Arrays Dinámicos

Programación Orientada a Objetos (CC2008) - 2026

Arreglos Dinámicos

Semestre 02, 2025

Arrays en Java

Tipo fijo: todos los elementos son del mismo tipo.

Tamaño fijo: se define al crear y no cambia.

Acceso por índice.

```
int[] numeros = new int[3];
numeros[0] = 10;
```

Limitación: redimensionar implica crear un nuevo arreglo y copiar.

¿Por qué un arreglo dinámico?

- Escenarios con cantidad **variable** de elementos.
- Necesidad de **insertar/eliminar** con frecuencia.
- Evitar manejar manualmente copias y tamaños.

Solución: `ArrayList<E>` (en `java.util`).

ArrayList

Implementa la interfaz `List<E>`.

Arreglo dinámico internamente (crece al necesitar más espacio).

Permite `null` como valor.

```
import java.util.ArrayList;

// <> usa el tipo del lado izquierdo
ArrayList<String> frutas = new ArrayList<>();

frutas.add("Manzana");
frutas.add("Pera");

// Manzana
System.out.println(frutas.get(0));
```

Operaciones comunes de ArrayList

Insertar

```
ArrayList<String> items = new ArrayList<>();

items.add("A");
items.add(0, "Inicio");
```

Leer

```
ArrayList<String> items = new ArrayList<>();

String x = items.get(1);
```

Reemplazar

```
ArrayList<String> items = new ArrayList<>();

items.set(1, "Nuevo");
```

Borrar

```
ArrayList<String> items = new ArrayList<>();

// eliminar por índice
items.remove(0);

// eliminar por objeto (usa equals)
items.remove("A");
```

Varios

```
ArrayList<String> items = new ArrayList<>();

// tamaño actual
int n = items.size();

boolean tiene = items.contains("A");

// vaciar
items.clear();
```

Iteración sobre ArrayList

```
for (int i = 0; i < items.size(); i++) {
    System.out.println(items.get(i));
}
```

```
for (String it : items) {
    System.out.println(it);
}

items.forEach(System.out::println); // expresión funcional
```

Cuidado: Modificar la lista mientras iteras con `for-each` puede lanzar `ConcurrentModificationException`.

Conversiones

```
// Array -> List (vista no modificable en tamaño)
String[] arr = {"A", "B", "C"};
// tamaño fijo, respalda el array
List<String> view = Arrays.asList(arr);

// Array -> ArrayList independiente
ArrayList<String> l = new ArrayList<>(Arrays.asList(arr));

// List -> Array
String[] copia = l.toArray(new String[0]);
```

Diferencia clave: `Arrays.asList` devuelve una **vista** con tamaño fijo (no `add/remove`).

ArrayList con objetos propios

```
class Persona {
    private String nombre;

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

```
    }

    public String getNombre() {
        return nombre;
    }

    @Override public String toString() {
        return nombre;
    }
}
```

```
ArrayList<Persona> personas = new ArrayList<>();

personas.add(new Persona("Ana"));
personas.add(new Persona("Luis"));

System.out.println(personas);
```

Arreglos dinámicos propios

Para comprender `ArrayList`, puedes implementar un **DynamicArray** simple:

```
class DynamicIntArray {
    private int[] data = new int[4];
    private int size = 0;

    public void add(int v) {
        if (size == data.length) {
            data = Arrays.copyOf(data, data.length * 2);
        }

        data[size++] = v;
    }

    public int get(int i) { return data[i]; }
```

```
    public int size() { return size; }
}
```

Excepciones en Java

Una **excepción** es un evento que interrumpe el flujo normal del programa.

Se **lanza (throw)** y puede **capturarse (catch)**.

Jerarquía base: `Throwable` → `Exception` / `Error`.

Tipos

Chequeadas:

Son verificadas por el compilador.

Representan condiciones que un programa bien escrito debería anticipar y manejar (ej. errores de entrada/salida, acceso a base de datos).

Ejemplos: IOException, SQLException.

El compilador obliga a rodearlas con try/catch o declararlas en la firma del método con throws.

Se usan para situaciones recuperables.

No chequeadas:

Heredan de RuntimeException.

Representan errores de programación o mal uso de una API.

Ejemplos: NullPointerException, IllegalArgumentException, ArithmeticException.

El compilador no obliga a declararlas ni capturarlas.

Normalmente indican bugs o condiciones que no deberían ocurrir si el código está correcto.

try/catch

```
try {
    int r = 10 / 0; // ArithmeticException
    System.out.println(r);
} catch (ArithmeticException e) {
    System.out.println("Error aritmético: " +
        e.getMessage());
}
```

- Del **más específico** al **más general** si hay múltiples `catch`.

Multi-catch y orden

```
try {
    Files.readString(Path.of("data.txt"));

// multi-catch
} catch (NoSuchFileException | AccessDeniedException e) {
    System.out.println("Archivo inaccesible: " +
        e.getMessage());

// más general al final
} catch (IOException e) {
    e.printStackTrace();
}
```

finally

El bloque `finally` **siempre** se ejecuta (haya o no excepción).

Útil para **liberar recursos**: cerrar archivos, conexiones, etc.

```
Reader r = null;

try {
    r = Files.newBufferedReader(Path.of("data.txt"));
    System.out.println(r.readLine());
} catch (IOException e) {
    System.out.println("I/O: " + e.getMessage());
} finally {
    if (r != null) {
        r.close();
    }
}
```

Lanzar y propagar excepciones

En Java, las excepciones pueden lanzarse (throw) dentro de un método y propagarse (throws) hacia el que lo invoque.

Lanzar (throw)

Se utiliza la palabra clave throw para generar una excepción en un punto específico del código.

Se usa cuando se detecta una condición inválida o inesperada.

```
public int dividir(int a, int b) {
    if (b == 0) {
        throw new IllegalArgumentException("b != 0");
    }

    return a / b;
```

```
}
```

En este ejemplo, si b es 0, se lanza una IllegalArgumentException y el flujo normal del programa se interrumpe.

Propagación (throws)

Se utiliza en la firma del método para indicar que ese método puede lanzar una excepción hacia quien lo invoque.

Es obligatorio para las excepciones chequeadas (checked).

Permite delegar el manejo del error a niveles superiores.

```
public void cargar() throws IOException {
    Files.readString(Path.of("config.json"));
}
```

Aquí, el método cargar() no captura la excepción, sino que declara que puede lanzarla. El código que llame a cargar() debe rodearlo con try/catch o volver a declarar throws.

Buenas prácticas

- Captura solo lo que **puedes manejar**.
- No **silencies** excepciones (no dejes `catch` vacío).
- Lanza `IllegalArgumentException` / `IllegalStateException` para validar precondiciones.