Aliases

Programación Orientada a Objetos (CC2008) - 2026

Aliases

Semestre 02, 2026

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 clases listas para usar. Estudiaremos cada una de forma individual junto con su paquete:

- `String` : manejo de texto — `java.lang`
- `Math` : funciones matemáticas — `java.lang`
- `Random` : números aleatorios — `java.util`
- `NumberFormat` : formato de números — `java.text`
- `DecimalFormat` : formato personalizado — `java.text`
- `Scanner` : entrada por teclado — `java.util`

`String` y `Math` pertenecen a `java.lang` y no necesitan `import`. Las demás sí requieren importarse.

String

Representa una cadena de texto. Es inmutable: cada operación devuelve un nuevo `String`. Métodos más usados:

- `length()` : cantidad de caracteres.
- `charAt(i)` : carácter en la posición `i`.
- `substring(a, b)` : subcadena entre índices.
- `indexOf(txt)` : posición de un texto.
- `replace(a, b)` : reemplaza texto.
- `equals(txt)` : compara contenido.

```
String nombre = "María López";

System.out.println(nombre.length());        // 11
System.out.println(nombre.toUpperCase());   // MARÍA LÓPEZ
System.out.println(nombre.charAt(0));       // M
System.out.println(nombre.substring(0, 5)); // María
System.out.println(nombre.indexOf("López")); // 6
System.out.println(nombre.replace("María", "Ana")); // Ana López
```

Para comparar texto usa `.equals()`, no `==` (que compara referencias).

Math

Ofrece funciones matemáticas mediante métodos estáticos: se usan con `Math.` sin crear objetos.

- `abs(x)`: valor absoluto.
- `max(a,b)` / `min(a,b)`: mayor y menor.
- `pow(a,b)`: potencia. `sqrt(x)`: raíz.
- `round`, `ceil`, `floor`: redondeo.
- `random()`: número entre 0.0 y 1.0.
- `PI`, `E`: constantes.

```
int maximo = Math.max(10, 20);      // 20
double raiz = Math.sqrt(144);       // 12.0
double potencia = Math.pow(2, 10);  // 1024.0
long redondeo = Math.round(3.7);    // 4
double area = Math.PI * Math.pow(5, 2); // círculo r=5
```

Random

Genera valores aleatorios. A diferencia de `Math`, sí requiere crear un objeto e importarse desde `java.util`.

```
import java.util.Random;

Random rnd = new Random();
int numero  = rnd.nextInt(10);   // entero de 0 a 9
double dec  = rnd.nextDouble();  // decimal de 0.0 a 1.0
boolean bit = rnd.nextBoolean(); // true o false
```

NumberFormat

Da formato a números según una configuración regional (locale): moneda, porcentaje o número general, con separadores adecuados.

- `getCurrencyInstance()` : formato de moneda.
- `getPercentInstance()` : formato de porcentaje.
- `getNumberInstance()` : número con separadores.

```
import java.text.NumberFormat;

NumberFormat moneda = NumberFormat.getCurrencyInstance();
System.out.println(moneda.format(1234.5)); // $1,234.50

NumberFormat pct = NumberFormat.getPercentInstance();
System.out.println(pct.format(0.75));      // 75%

NumberFormat num = NumberFormat.getNumberInstance();
System.out.println(num.format(1000000));   // 1,000,000
```

El símbolo de moneda y los separadores dependen del `Locale` del sistema.

DecimalFormat

Permite definir un patrón personalizado para dar formato a un número: cuántos decimales mostrar, separadores de miles, porcentajes, etc.

- `0` : dígito obligatorio (rellena con cero).
- `#` : dígito opcional (se omite si no hay).
- `.` separador decimal · `,` separador de miles.
- `%` : multiplica por 100 y añade el símbolo.

```
import java.text.DecimalFormat;

DecimalFormat df = new DecimalFormat("#,##0.00");
```

```
System.out.println(df.format(1234.5));   // 1,234.50

DecimalFormat df2 = new DecimalFormat("0.###");
System.out.println(df2.format(3.14159)); // 3.142

DecimalFormat df3 = new DecimalFormat("0.0%");
System.out.println(df3.format(0.85));    // 85.0%
```

`NumberFormat` usa formatos predefinidos por región; `DecimalFormat` te da control total mediante un patrón propio.

Scanner

Lee datos de entrada, típicamente desde el teclado (`System.in`).

```
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.