Arrays

Semestre 02, 2025

Definición

Lista ordenada de elementos del mismo tipo (homogeneas).

Tamaño fijo: los arreglos no pueden crecer o reducirse.

Se accede a cada elemento mediante un índice.

Declaración y creación

```

  int[] scores = new int[10];


  float[] prices = new float[500];


  boolean[] flags = new boolean[20];


  char[] codes = new char[1750];

```

Indexación

```

  scores[0] = 85;

  scores[1] = 90;
```

```
5
6
7
8   System.out.println(scores[0]); // 85
9
```

Cada elemento de un arreglo se accede mediante su **índice**, comenzando en `0` hasta `length - 1`.

- `scores[0]` = **primer elemento**.
- `scores[scores.length - 1]` = **último elemento**.

La indexación permite **leer y modificar** los valores directamente.

Internamente, Java no verifica si el índice es válido antes de acceder al elemento. Si el índice está fuera del rango permitido, se lanza una excepción.

El acceso fuera de límites lanza `ArrayIndexOutOfBoundsException`:

```
1
2   System.out.println(scores[10]); // Error!
3
```

Este error ocurre cuando se intenta acceder a una posición que no existe en el arreglo. Es importante usar `scores.length` para evitar este tipo de errores.

Iteración

Para recorrer un arreglo y procesar todos sus elementos, se utilizan ciclos como `for` o `for-each`.

Ciclo for tradicional:

- Se controla manualmente el índice.
- Permite acceso total al arreglo (lectura y escritura).

```
1
2   for (int i = 0; i < scores.length; i++) {
3
```

```
4   System.out.println(scores[i]);
5
6   }
7
```

Ciclo for-each (enhanced for):

- Más legible y simple para recorrer todos los elementos.
- Solo lectura (no se puede modificar directamente el arreglo).

```
1
2   for (int score : scores) {
3
4   System.out.println(score);
5
6   }
7
```

Ambos ciclos son válidos. El `for` tradicional da más control, mientras que el `for-each` es ideal para recorrer de forma sencilla los valores.

Inicialización directa

Java permite declarar e inicializar un arreglo en una sola línea usando **llaves** `{}` y una lista de valores.

```
1
2   int[] unidades = {147, 323, 89, 933};
3
4
5
6   char[] calificaciones = {'A', 'B', 'C', 'D', 'F'};
7
8
9
10  System.out.println(unidades.length); // 4
11
```

Esta forma es conocida como **inicialización directa** y es muy útil para arreglos cuyos valores son conocidos desde el principio.

Esta sintaxis solo se puede utilizar al momento de declarar el arreglo. No se puede usar en una asignación posterior:

```
int[] numeros;

numeros = {1, 2, 3}; // Esto causa error

```

En ese caso, debes usar:

```
numeros = new int[] {1, 2, 3};

```

Arreglos como parámetros

Los arreglos pueden pasarse como parámetros a métodos. Dado que son objetos en Java, al pasarlos se transfiere su referencia y no una copia.

Lo que se manda es un Alias.

```
public void imprimir(int[] arreglo) {

for (int valor : arreglo) {

System.out.println(valor);

}

}

```

Esto significa que cualquier modificación hecha dentro del método afecta directamente al arreglo original.

Ejemplo

```
public void modificar(int[] datos) {
datos[0] = 999;
}

int[] valores = {1, 2, 3};

modificar(valores);

System.out.println(valores[0]); // 999

```

Para copiar:

Para evitar efectos colaterales, se puede usar `Arrays.copyOf()` si necesitas una copia.

```
import java.util.Arrays;

int[] original = {1, 2, 3, 4};
int[] copia = Arrays.copyOf(original, original.length);

```

Arreglos de objetos

Cuando se declara un arreglo de objetos, Java **reserva memoria para las referencias**, pero **no crea los objetos en sí**.

```
String[] palabras = new String[5];
```

Esto crea un arreglo con 5 espacios que inicialmente contienen `null`.

Acceder a un elemento sin haberlo inicializado lanza una `:`

```
palabras[0] = "hola";


System.out.println(palabras[0]);
```

Es importante asegurarse de que cada posición contenga un objeto antes de usarlo.

Inicialización de arreglos de objetos

Para llenar un arreglo de objetos personalizados, debes crear cada instancia de forma individual:

```
Persona[] personas = new Persona[3];


for (int i = 0; i < personas.length; i++) {
personas[i] = new Persona();
}
```

Esto garantiza que cada elemento apunte a un objeto `Persona` válido. Sin esta inicialización, cualquier intento de acceder a sus métodos o atributos lanzará una excepción.

Parámetros variables

Java permite definir un método que acepte un número variable de argumentos del mismo tipo, usando la sintaxis `...` (varargs):

```
1
2   public double promedio(int... numeros) {
3
4   int suma = 0;
5
6
7
8   for (int n : numeros) {
9
10  suma += n;
11
12  }
13
14
15
16  return (double) suma / numeros.length;
17
18  }
19
20
21
22  promedio(10, 20, 30);
23
24  promedio(1, 2, 3, 4, 5, 6);
25
```

Internamente, los argumentos se tratan como un **arreglo**.

Solo puede haber **un parámetro varargs por método**.

Debe ser el **último parámetro** de la lista.

Ideal cuando no sabes cuántos argumentos se pasarán al método.

Arreglos multidimensionales

Un arreglo multidimensional en Java es en realidad un **arreglo de arreglos**.

En álgebra lineal le llamamos matriz.

```

  int[][] matriz = new int[3][4];


  matriz[0][0] = 10;


  System.out.println(matriz[0][0]);

```

Esto crea una matriz de 3 filas y 4 columnas.

Buenas prácticas

Siempre verificar el tamaño (`.length`) antes de acceder.

Inicializar arreglos de objetos antes de usarlos.

Evitar modificar directamente los arreglos en métodos a menos que sea intencional.

Conclusiones

Los arreglos permiten agrupar datos del mismo tipo de manera ordenada.

Son objetos y se comportan como tales.

Permiten trabajar con datos primitivos y objetos.

Fundamental para el manejo de estructuras más complejas más adelante.