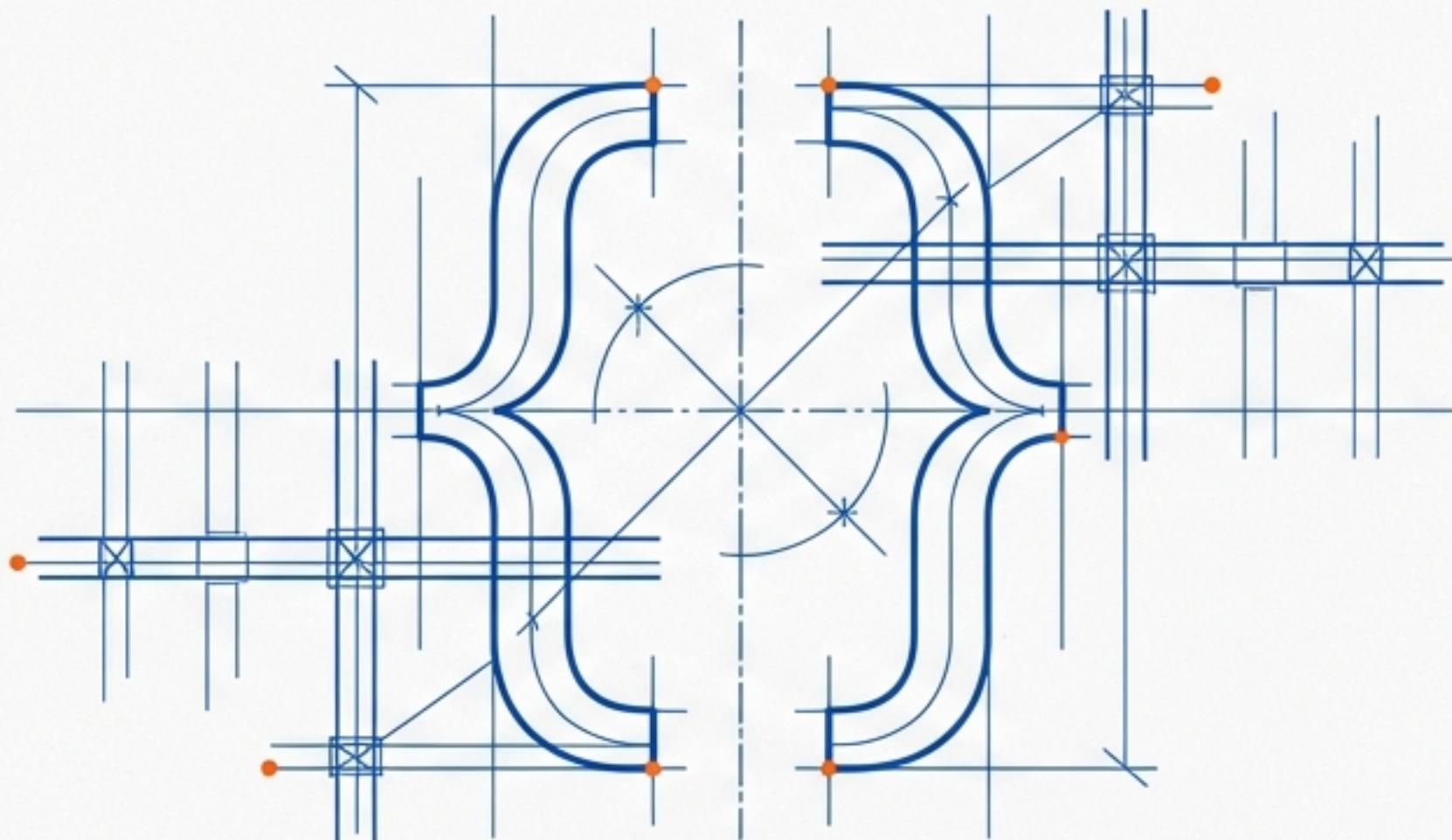


De los Cimientos al Código: Tu Guía Definitiva de Java



Un repaso detallado de nuestra sesión para
solidificar tus conocimientos.

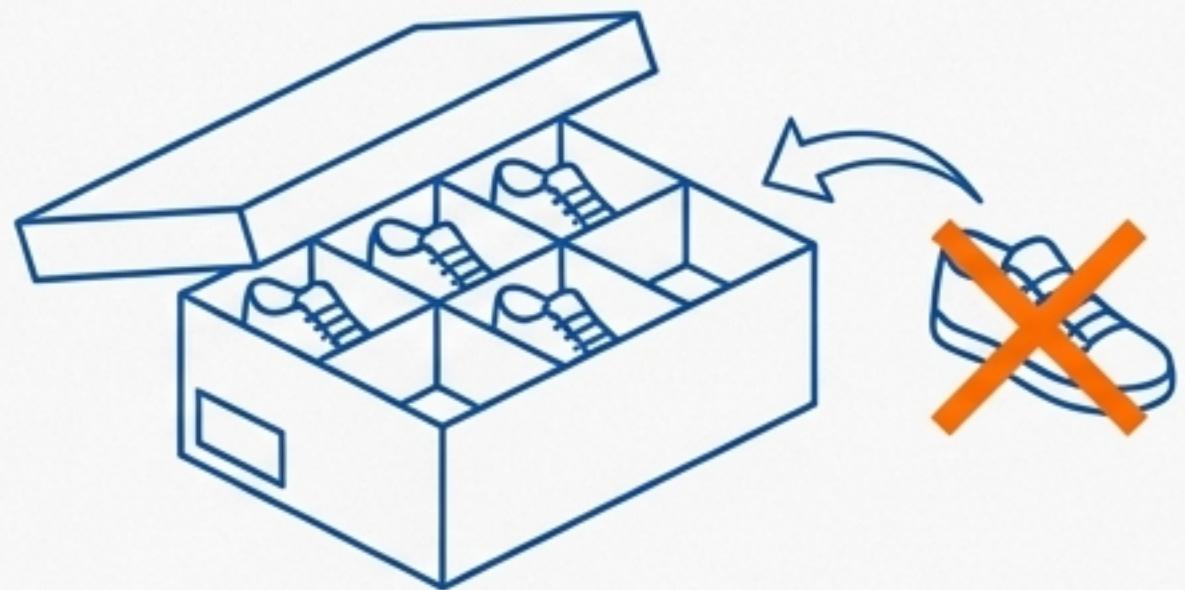
Nuestro Plan de Construcción



Cada concepto se apoya en el anterior. Seguiremos este orden para construir una base de conocimiento fuerte y sin fisuras.

La Caja de Zapatos vs. El Acordeón: Elige tu Contenedor

Array (La Caja Fija)

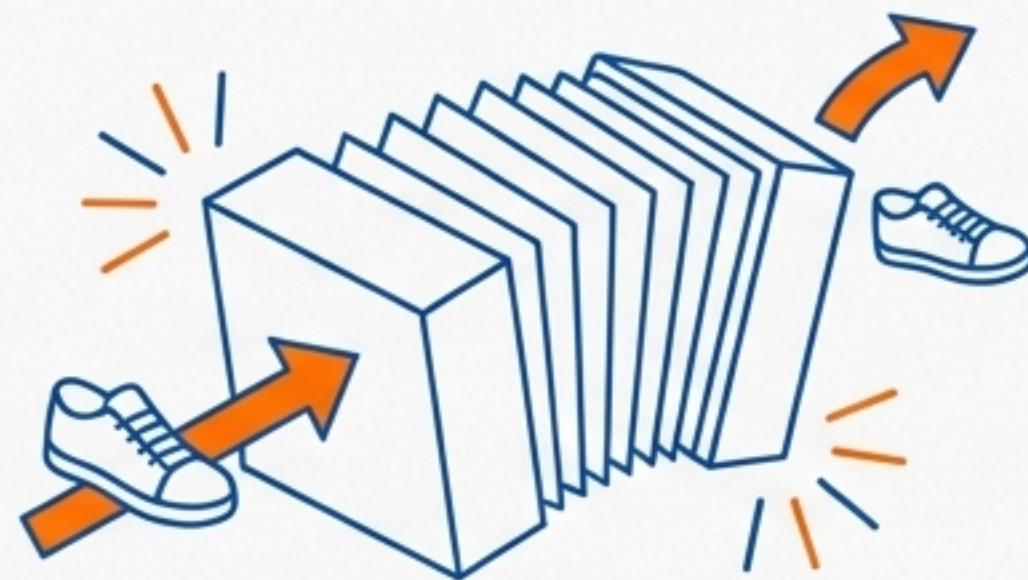


Es un espacio reservado en memoria. Si asignas espacio para 6 elementos, no puedes meter un séptimo.

- Tamaño fijo, definido en su creación.
- Eficiente en el uso de memoria estática.
- Menos flexible.

```
// Creas una "caja" con exactamente 6 espacios.  
String[] miArrayFijo = new String[6];
```

ArrayList (El Contenedor Dinámico)



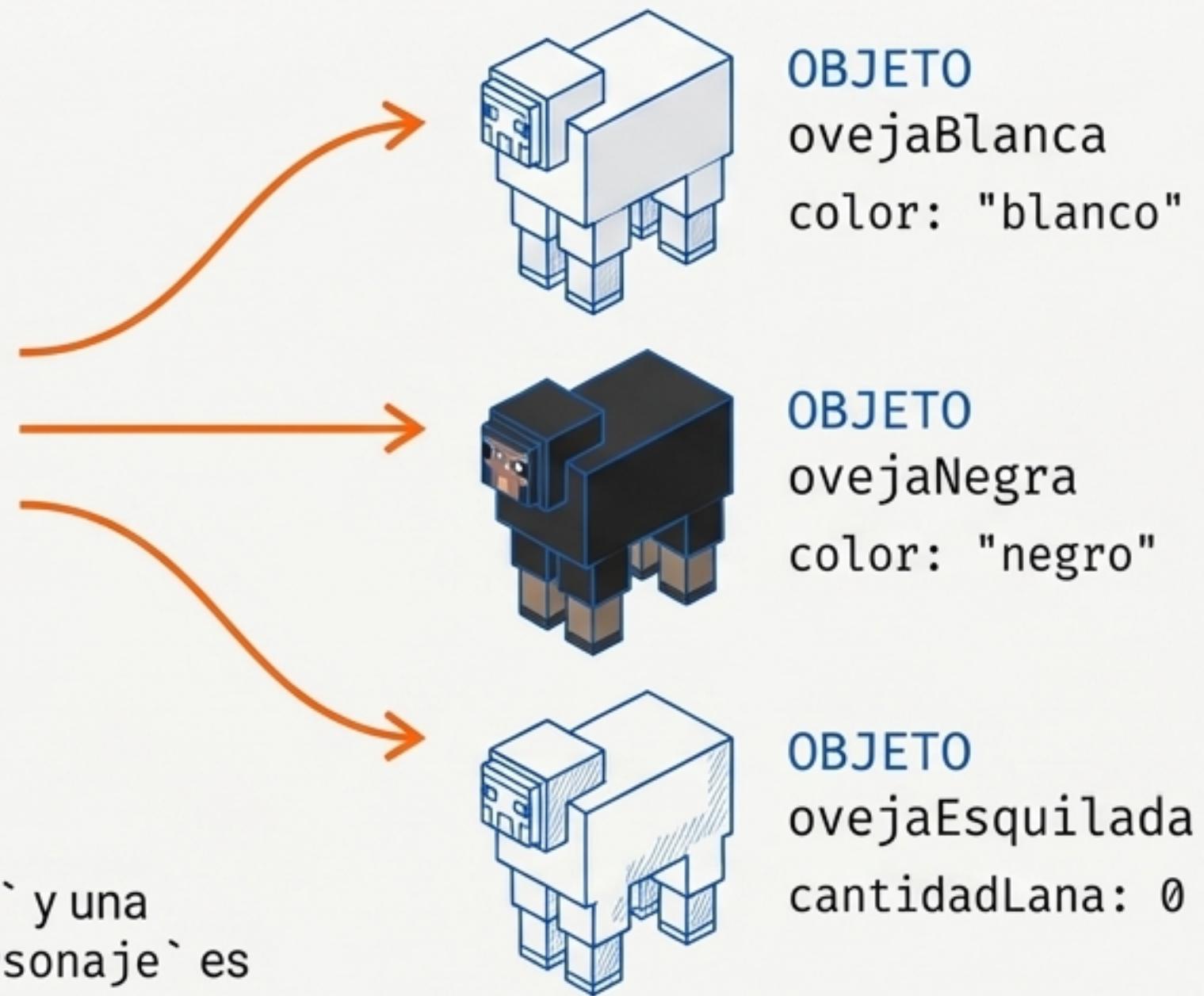
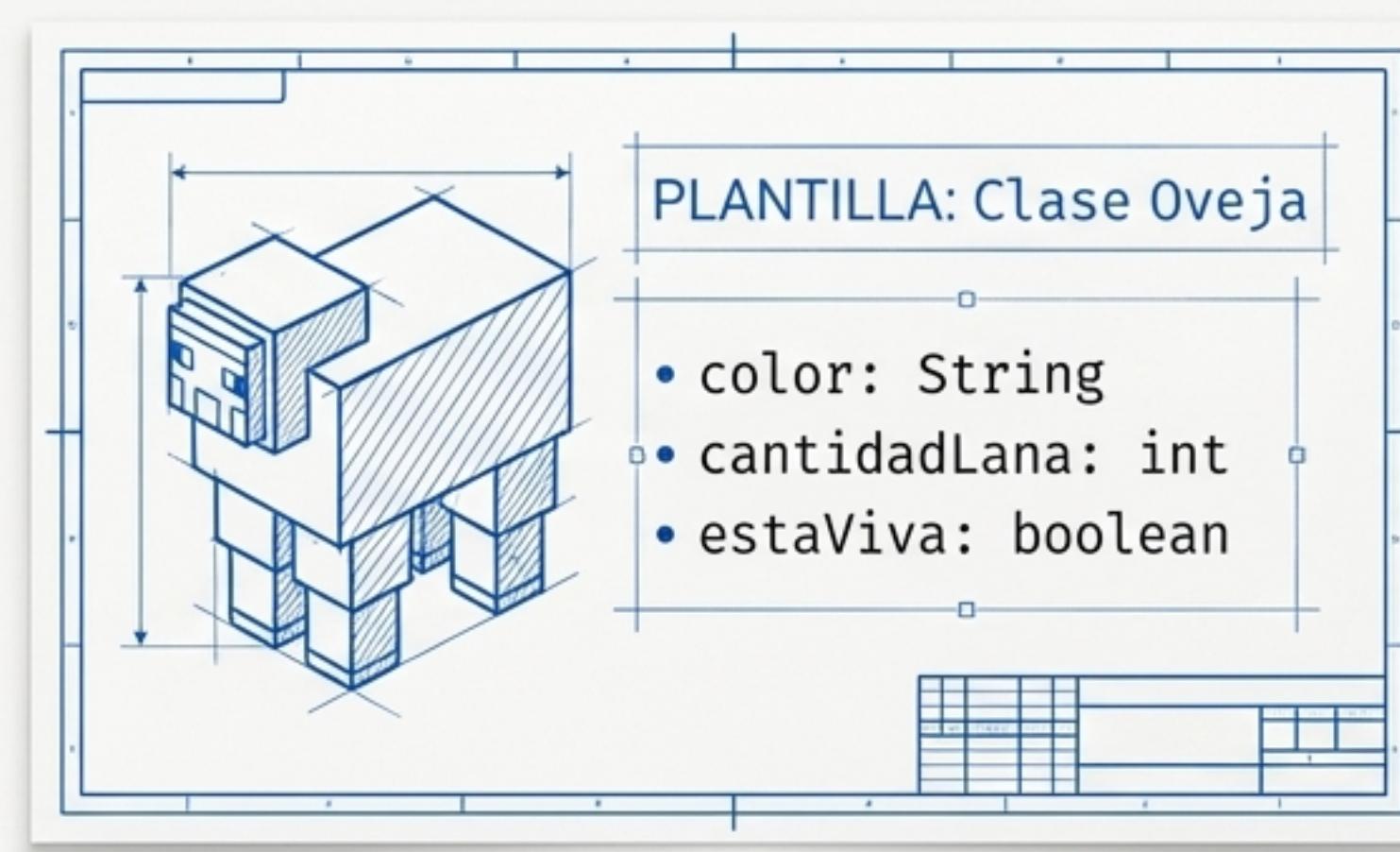
También reserva espacio en memoria, pero es dinámico. Crece si añades datos y se encoge si los quitas.

- Tamaño dinámico.
- Más flexible, ideal cuando no sabes cuántos elementos tendrás.
- Usa el método `.add()` para añadir elementos.

```
// Creas un contenedor que puede crecer según lo necesites.  
ArrayList<String> miListaDinamica = new ArrayList<>();
```

Los Pilares (1/3): La Plantilla y la Creación

Una **Clase** es la plantilla. Un **Objeto** es la creación específica que nace de esa plantilla.



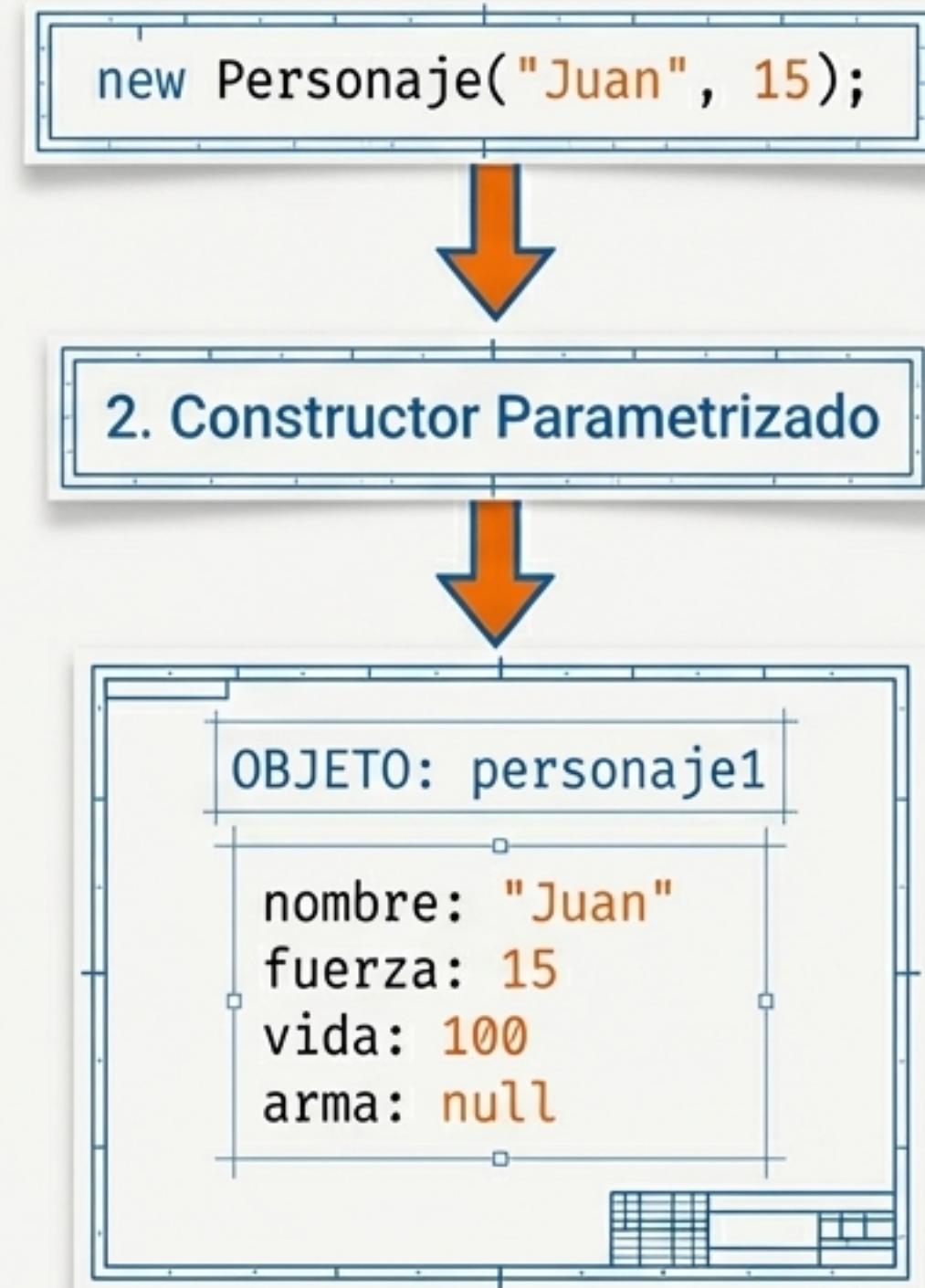
Nuestro Ejemplo

A lo largo de esta guía, usaremos una clase `Personaje` y una clase `Arma` para construir nuestro propio mundo. `Personaje` es la plantilla; "Juan" y "Pedro" serán los objetos.

Los Pilares (2/3): El Constructor, la Chispa de Vida

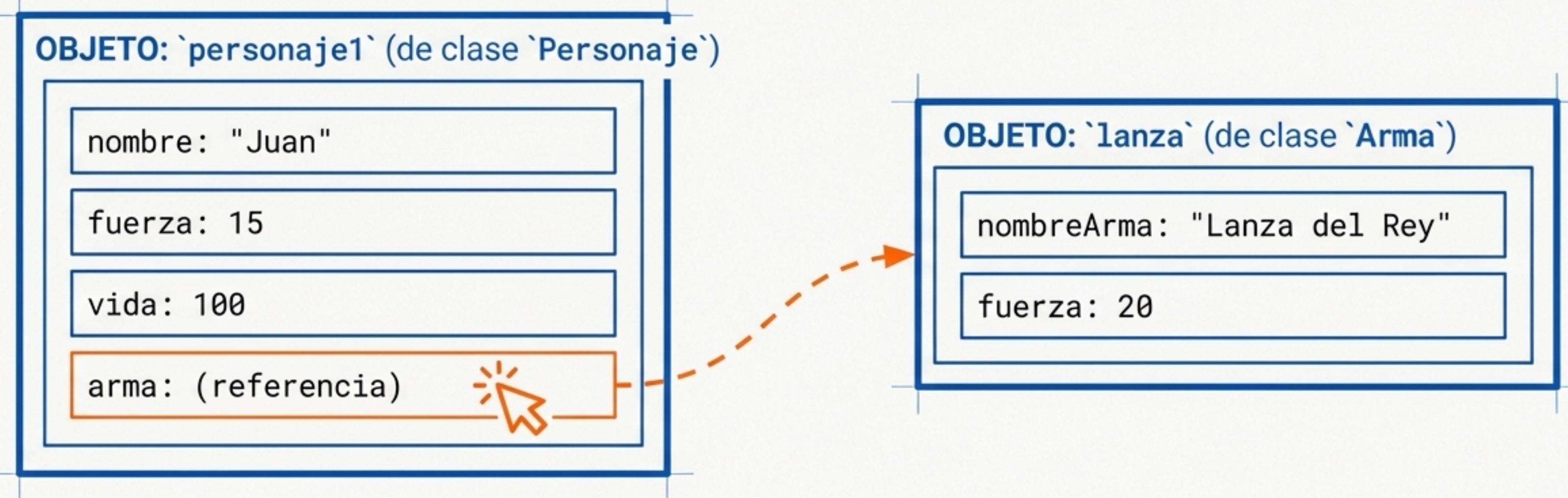
```
public class Personaje {  
    String nombre;  
    int fuerza;  
    int vida;  
    Arma arma; // Un objeto dentro de otro objeto  
  
    // 1. Constructor por Defecto (el "modelo básico")  
1. public Personaje() {  
    this.vida = 100;  
    this.arma = null;  
}  
  
// 2. Constructor Parametrizado (el "modelo personalizado")  
2. public Personaje(String nombre, int fuerza) {  
    this.nombre = nombre;  
    this.fuerza = fuerza;  
    this.vida = 100;  
    this.arma = null;  
}
```

El **constructor** es un método especial que se llama al crear un objeto con `new`. Sirve para inicializar sus atributos y dejarlo listo para usarse.



Los Pilares (3/3): Composición, un Personaje *tiene* un Arma

Un objeto no solo contiene datos primitivos (números, texto). También puede contener otros objetos como atributos. Tu `Personaje` *posee* un objeto de tipo `Arma`.



Visualmente, así es como se organiza en memoria. `personaje1` no contiene directamente los datos del arma, sino una referencia (una 'conexión') al objeto `lanza` que existe de forma independiente.

La Mecánica (1/2): ¿Cuál es Cuál? Resolviendo la Ambigüedad con `this`

El Problema

```
public class Arma {  
    String nombreArma;  
    // ...  
    public Arma(String nombreArma) {  
        nombreArma = nombreArma; // ¿Cuál es cuál?  
    }  
}
```

Fira Code

Aquí, Java no sabe si `nombreArma` se refiere al **atributo de la clase** o al **parámetro del método**. Esto es ambigüedad.

La Solución

```
public class Arma {  
    String nombreArma; // El atributo del objeto  
    // ...  
    public Arma(String nombreArma) {  
        // El parámetro que llega  
        this.nombreArma = nombreArma;  
    }  
}
```

Fira Code

Atributo de la clase

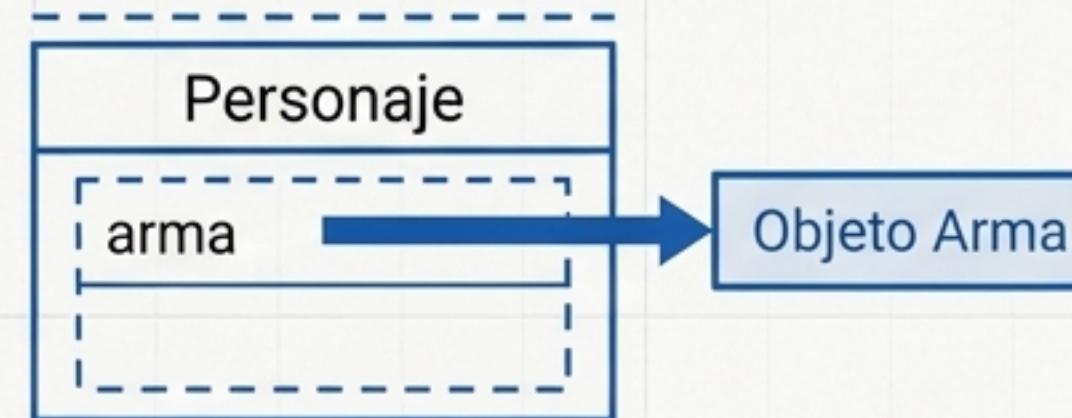
Parámetro del método

`this` se refiere explícitamente al objeto actual. `this.nombreArma` significa "el atributo de ESTE objeto".

La Mecánica (2/2): Getters y Setters, Acceso Controlado

Los atributos de una clase suelen ser `private` para protegerlos. Los métodos `get` (obtener) y `set` (establecer) son la forma pública y segura de interactuar con ellos.

get (Obtener)



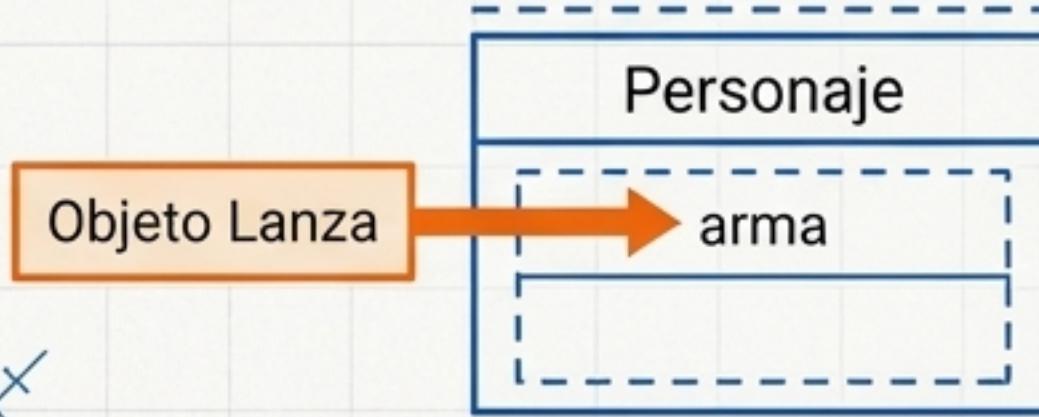
Definición:

```
// En Personaje.java  
public Arma getArma() {  
    return this.arma;  
}
```

Uso:

```
Arma armaDeJuan = personaje1.getArma();
```

set (Establecer)



Definición:

```
// En Personaje.java  
public void setArma(Arma nuevaArma) {  
    this.arma = nuevaArma;  
}
```

Uso:

```
personaje1.setArma(lanza);
```

Navegando Entre Objetos: El Acceso en Cadena

El Reto

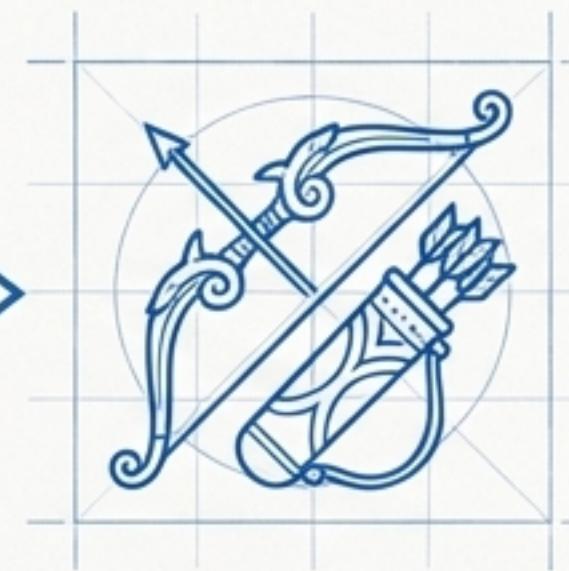
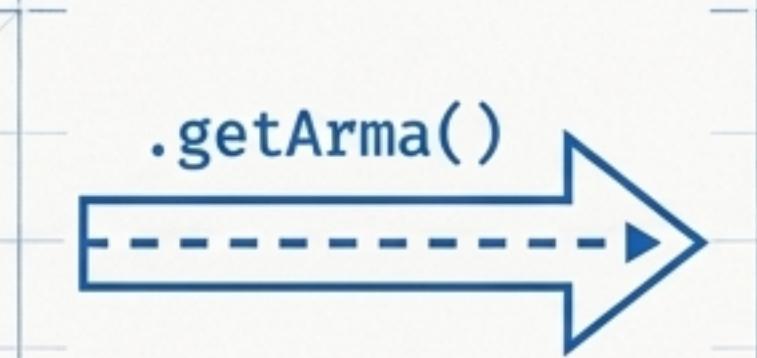
Queremos obtener el **nombre** (un String) del **arma** (un Objeto) que tiene el **personaje2** (otro Objeto).

```
System.out.println( personaje2.getArma().getNombreArma() );
```



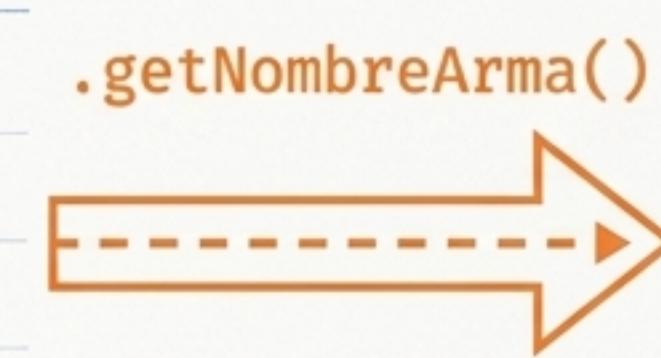
personaje2

Empezamos con nuestro objeto 'Personaje'.



Objeto Arma

Llamamos a su `'getArma()'`. Esto nos devuelve el **OBJETO 'Arma'** completo que tiene dentro.

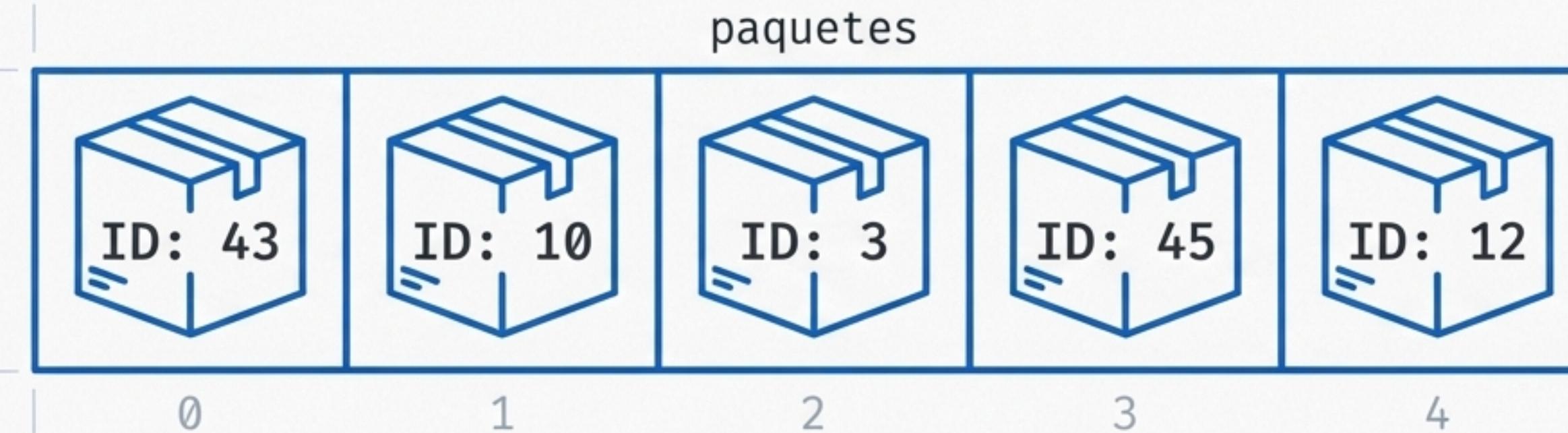


Sobre el objeto 'Arma' que acabamos de obtener, ahora llamamos a su `'getNombreArma()'`. Esto nos devuelve el **STRING** que queremos.

La Estructura (1/2): El Reto de Ordenar Objetos

El Problema

Tenemos un array de objetos de tipo Paquete. Cada Paquete tiene un atributo numérico llamado identificador. El array está desordenado.



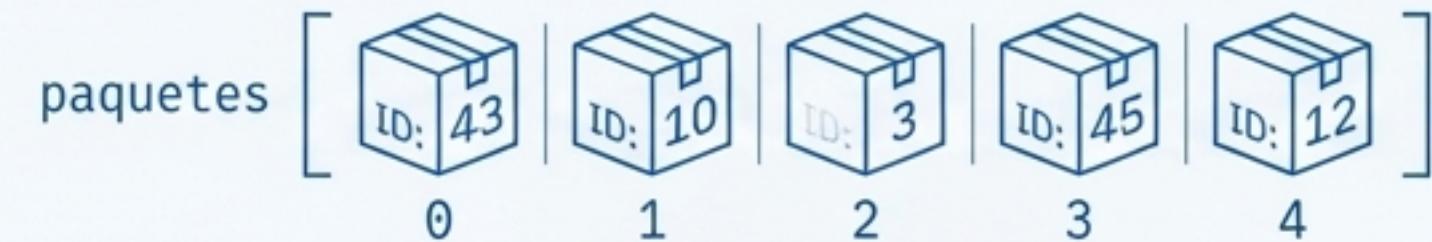
Objetivo: Ordenar este array basándonos en el identificador de cada objeto Paquete, de menor a mayor.



Herramienta: Para lograrlo, analizaremos el **algoritmo de ordenamiento por selección**, paso a paso.

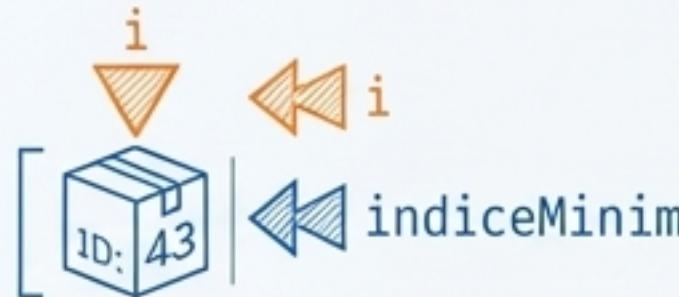
La Estructura (2/2): El Algoritmo de Selección, Paso a Paso

Analicemos la primera iteración completa del bucle exterior ($i = 0$).



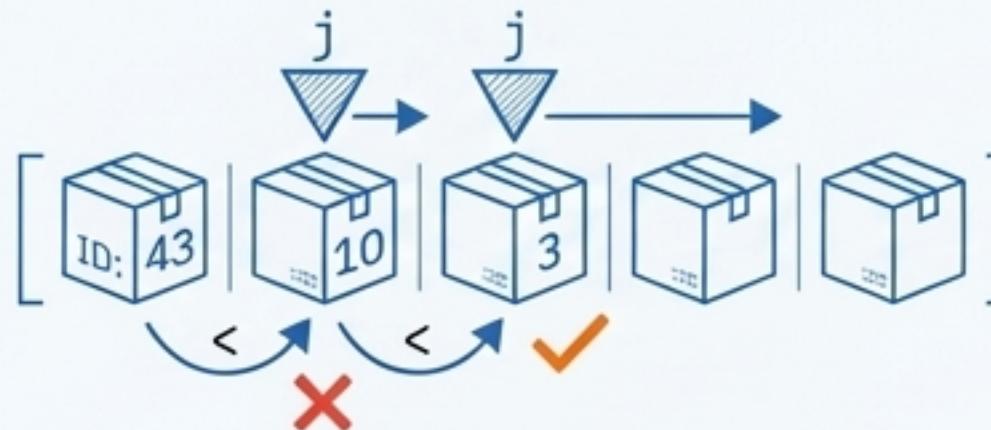
Paso 1: Inicialización.

La i empieza en el índice 0. Asumimos que es el mínimo, así que `indiceMinimo` también es 0.



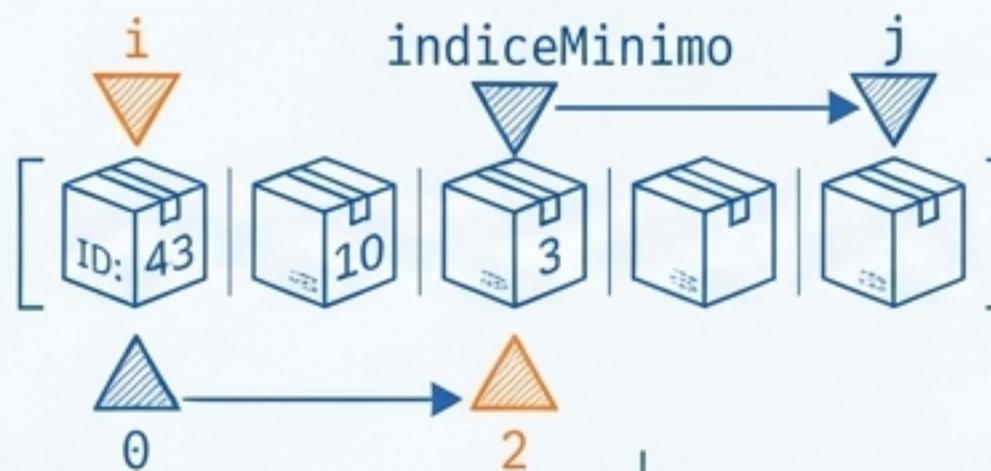
Paso 2: Búsqueda.

La j recorre el resto del array buscando un ID más pequeño que `paquetes[indiceMinimo]`.



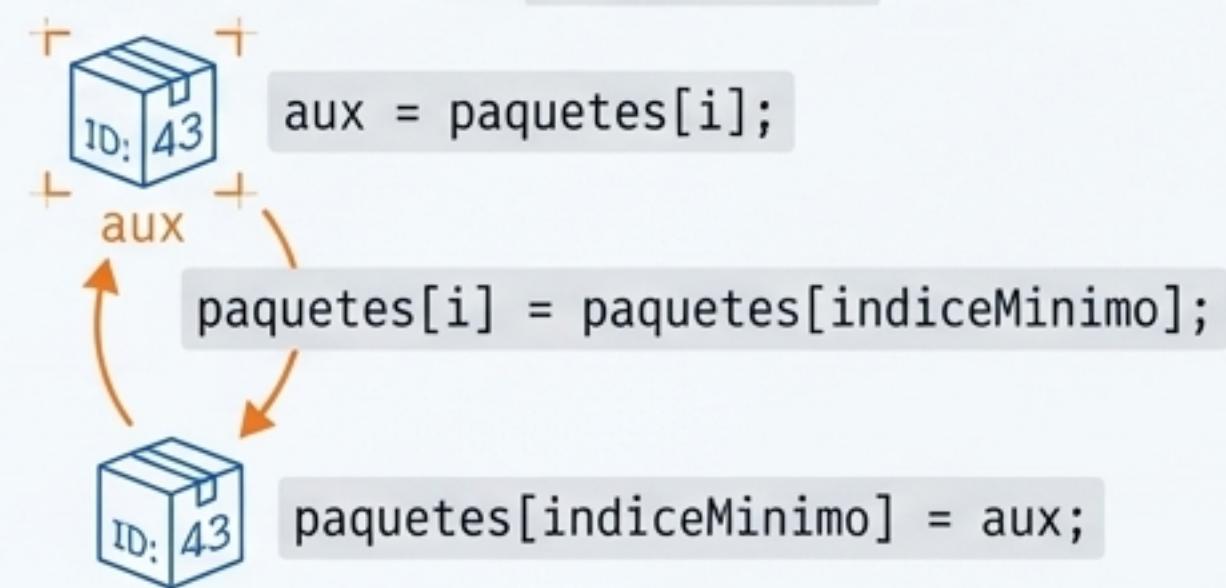
Paso 3: Encontrar un Nuevo Mínimo.

¡Encuentra uno! `paquetes[2]` (ID: 3) es menor. Actualizamos: `indiceMinimo` ahora vale 2.



Paso 4: El Intercambio (Swap).

Al final, intercambiamos el elemento en i (43) con el elemento en `indiceMinimo` (3).



Resultado Final de la Iteración:



Elemento ordenado.

El Algoritmo en Código Java

```
public void ordenarPaquetes(Paquete[] paquetes) {  
    Paquete aux; // Variable temporal para el intercambio  
  
    // Bucle exterior: recorre el array para colocar el mínimo en su lugar  
    for (int i = 0; i < paquetes.length - 1; i++) {  
  
        // Asumimos que el primer elemento de la sección sin ordenar es el mínimo  
        int indiceMinimo = i;  
  
        // Bucle interior: busca el verdadero mínimo en el resto del array  
        for (int j = i + 1; j < paquetes.length; j++) {  
  
            // Comparamos identificadores. Si encontramos uno menor...  
            if (paquetes[j].getIdentificador() < paquetes[indiceMinimo].getIdentificador()) {  
                // ...actualizamos la posición del mínimo.  
                indiceMinimo = j;  
            }  
        }  
  
        // Hacemos el intercambio (swap) del elemento en 'i' con el mínimo encontrado  
        aux = paquetes[i];  
        paquetes[i] = paquetes[indiceMinimo];  
        paquetes[indiceMinimo] = aux;  
    }  
}
```

Los Acabados (1/2): La Primera Vez Siempre Cuenta

La diferencia clave entre `while` y `do-while` es **CUÁNDΟ** se comprueba la condición.

`while`

Comprueba **ANTES** de ejecutar.



Si la condición es falsa desde el principio, el código del bucle no se ejecuta **NUNCA**.

`do-while`

Ejecuta **AL MENOS UNA VEZ**, y luego comprueba.



Garantiza que el bloque de código se ejecute como mínimo una vez, sin importar la condición inicial.

Caso de Uso Práctico: Perfecto para mostrar un menú de opciones. Quieres que el menú aparezca al menos una vez para que el usuario pueda elegir.

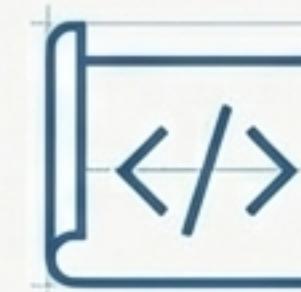
Los Acabados (2/2): Prácticas de un Desarrollador Profesional



1. Usa `try-catch` para Entradas de Usuario

Protege tu programa de datos inesperados. Si esperas un número y el usuario escribe 'hola', un `try-catch` evita que el programa se rompa (crash).

```
try {  
    int edad = scanner.nextInt();  
} catch (InputMismatchException e) {  
    System.out.println("Error: Debes introducir un número.");  
}
```



2. Comenta Tu Código (Explica el 'Porqué')

Tu 'yo' del futuro y tus compañeros te lo agradecerán. No comentes lo obvio (`// Suma 1 a i`), comenta la intención (`// Recorremos para encontrar el mínimo`).

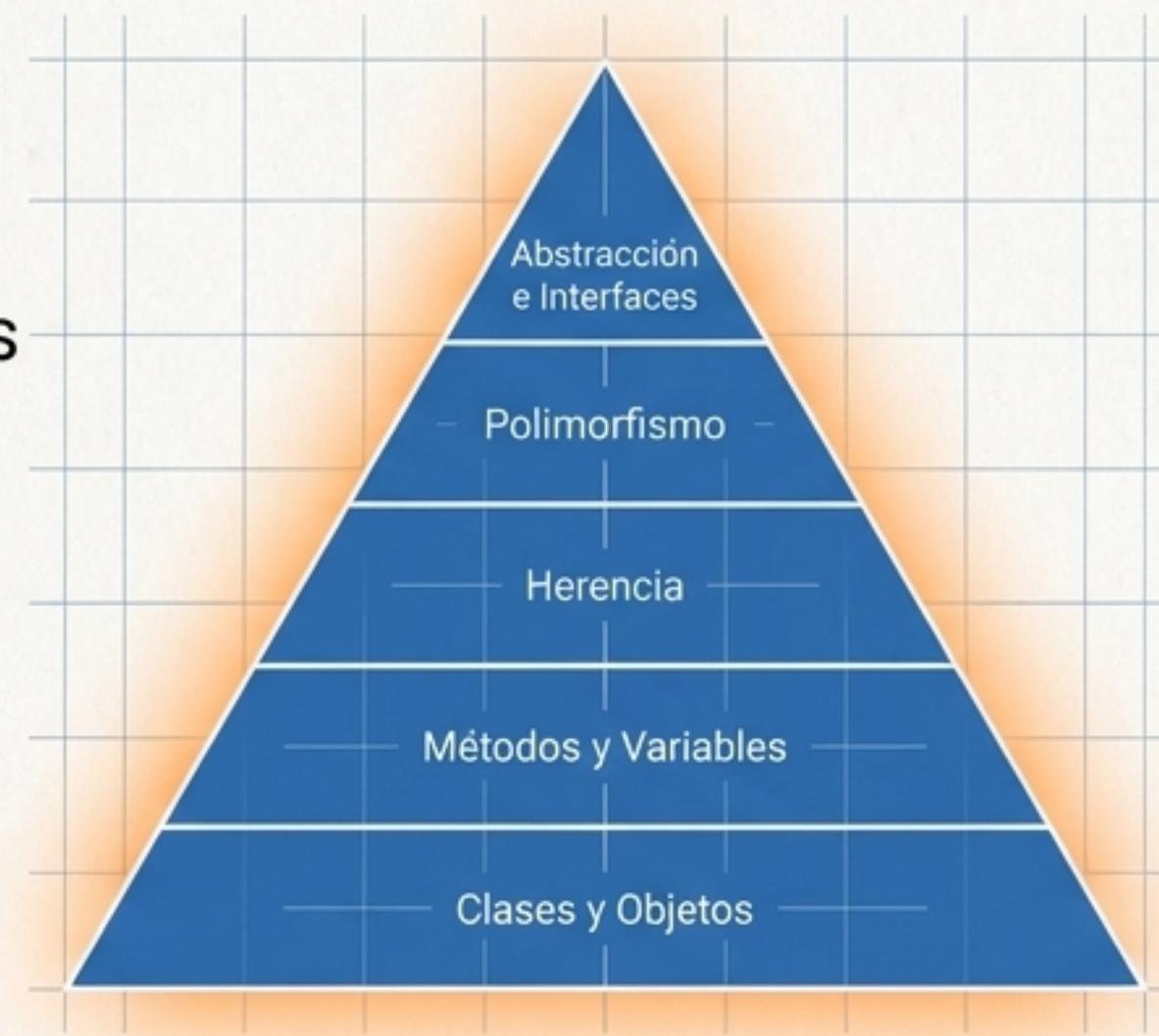
3. Nomenclatura Clara (CamelCase)

La claridad es la clave para un código mantenible.

`nombreDelPersonaje` 
~~`nombredelpersonaje`~~ 

Has Construido una Base Sólida

Has repasado los cimientos de la Programación Orientada a Objetos, desde cómo guardar datos hasta cómo manipularlos con lógica compleja.



La clave ahora es la práctica: revisa estos conceptos, experimenta con el código y no temas cometer errores.

Ahora, a Aplicarlo en el Mundo Real

Todos estos bloques de construcción son esenciales para nuestro siguiente reto: el proyecto 'Anify'.

