# Estructuras de Datos en Java

Las estructuras de datos son una de las áreas fundamentales en la informática y el desarrollo de software. En Java, un lenguaje de programación orientado a objetos muy popular, el manejo eficiente de estructuras de datos es esencial para construir aplicaciones rápidas, robustas y escalables. Este documento presenta una descripción detallada de algunas de las estructuras de datos más importantes disponibles en Java, sus características, usos y ejemplos prácticos.

## 1. ¿Qué son las estructuras de datos?

Una estructura de datos es una forma particular de almacenar y organizar datos en una computadora para que puedan ser usados de manera eficiente. Diferentes estructuras de datos tienen diferentes formas de almacenar y acceder a los elementos, y la elección adecuada de una estructura puede afectar significativamente el desempeño de un algoritmo o aplicación.

Java proporciona integradas muchas estructuras de datos a través de la biblioteca estándar java.util, pero también los programadores pueden crear sus propias estructuras.

## 2. Tipos Comunes de Estructuras de Datos en Java

### 2.1 Arrays (Arreglos)

Un array es una estructura básica que almacena elementos del mismo tipo en posiciones contiguas de memoria. En Java, los arrays tienen un tamaño fijo una vez creados y se accede a sus elementos mediante índices.

**Ejemplo de declaración y uso de un array:**

int[] numeros = new int[5];   
numeros[0] = 10;  
numeros[1] = 20;  
// y así sucesivamente

Los arrays son útiles cuando el número de elementos es conocido y se requiere acceso rápido mediante índice.

### 2.2 Listas (List)

La interfaz List en Java representa una colección ordenada de elementos que permite elementos duplicados y acceso posicional. La implementación más común es ArrayList, que proporciona un array que puede crecer dinámicamente.

**Características de ArrayList:**

* Dinámico en tamaño.
* Acceso rápido por índice.
* Inserción y eliminación en posiciones finales es eficiente.

**Ejemplo:**

import java.util.ArrayList;  
ArrayList<String> nombres = new ArrayList<>();  
nombres.add("Ana");  
nombres.add("Luis");  
System.out.println(nombres.get(1)); // Imprime Luis

### 2.3 Conjuntos (Set)

Un Set es una colección que no permite elementos duplicados, ideal para almacenar elementos únicos. La implementación más común es HashSet, la cual es eficiente para operaciones de inserción, borrado y búsqueda.

**Propiedades de HashSet:**

* No ordena los elementos.
* No permite duplicados.
* Operaciones rápidas gracias al uso de tablas hash.

**Ejemplo:**

import java.util.HashSet;  
HashSet<Integer> numerosUnicos = new HashSet<>();  
numerosUnicos.add(5);  
numerosUnicos.add(10);  
numerosUnicos.add(5); // No se añade porque ya existe

### 2.4 Mapas (Map)

Los mapas almacenan pares clave-valor, donde cada clave es única y se asocia a un valor. La implementación más popular es HashMap.

**Usos comunes:** Contar frecuencias, crear diccionarios, indexar datos para accesos rápidos.

**Ejemplo:**

import java.util.HashMap;  
HashMap<String, Integer> edades = new HashMap<>();  
edades.put("Carlos", 25);  
edades.put("Maria", 30);  
System.out.println(edades.get("Maria")); // 30

### 2.5 Pilas (Stack) y Colas (Queue)

La pila (stack) es una estructura LIFO (Last In, First Out), donde el último elemento añadido es el primero en salir. Por otro lado, la cola (queue) es FIFO (First In, First Out), donde el primer elemento que entra es el primero en salir.

**Uso habitual:**

* **Stack:** Evaluación de expresiones, llamadas de funciones, backtracking.
* **Queue:** Manejo de tareas en orden, impresión, procesamiento de datos.

**Ejemplo de una pila con la clase Stack:**

import java.util.Stack;  
Stack<String> stack = new Stack<>();  
stack.push("primero");  
stack.push("segundo");  
System.out.println(stack.pop()); // "segundo"

**Ejemplo de una cola con Queue (LinkedList):**

import java.util.LinkedList;  
import java.util.Queue;  
Queue<String> queue = new LinkedList<>();  
queue.add("primero");  
queue.add("segundo");  
System.out.println(queue.poll()); // "primero"

## 3. Estructuras de Datos Personalizadas

Además de las estructuras de datos que Java ofrece en su API estándar, es común que los desarrolladores creen sus propias estructuras para satisfacer necesidades específicas. Por ejemplo, árboles, listas enlazadas, grafos, etc.

**Ejemplo: Lista enlazada simple**

public class Nodo {  
 int valor;  
 Nodo siguiente;  
  
 public Nodo(int valor) {  
 this.valor = valor;  
 this.siguiente = null;  
 }  
}  
  
public class ListaEnlazada {  
 Nodo cabeza;  
  
 public void agregarAlFinal(int valor) {  
 Nodo nuevo = new Nodo(valor);  
 if (cabeza == null) {  
 cabeza = nuevo;  
 } else {  
 Nodo actual = cabeza;  
 while (actual.siguiente != null) {  
 actual = actual.siguiente;  
 }  
 actual.siguiente = nuevo;  
 }  
 }  
  
 public void imprimirLista() {  
 Nodo actual = cabeza;  
 while (actual != null) {  
 System.out.print(actual.valor + " -> ");  
 actual = actual.siguiente;  
 }  
 System.out.println("null");  
 }  
}

Esta lista permite agregar elementos al final y recorrerlos en orden.

## 4. Importancia y Buenas Prácticas

Seleccionar la estructura de datos adecuada impacta significativamente en la eficiencia de un programa. Por ejemplo:

* Para búsquedas rápidas se prefieren HashSet o HashMap.
* Para mantener orden se usan TreeSet o TreeMap.
* Si la inserción o eliminación frecuente en medio de la colección es necesaria, una LinkedList puede ser más eficiente que un ArrayList.

Además, es fundamental utilizar las interfaces adecuadas (List, Set, Map) para facilitar cambios futuros y mantener un código limpio y escalable.

## 5. Conclusiones

Las estructuras de datos son la base para construir algoritmos eficientes y aplicaciones de calidad. Java ofrece una rica biblioteca estándar con diversas estructuras listos para usar, pero también promueve la creación de estructuras personalizadas cuando sea necesario. Conocer las características, ventajas y limitaciones de cada estructura permite diseñar software más rápido y optimizado.

Por lo tanto, entender y dominar las estructuras de datos en Java es fundamental para cualquier programador que desee sobresalir en el desarrollo de software moderno.