Aliases

Semestre 02, 2025

Referencias y creación de objetos

En Java, cuando creamos un objeto, lo que realmente obtenemos es una **referencia** a un área de memoria donde está almacenado el objeto.

```
Persona p = new Persona();
```

- `Persona` es el tipo (la clase).
- `p` es una referencia al objeto.
- `new Persona()` crea el objeto en memoria.

Ejemplo

```
Persona p1 = new Persona();

p1.nombre = "Ana";
```

En memoria:

```
p1 ---> [ Objeto Persona (nombre: Ana) ]
```

Referencias múltiples (aliases)

Dos variables pueden apuntar al mismo objeto:

```
Persona p2 = p1;

p2.nombre = "Luis";
```

Ahora:

```
p1 ---> [ Objeto Persona (nombre: Luis) ] <--- p2
```

Ambas referencias afectan al **mismo objeto**.

Esto significa que cualquier modificación hecha por medio de una referencia será visible también a través de la otra:

```
System.out.println(p1.nombre); // Luis

System.out.println(p2.nombre); // Luis
```

Precaución:

Este comportamiento puede provocar errores difíciles de detectar si se asume que `p1` y `p2` son objetos distintos.

Para crear una **copia independiente**, se necesita una nueva instancia:

```
Persona p3 = new Persona();


// copia el valor del nombre, no la referencia
```

```
p3.setNombre(p1.getNombre());
```

Este tipo de copia se conoce como **copia superficial** (shallow copy).

En casos más complejos (objetos con atributos que también son objetos), se debe hacer una **copia profunda** (deep copy).

Una copia profunda crea un nuevo objeto junto con nuevas instancias de todos los objetos que contiene internamente.

Esto garantiza que no se comparta memoria entre el objeto original y su copia.

Comparación de referencias

```
if (p1 == p2) {

System.out.println("Misma referencia");

}
```

- `==` compara **referencias**, no contenido.
- Para comparar contenido usamos `.equals()`.

Recolector de basura (Garbage Collector)

Java maneja automáticamente la memoria.

```
p1 = null;
```

Si no hay más referencias a un objeto, el recolector puede liberar su memoria. No tenemos control directo sobre este proceso, pero podemos sugerirlo.

```
System.gc();
```

Clases útiles en Java

Java incluye muchas clases listas para usar:

- `String`: manejo de texto
- `Math`: funciones matemáticas
- `Random`: generación de números aleatorios
- `Scanner`: entrada por teclado

String y Math

```
String nombre = "María";

System.out.println(nombre.toUpperCase());


int maximo = Math.max(10, 20);
```

Random

```
import java.util.Random;
```

```
Random rnd = new Random();

int numero = rnd.nextInt(10); // 0 a 9
```

Scanner

```
import java.util.Scanner;



Scanner sc = new Scanner(System.in);

System.out.print("Edad: ");

int edad = sc.nextInt();
```

Tipos primitivos vs objetos

Java distingue entre:

- **Primitivos**: `int`, `double`, `char`, `boolean` ...
- **Objetos**: `Integer`, `Double`, `Character`, `Boolean` ...

Clases Wrapper

Cada tipo primitivo tiene una clase equivalente:

| Primitivo | Clase Wrapper |

| ————— | ————————— |

| int | Integer |

| double | Double |

| char | Character |

| boolean | Boolean |

Permiten usar métodos útiles y trabajar con colecciones:

```
Integer edad = 25;

System.out.println(edad.toString());
```

Autoboxing y unboxing

Java permite convertir automáticamente entre tipos primitivos y sus equivalentes en clases wrapper.

Autoboxing

Es el proceso por el cual Java convierte automáticamente un tipo primitivo a su clase wrapper cuando se requiere un objeto.

```
int numero = 20;

Integer edad = numero; // autoboxing: int → Integer
```

Unboxing

Es el proceso inverso: Java convierte automáticamente un objeto wrapper a su tipo primitivo.

```
Integer altura = 180;

int h = altura; // unboxing: Integer → int
```

Enumeraciones (enum)

Tipo especial que permite definir un conjunto limitado de valores:

```
public enum Dia {

LUNES, MARTES, MIERCOLES, JUEVES, VIERNES

}


Dia hoy = Dia.LUNES;
```

Se comportan como clases y se pueden comparar:

```
if (hoy == Dia.LUNES) {

System.out.println("Inicio de semana");

}
```

Beneficios

- Código más legible.
- Funciones ya implementadas.

- Menos errores.
- Mejores prácticas de programación orientada a objetos.

```
Dataview (inline field '='): Error:
-- PARSING FAILED ------------------------------------------------
-

> 1 | =
    | ^

Expected one of the following:

'(', 'null', boolean, date, duration, file link, list ('[1, 2,
3]'), negated field, number, object ('{ a: 1, b: 2 }'), string,
variable
```