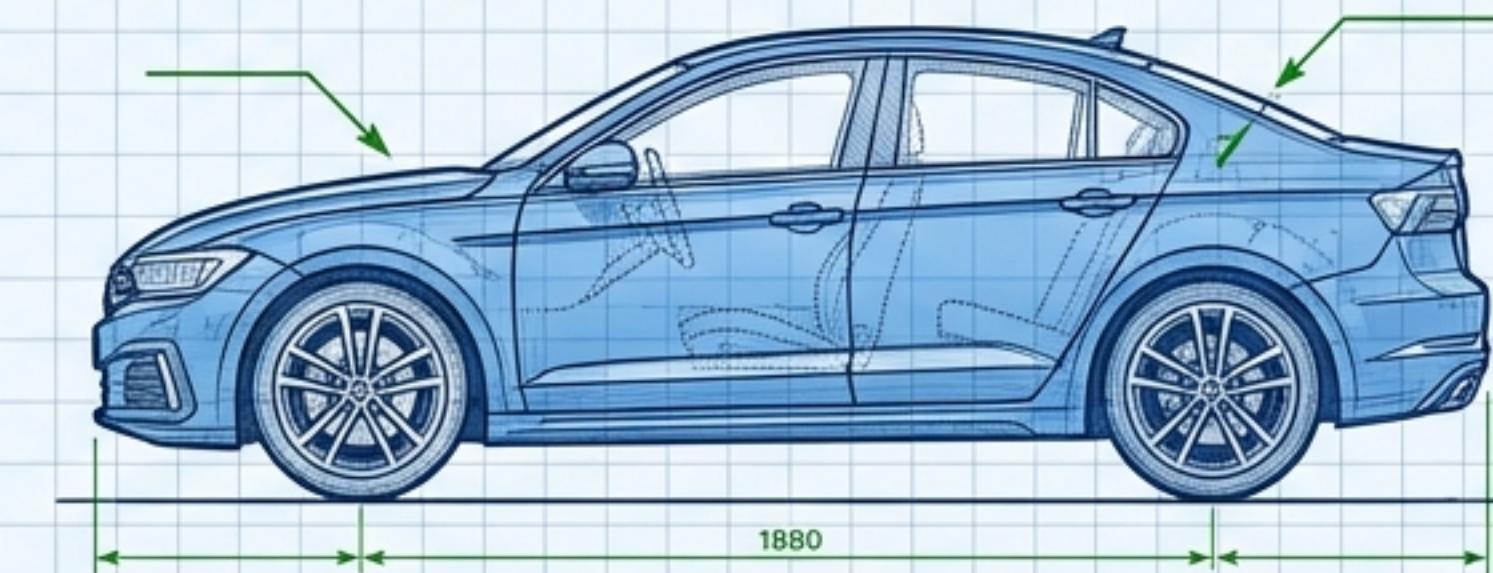
### EL PELIGRO DEL DEFAULT



Si no se define constructor,  
el compilador crea uno vacío.

Values:  
int = 0,  
String = null,  
boolean = false.

### CONSTRUCTOR PARAMETRIZADO

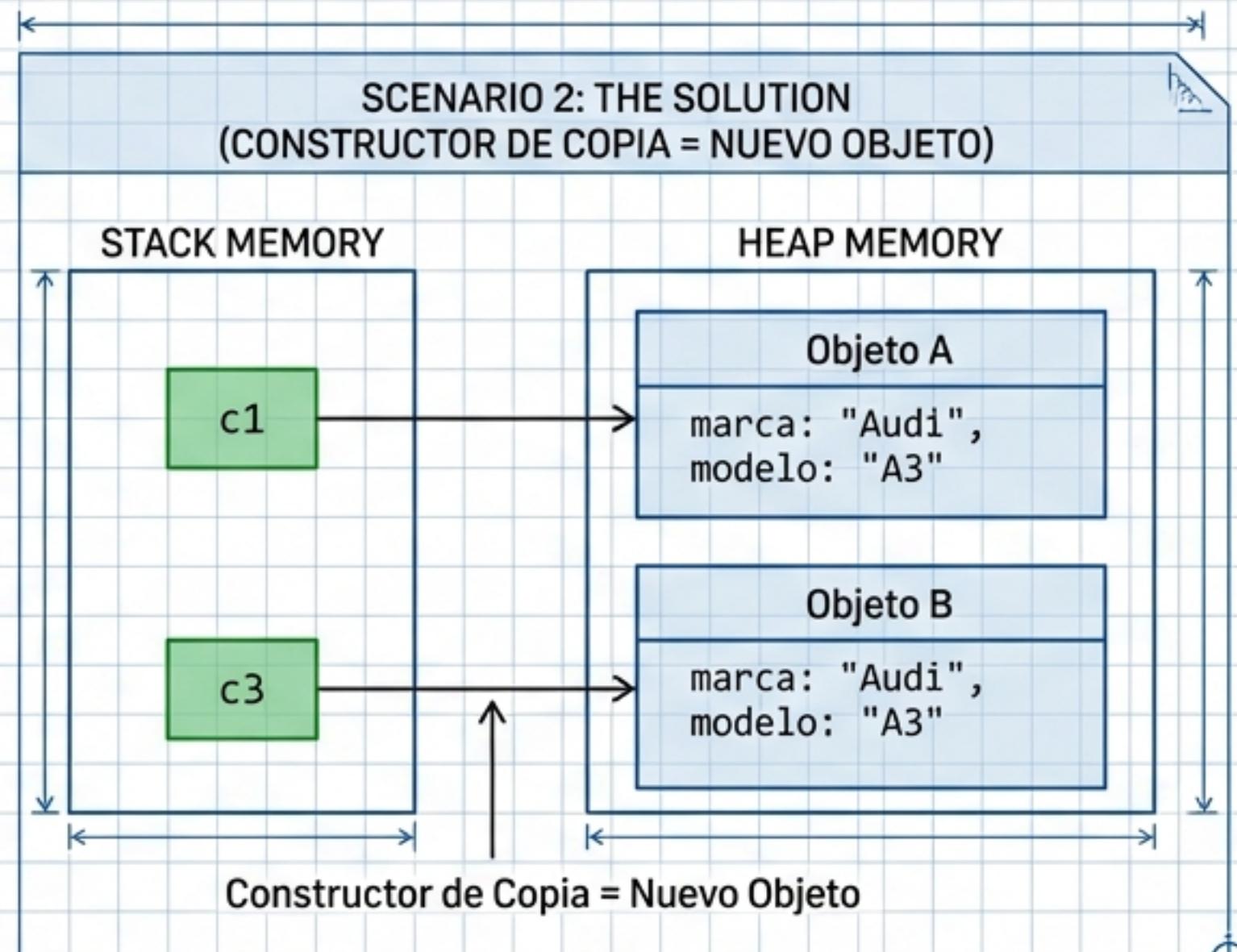
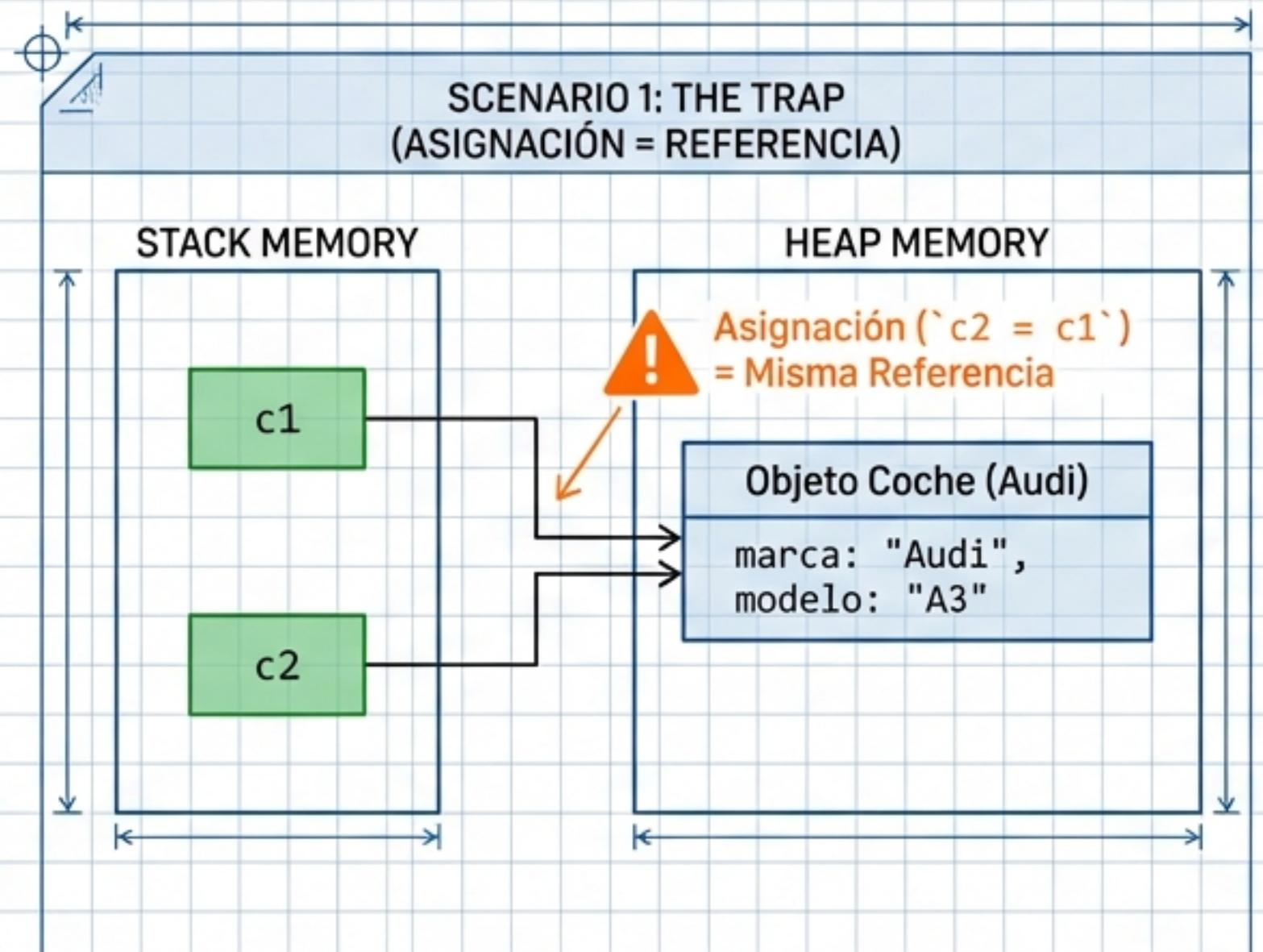


Inyección de dependencias  
al nacer. Elimina el  
constructor por defecto.

```
// Constructor Robusto
public Coche(String marca, String modelo) {
    this.marca = marca;
    this.modelo = modelo;
}
```

# LA TRAMPA DE LAS REFERENCIAS

## CLONACIÓN REAL VS. PUNTEROS



```
// Constructor de Copia
public Persona(Persona copia) {
    this.nombre = copia.nombre;
    this.edad = copia.edad;
}
```

# EFICIENCIA EN LA CONSTRUCCIÓN SOBRECARGA Y ENCADENAMIENTO CON 'THIS'

**CONCEPT:** Una clase puede tener múltiples constructores (Sobrecarga).  
'this()' permite reutilizar la lógica de uno dentro de otro.

```
// 1. Constructor Maestro (3 parámetros)
public Persona(String nom, String ape, int edad) {
    this.nombre = nom;
    this.apellido = ape;
    this.edad = edad;
}
```

CONSTRUCTOR MAESTRO  
(3 PARÁMETROS)

this(nom, ape, 0);

this(nom, ape, 0);

```
// 2. Constructor Parcial (2 parámetros)
public Persona(String nom, String ape) {
    this(nom, ape, 0); // <-- Llama al Maestro
}
```

CONSTRUCTOR PARCIAL  
(2 PARÁMETROS)

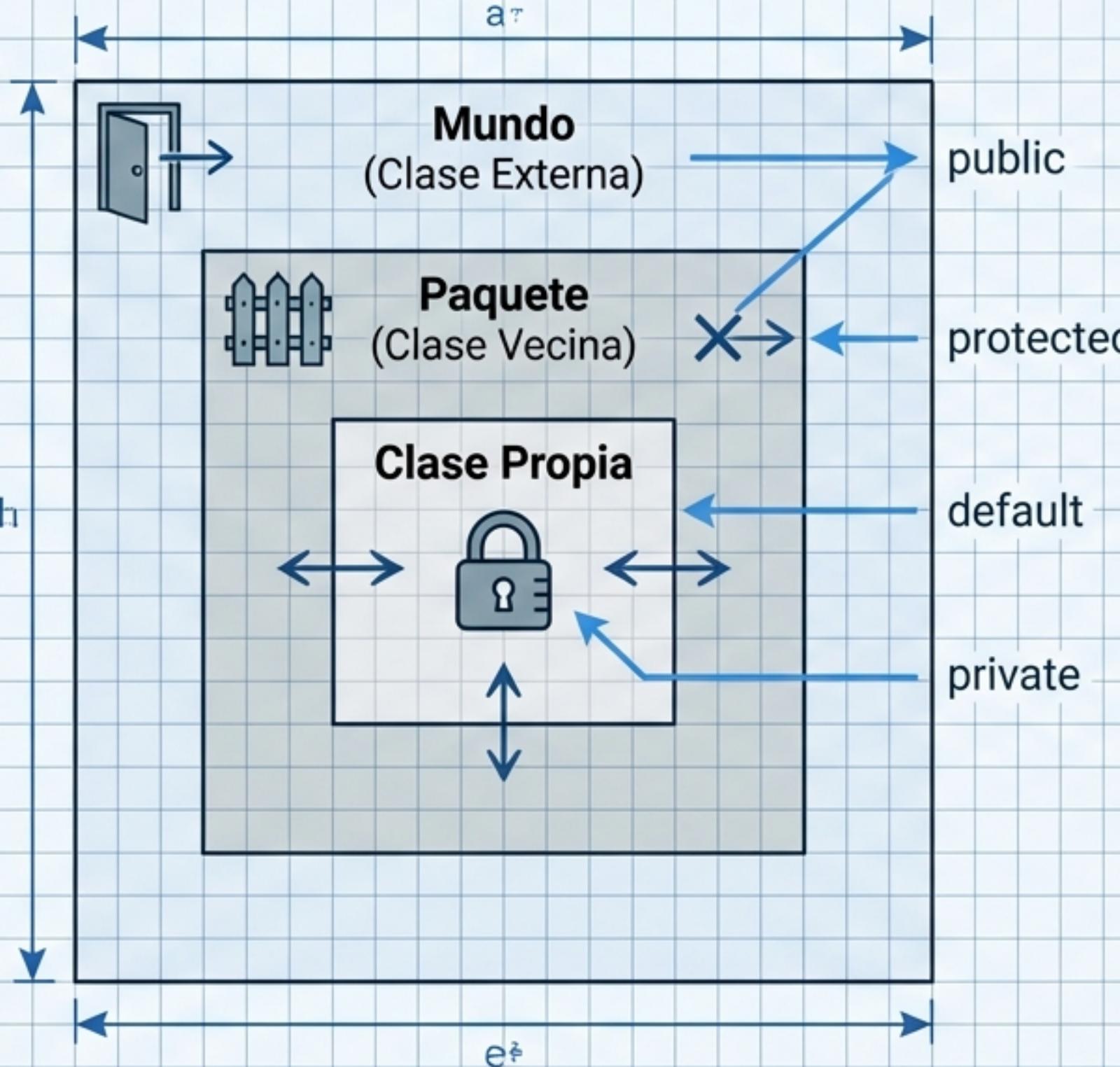
CONSTRUCTOR MÍNIMO  
(1 PARÁMETRO)

this(nom, "Desconocido");

```
// 3. Constructor Mínimo (1 parámetro)
public Persona(String nom) {
    this(nom, "Desconocido"); // <-- Llama al Parcial
}
```

# LA FORTALEZA: VISIBILIDAD

MODIFICADORES DE ACCESO Y SEGURIDAD



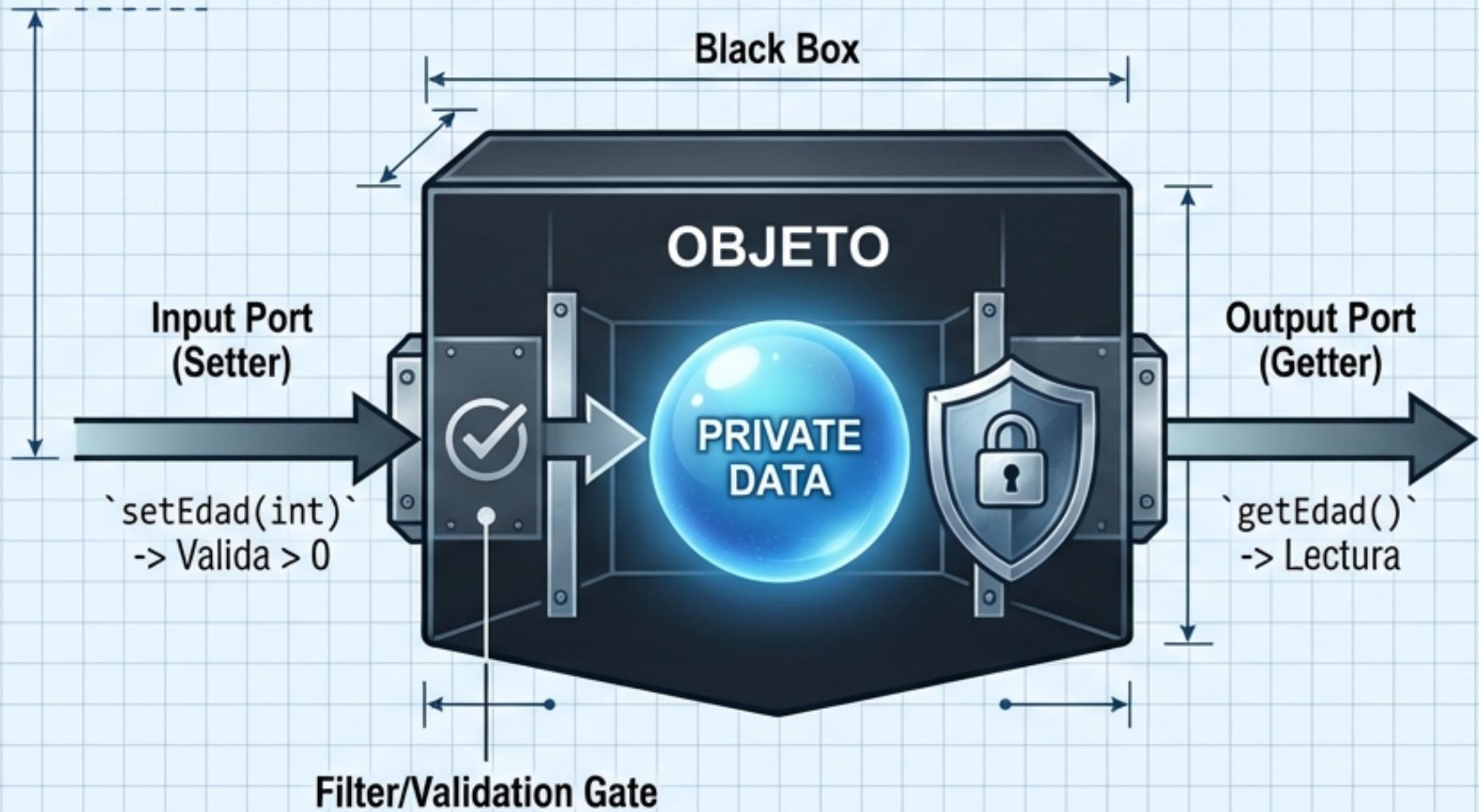
THE CHEAT SHEET			
Modificador	Propia Clase	Clase Vecina (Paquete)	Clase Externa (Mundo)
<b>public</b>	✓ SÍ	✓ SÍ	✓ SÍ
<b>protected</b>	✓ SÍ	✓ SÍ	✗ NO (salvo herencia)
<b>(default)</b>	✓ SÍ	✓ SÍ	✗ NO
<b>private</b>	✓ SÍ	✗ NO	✗ NO



‘private’ es la caja fuerte.  
‘public’ es la ventanilla de atención.

# GUARDIANES DE LOS DATOS

## ENCAPSULAMIENTO CON GETTERS Y SETTERS



### CODE COLUMN

```
private int edad; // Oculto  
  
// Getter (Observador)  
public int getEdad() {  
    return edad;  
}  
  
// Setter (Modificador con Validación)  
public void setEdad(int edad) {  
    this.edad = edad;  
}
```



Architectural Precision: Data Integrity through Encapsulation.

# ESTRUCTURA DE DATOS: ENUMERADOS

## CONSTANTES CON TIPADO SEGURO



**DEFINICIÓN:** Tipos de datos especiales para conjuntos fijos de constantes (Días, Meses, Estados).

### Antes (Constantes enteras):

```
public static final int LUNES = 1;  
public static final int MARTES = 2;
```

→ **✗ Inseguro**, permite `dia = 99`

### Ahora (Enums):

```
enum Dia { LUNES, MARTES, MIERCOLES... }  
Dia hoy = Dia.LUNES;
```

→ **✓ Seguro**, legible, ideal para `switch`



### CAPACIDADES DE CLASE:

En Java, los `enum` son clases completas. Pueden tener:

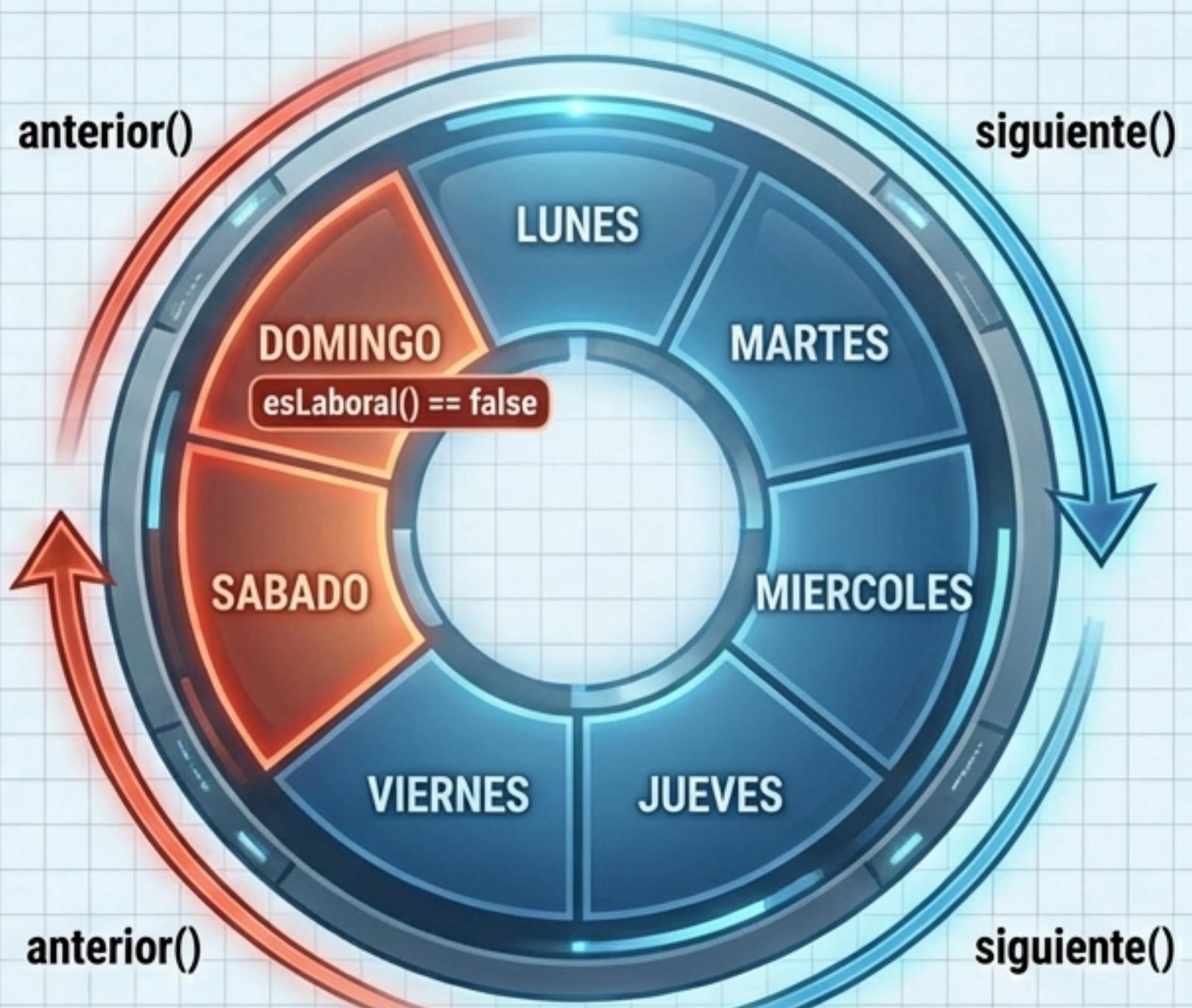
- Atributos propios (ej. abreviatura).
- Constructores privados.
- Métodos de lógica interna.



# CASO DE ESTUDIO: DÍAS DE LA SEMANA

## LÓGICA ENCAPSULADA EN EL DATO

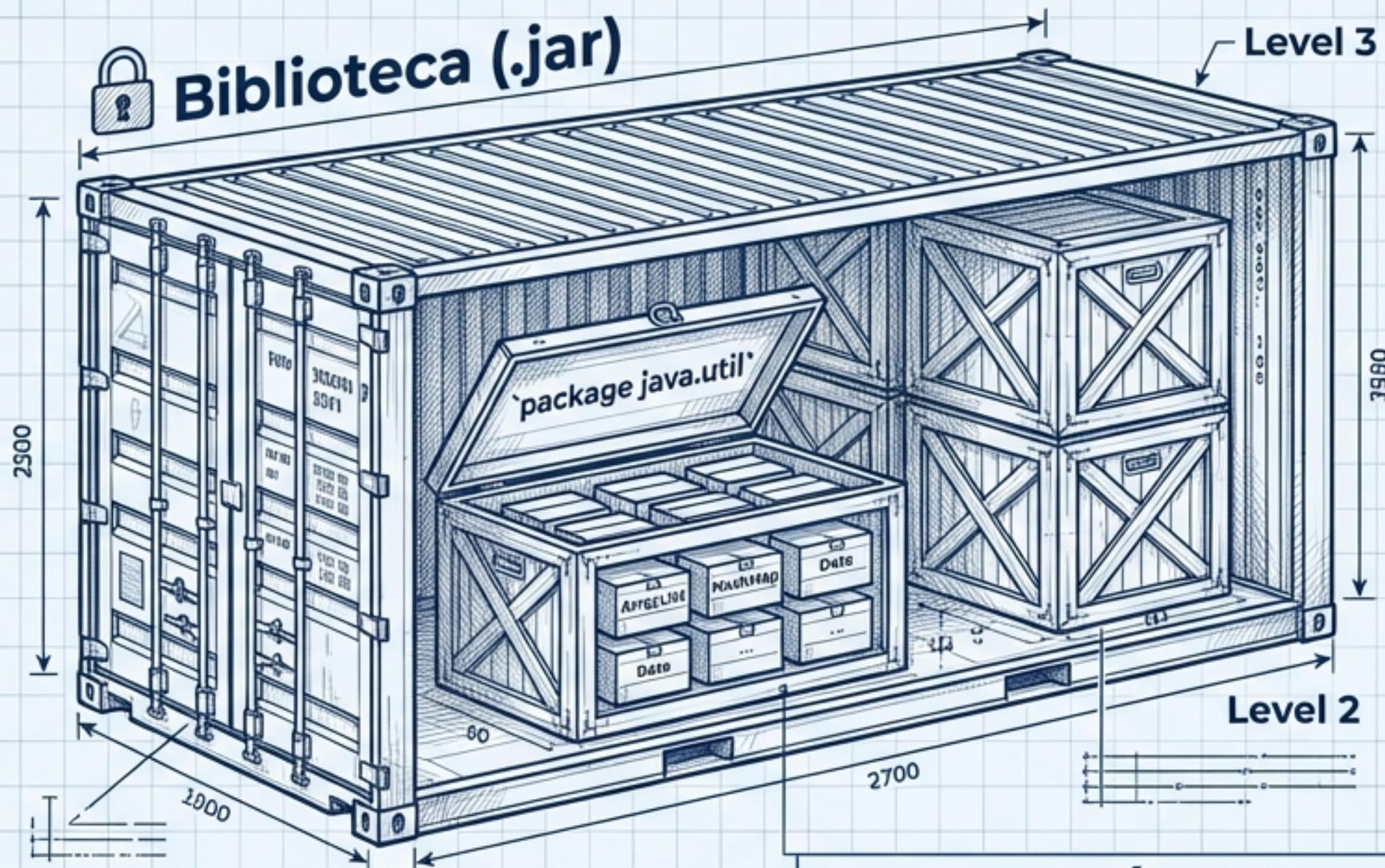
```
public enum DiaSemana {  
    LUNES, MARTES, MIERCOLES,  
    JUEVES, VIERNES,  
    SABADO, DOMINGO;  
  
    // Método para obtener el siguiente día de la semana  
    public DiaSemana siguiente() {  
        // Aritmética modular para circularidad  
        return values()[(ordinal() + 1) % 7];  
    }  
  
    // Método para verificar si es un día laboral  
    public boolean esLaboral() {  
        return this != SABADO && this != DOMINGO;  
    }  
}
```



Architectural Precision: Encapsulating Logic within Data.

# ORGANIZACIÓN A GRAN ESCALA

## BIBLIOTECAS (.JAR) Y PAQUETES



**FILOSOFÍA:** Resolver un problema una vez, empaquetarlo y reutilizarlo.

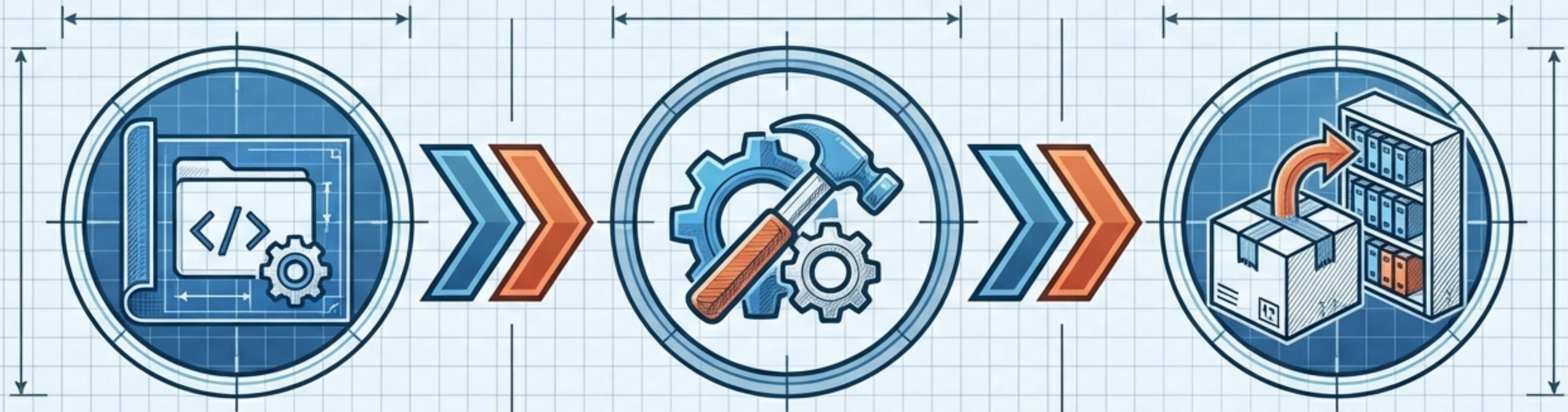
### IMPORTACIÓN:

`import java.util.ArrayList;` (Específico - Recomendado)  
`import java.util.*;` (Comodín - Trae todo el paquete)



# FLUJO DE TRABAJO: CREACIÓN DE BIBLIOTECAS

## PROCESO EN NETBEANS / IDE



### PROYECTO

Tipo 'Java Class Library'.  
Estructura estándar sin  
'main' obligatorio.

### CONSTRUCCIÓN (BUILD)

Ejecutar 'Clean and Build'.  
Genera el fichero .jar en  
la carpeta 'dist/'.

### IMPORTACIÓN

Add JAR/Folder. El código  
externo ahora es accesible  
vía 'import'.

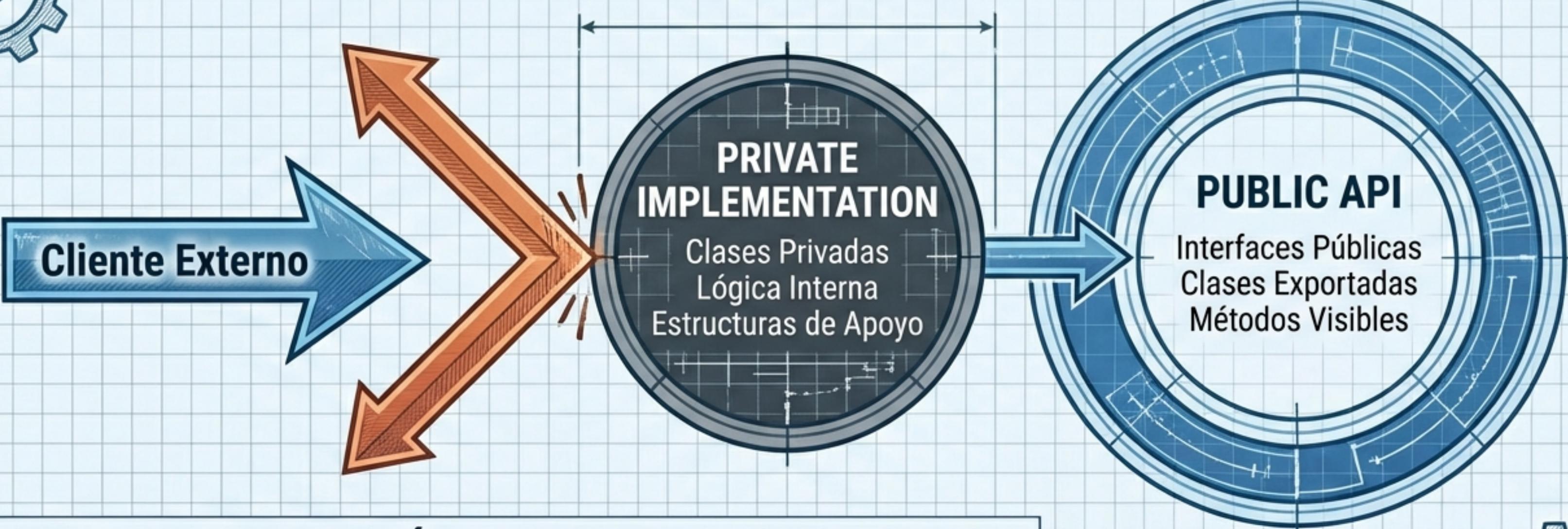
- Add Project...
- Add Library...
- Add JAR/Folder...

Properties



# ENCAPSULAMIENTO ARQUITECTÓNICO

## LA BIBLIOTECA COMO CAJA NEGRA



### PRINCIPIO DE OCULTACIÓN:

Expone funcionalidad (API), oculta complejidad. Permite cambiar el código interno sin romper los proyectos que usan la biblioteca.



# LOS 5 PILARES DE LA ARQUITECTURA

## RESUMEN DE MAESTRÍA

### SOFTWARE ROBUSTO

**COMUNICACIÓN**  
Paso de mensajes  
y ciclo de vida.

**IDENTIDAD**  
Instancia (único)  
vs. Estático  
(compartido).

**CICLO DE VIDA**  
Constructores,  
copias y referencias.

**SEGURIDAD**  
Visibilidad,  
Getters y Setters.

**ORGANIZACIÓN**  
Enums, Paquetes  
y Bibliotecas.

"El código bien organizado es fácil de leer, seguro de modificar y eficiente para ejecutar."

