

Taller 2 - Programación Orientada a Objetos en Java

Supongamos esta clase:

```
public class Persona {  
    String nombre;  
  
    public Persona(String nombre) {  
        this.nombre = nombre;  
    }  
}
```

En este constructor:

- `nombre` (sin `this`) es el **parámetro** que recibís.
- `this.nombre` es el **atributo** del objeto.

Entonces la línea:

```
this.nombre = nombre;
```

significa:

"Al atributo `nombre` de este objeto, asigne el valor del parámetro `nombre`."

En tu caso

En tu clase `ListaEnlazada`, cuando estás en un método como `copiar()` y escribís:

```
Nodo actual = primero;
```

podrías poner perfectamente:

```
Nodo actual = this.primeros;
```

Y estarías accediendo al atributo `primero` que **está fuera del método**, declarado en la clase:

```
private Nodo primeros;
```



Resumen express

Código

¿Qué significa?

`primero`

accede al atributo si no hay confusión

`this.primer
o`

accede explícitamente al atributo de la instancia actual

`this`

siempre se refiere al objeto actual

¿Querés que te arme un ejercicio guiado para practicar esto de `this`, atributos y constructores?

constructor por copia

¡Claro que sí, Anita! Vamos con una explicación bien clarita y directa:



¿Qué es un constructor por copia?

Un **constructor por copia** es un constructor especial que permite crear un **nuevo objeto** a partir de **otro objeto del mismo tipo**.

👉 En vez de armar el objeto desde cero, copiás lo que ya hay en otro.



¿Para qué sirve?

Para que puedas hacer esto:


```
ListaEnlazada<Integer> listaOriginal = new ListaEnlazada<>();  
listaOriginal.agregarAtras(1);  
listaOriginal.agregarAtras(2);
```

```
ListaEnlazada<Integer> listaCopia = new ListaEnlazada<>(listaOriginal);
```

Y así `listaCopia` tiene los mismos elementos que `listaOriginal`, pero es un objeto nuevo e independiente.

¡Importante!

Hay dos formas de copiar:

- **Copia superficial:** copiás solo las referencias (queda medio “pegada” al original).
-  **Copia profunda:** creás **nuevos objetos internos** con los mismos valores (como hiciste con `copiar()`).

Tu constructor por copia usa `copiar()` para hacer una copia profunda, lo cual está genial porque evita errores de aliasing.

¿Por qué hay *dos* constructores en la misma clase?

¡Gran pregunta! Y sí, puede parecer raro al principio, pero es muy común.

1. Constructor "vacío" (por defecto):

```
java
CopiarEditar
public ListaEnlazada() {
    primero = null;
    size = 0;
}
```

Este sirve para crear **una lista vacía** desde cero.

2. Constructor por copia:

```
java
CopiarEditar
public ListaEnlazada(ListaEnlazada<T> lista) {
    ...
}
```

Este sirve para crear **una lista nueva con el mismo contenido que otra.**