códigofacilito

Java Profesional

Bloque 6 : Colecciones y arreglos

Tatiana Borda
https://tatianaborda.com/
https://www.linkedin.com/in/tatiana-borda/
https://www.youtube.com/@AlienExplorer

códigofacilito

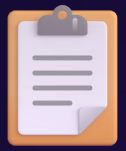


Tabla de contenido

códigofacilito

/01

Arreglos (arrays) unidimensionales

Ya hablamos de ellos en bloques anteriores.
En Java, un array es una estructura que almacena elementos del mismo tipo en una secuencia contigua de memoria.
Los arrays son una herramienta fundamental en Java para organizar y manipular datos de manera eficiente. Ya que permiten: Almacenar y organizar múltiples valores del mismo tipo en una sola variable.Acceder a elementos de forma rápida y eficiente utilizando índices. Y realizar operaciones con grandes volúmenes de datos de forma más sencilla.

códigofacilito

Declaración e Inicialización:

Se declara el tipo de dato de los elementos del array seguido de [] y el nombre del array:
Ejemplo: `int[] numeros;`

Para inicializar podemos hacerlo:

*Con tamaño: Se puede crear un array de un tamaño específico.
Ejemplo: `int[] numeros = new int[5];`
Crea un array de 5 enteros

*Con valores: Se puede inicializar el array directamente con los valores.
Ejemplo: `int[] numeros = {1, 2, 3, 4, 5};`

códigofacilito

Acceso a los elementos:

Cada elemento del array se accede mediante un índice numérico, comenzando en 0. El primer elemento tiene el índice 0, el segundo el índice 1, y así sucesivamente.
Ejemplo: `numeros[0] = 10;`

Asigna el valor 10 al primer elemento del array numeros.
Lo que facilita la organización y manipulación eficiente de datos.

códigofacilito

Características:

*Tamaño fijo (no se puede cambiar después de la creación)

*Acceso por índice (comienza en 0)

*Propiedad length para obtener el tamaño

*Tipo de dato NO primitivo

```
int[] numeros = {10, 20, 30};  // Forma simplificada
String[] palabras = new String[3];  // Forma con tamaño definido
palabras[0] = "Hola";
System.out.println(numeros[1]);  // 20
```

códigofacilito

Ejemplo:

Creamos un array de int con 3 elementos, asignamos valores a cada posición y luego lo recorremos con un for:

```java
int[] edades = new int[3];
edades[0] = 25;
edades[1] = 30;
edades[2] = 22;

// Recorrido tradicional
for(int i = 0; i < edades.length; i++) {
    System.out.println("Edad " + i + ": " + edades[i]);
}
```

Salida:

```
Edad 0: 25
Edad 1: 30
Edad 2: 22
```

códigofacilito

/02 Arreglos multidimensionales

Son aquellos que tienen más de una dimensión y, en consecuencia, más de un índice. Los arreglos más usados son los de dos dimensiones, conocidos también por el nombre de tablas o matrices.
Se les llama Arrays de arrays, heredan las características propias de los arrays, hay que declarar su tipo de dato, ambos almacenan sus elementos en posiciones contiguas de memoria y en ambos objetos accedemos a su valor a través del índice que empieza en 0

Declaración y acceso a valores:

```
int[][] matriz = new int[3][3];  // Matriz 3x3
String[][] nombresApellidos = {{"Ana", "Pérez"}, {"Juan", "Gómez"}};
```

```
matriz[0][0] = 1;  // Primera fila, primera columna
matriz[1][2] = 5;  // Segunda fila, tercera columna
```

códigofacilito

Ejemplo:

Creamos un array multidimensional con 2 filas y 3 columnas

```
int[][] calificaciones = {
    {8, 9, 7},
    {6, 10, 9}
};
```

Representando una tabla de notas.

```
Fila 0: [8, 9, 7]
Fila 1: [6, 10, 9]
```

códigofacilito

Ejemplo:

Para recorrerlo necesitaremos for anidados

```
for (int i = 0; i < calificaciones.length; i++) {
    for (int j = 0; j < calificaciones[i].length; j++) {
        System.out.println("calificaciones[" + i + "][" + j + "] = " + calificaciones[i][j]);
    }
}
```

La salida:

```
calificaciones[0][0] = 8
calificaciones[0][1] = 9
calificaciones[0][2] = 7
calificaciones[1][0] = 6
calificaciones[1][1] = 10
calificaciones[1][2] = 9
```

códigofacilito

/03

ArrayList y List

Es una de las estructuras de datos más utilizadas en Java. Permite modificar y almacenar dinámicamente una colección de objetos.
En Java, `List` es una interfaz del Java Collections Framework que representa una colección ordenada de elementos (también llamada secuencia). `ArrayList` es una de las implementaciones más comunes de `List` y está basada en un arreglo dinámico que puede crecer o reducirse según sea necesario.

códigofacilito

Cuándo usar cada una?

Dijimos que List es el contrato (qué puede hacer una lista). ArrayList es la herramienta concreta.
Declarar con List y crear con ArrayList es la mejor práctica.
Ejemplo: List (interfaz):Se usa en la declaración para escribir código más flexible y desacoplado.

```
List<String> nombres = new ArrayList<>();
```

Esto te permite más adelante cambiar a una LinkedList o a un Vector o a un Stack:

```
List<String> nombres = new LinkedList<>();
```

El código que lo usa no necesita saber qué tipo específico es, solo que se comporta como una lista, recuerda el concepto de Interfaz!

códigofacilito

ArrayList (implementación)

*Tiene tamaño Dinámico (crece o se reduce)
*Admite Solo objetos (usa wrappers como Integer)
*Iteración rápida
*Posee métodos útiles como add(), remove(), contains(), get(index), size() etc.
*Es más lento en cuanto a rendimiento (pero más flexible)
*Declaración e inicialización:

```java
import java.util.ArrayList;
import java.util.List;

// Formas de declarar un ArrayList
List<String> nombres = new ArrayList<>(); // Recomendado (uso de interfaz)
ArrayList<Integer> numeros = new ArrayList<>(); // También válido
```

```java
import java.util.ArrayList;
import java.util.List;

public class ejemploArrayList  {
    public static void main(String[] args) {
        // 1. Creacion de la lista y agregar elementos
        List<String> tareas = new ArrayList<>();
        tareas.add("Estudiar");
        tareas.add("Practicar");
        tareas.add("Descansar");
        System.out.println(tareas);

        // 2. Acceder a elementos y verificar existencia
        System.out.println("--------------");
        System.out.println("Primera tarea: " + tareas.get(0));
        System.out.println("Está 'Practicar'? " + tareas.contains("Practicar"));

        // 3. Insertar en posicion especifica
        System.out.println("--------------");
        tareas.add(1, "Repasar");
        System.out.println(tareas);

        // 4. Eliminar elemento y mostrar cambios
        System.out.println("--------------");
        tareas.remove("Practicar");
        System.out.println(tareas);
        System.out.println("Cantidad de tareas: " + tareas.size());
    }
}
```

códigofacilito

SALIDA:

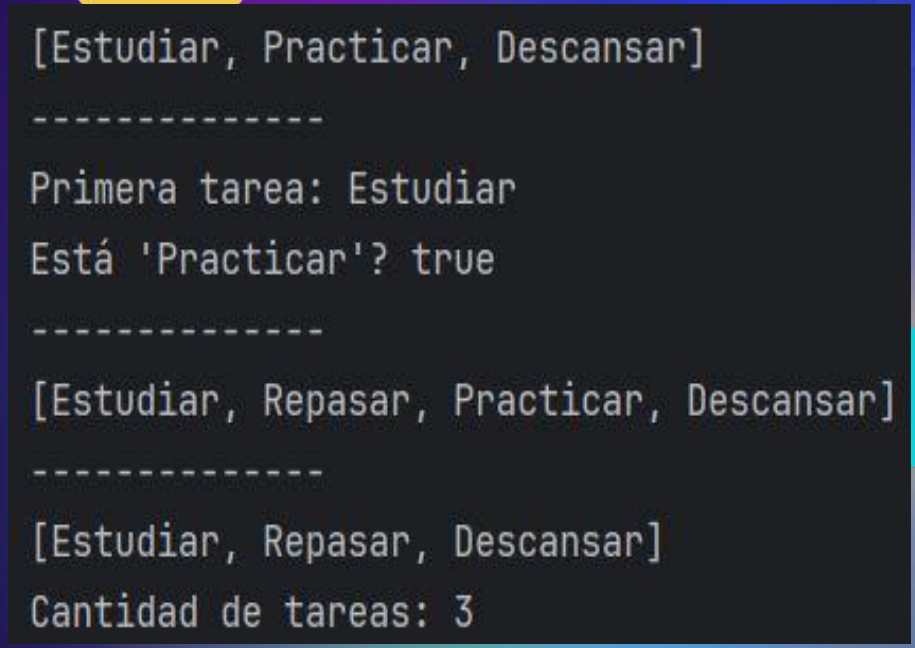
*Inicialización y agregado: Creamos el ArrayList y añadimos 3 elementos, los mostramos.

*Accediendo al primer elemento (get(0)) vemos la primera tarea y evaluando si existe un elemento específico (con contains) verificmos su existencia

*Insertamos "Repasar" en la posición 1 (entre "Estudiar" y "Practicar") y mostramos cómo cambia el orden.

*Eliminamos "Practicar" y mostramos la lista resultante

*Verificamos el tamaño final con size()



códigofacilito

/04 Mapas y diccionarios básicos (HashMap)

Los mapas (o diccionarios) son estructuras de datos que almacenan pares clave-valor, donde cada clave es única.En Java, `HashMap` es una estructura de datos que implementa la interfaz `Map`. Permite almacenar pares de clave-valor, donde cada clave es única y se mapea a un valor específico.

*Al igual que List, Map tiene implementaciones alternativas como TreeMap, LinkedHashMap y ConcurrentHashMap

códigofacilito

Características Clave

1. Pares clave-valor: Almacena datos como pares donde cada clave única está asociada a un valor específico

2.Rendimiento eficiente: Proporciona un rendimiento en tiempo constante para las operaciones de búsqueda, inserción y eliminación de elementos

3.No garantiza el orden de los elementos: La ordenación de los elementos no está garantizada, lo que significa que el orden en que se insertan puede ser diferente al orden en que se recogen

4.Permite claves y valores nulos: puede contener una clave nula y múltiples valores nulos

5.No es sincronizado: lo que significa que no es thread-safe (seguro para hilos) se refiere a código o estructuras de datos que pueden ser usados por múltiples hilos (threads) simultáneamente sin causar inconsistencias, errores o corrupción de datos

Ejemplo básico:

```
HashMap<String, String> telefonos = new HashMap<>();
telefonos.put("Mamá", "1122334455");
telefonos.put("Tatiana", "1199887766");

System.out.println(telefonos.get("Tatiana"));
```

Como recorrerlos?

Usando entrySet() (Recomendado para acceder a claves y valores)

```java
Map<String, Integer> edades = new HashMap<>();
edades.put("Ana", 25);
edades.put("Carlos", 30);
edades.put("María", 28);

// Método más eficiente (acceso a clave y valor)
for (Map.Entry<String, Integer> entrada : edades.entrySet()) {
    System.out.println(entrada.getKey() + ": " + entrada.getValue() + " años");
}
```

Salida:
Ana: 25 años
Carlos: 30 años
María: 28 años

OPERACIONES COMUNES

Operación	Ejemplo	Descripción
put(clave, valor)	map.put("llave);	Añade o actualiza
get(clave)	map,get("llave);	Obtiene valor
remove(clave)	map.remove("lave");	Elímina entrada
containsKey(clave)	map.containsKey("lave);	Verifica existencia
KeySet()	map.keySet();	Devuelve todos los valores
entrySet()	map.entrySet()	Devuelve pares clave-valor

códigofacilito

/05 Iteración con for-each y lambdas simples

El uso de `forEach` es uno de los más habituales en Java, ya que continuamente estamos recorriendo colecciones de objetos y la forma de recorrerlos es mucho más comoda que un bucle clásico solo usando una instrucción `forEach` sin necesidad de usar índices.
También tenemos los lambdas que son básicamente una función anónima que podés pasar como parámetro a otros métodos.
Java permite escribir código más conciso, sobre todo cuando hacés algo simple y repetido, como imprimir cada elemento. Veamoslo!

códigofacilito

For-Each (Iteración Simplificada)

Sintaxis básica

```
for (Tipo elemento : coleccion) {
    // Código a ejecutar con cada 'elemento'
}
```

En un array tradicional:

```
String[] frutas = {"Manzana", "Banana", "Naranja"};

// For-each tradicional
for (String fruta : frutas) {
    System.out.println(fruta);
}
```

Salida:
Manzana
Banana
Naranja

códigofacilito

For-Each (Iteración Simplificada)

En un arrayList :

```java
// Creación del ArrayList (mutabled)
ArrayList<String> frutas = new ArrayList<>();
frutas.add("Manzana");
frutas.add("Banana");
frutas.add("Naranja");

// Recorrido con for-each
System.out.println("Frutas en la lista:");
for (String fruta : frutas) {
    System.out.println("- " + fruta);
}
```

Salida:

```
Frutas en la lista:
-Manzana
-Banana
-Naranja
```

códigofacilito

Lambdas Simples:

Sintaxis básica

```
(parametros) -> { cuerpo }
// O si es una sola línea:
(parametros) -> expresión
```

En un arrayList:

```
// 1. Crear ArrayList de lenguajes
ArrayList<String> lenguajes = new ArrayList<>();
lenguajes.add("Java");
lenguajes.add("Python");
lenguajes.add("JavaScript");

// 2. Recorrer con forEach + lambda (forma más simple)
lenguajes.forEach(leng -> System.out.println(leng));
```

Salida:
Java
Python
JavaScript

códigofacilito

Cuándo usar for-each vs lambda?

Si:
*Querés que sea fácil de leer usa for-each
*Querés escribir código conciso usa .forEach() con lambda
*Vas a hacer más de 1 línea por ítem usa for-each (más claro)

EN RESUMEN:
for-each → cuando querés recorrer y operar uno a uno

.forEach(lambda) → cuando querés recorrer rápido y elegante, con funciones cortitas

códigofacilito

/06 HANDS ON

A codear, vamos al editor de código!

REPOSITORIO CON EL CÓDIGO:
https://github.com/tatianaborda/java-course-final

códigofacilito

Objetivo:
Aplicar todo lo aprendido en el bloque 6 creando un menú integrando:

ArrayList<Tarea> para mantener el orden de las tareas

HashMap<String, Tarea> para acceder rápidamente por ID

Herencia, encapsulamiento y polimorfismo (desde el Bloque 5)

Menú interactivo usando Scanner

Operaciones con for-each, get(), remove(), etc.

Actividad (abstracta)

Tarea (subclase concreta)

con menú interactivo y lógica