# Resumen: Estructuras de Datos en Java

José Ángel Olmedo Guevara

## ¿Qué es una estructura de datos?

Lista de formas de organizar datos en memoria para optimizar operaciones como inserción, eliminación y acceso. Java incluye muchas implementaciones en el **Java Collections Framework**.

## Contenedores comunes en Java

- `Array` (arreglo de tamaño fijo)
- `ArrayList` (arreglo dinámico)
- `LinkedList` (lista doblemente enlazada)
- `Stack` (pila LIFO)
- `Queue` (cola FIFO)
- `HashSet` (conjunto sin duplicados)
- `HashMap` (mapa clave-valor)

## Ejemplos de uso

### Array

```
int[] arr = {1, 2, 3, 4};
System.out.println(arr[2]); // 3
```

### ArrayList

```
import java.util.ArrayList;

ArrayList<String> list = new ArrayList<>();
list.add("A");
list.add("B");
System.out.println(list.get(0)); // A
```

## LinkedList

```
import java.util.LinkedList;
import java.util.List;

List<String> listaLigada = new LinkedList<>();
listaLigada.add("X");
listaLigada.add(0, "Y");    // inserta al inicio
listaLigada.removeLast();   // elimina el ltimo
```

## Stack

```
import java.util.Stack;

Stack<Integer> stack = new Stack<>();
stack.push(10);
stack.push(20);
System.out.println(stack.pop()); // 20
```

## Queue

```
import java.util.Queue;
import java.util.LinkedList;

Queue<Integer> queue = new LinkedList<>();
queue.add(5);
queue.add(6);
System.out.println(queue.poll()); // 5
```

## HashSet

```
import java.util.HashSet;
import java.util.Set;

Set<Integer> hs = new HashSet<>();
hs.add(1);
hs.add(1);          // duplicado ignorado
System.out.println(hs); // [1]
```

## HashMap

```
import java.util.HashMap;
import java.util.Map;

Map<String,Integer> map = new HashMap<>();
map.put("Ana", 25);
```

```
map.put("Luis", 30);
System.out.println(map.get("Ana")); // 25
```

# Ventajas y Desventajas

## Array

**Ventajas:** - Acceso por índice en O(1) - Uso eficiente de memoria
**Desventajas:** - Tamaño fijo - Expansión costosa (hay que copiar a un arreglo nuevo)

## ArrayList

**Ventajas:** - Tamaño dinámico - Acceso aleatorio en O(1)
**Desventajas:** - Inserción/borrado en medio es O(n) - Realineamiento interno al crecer

## LinkedList

**Ventajas:** - Inserción/eliminación en cualquier posición en O(1) si se tiene el nodo - Útil para colas dobles (Deque)
**Desventajas:** - Acceso aleatorio en O(n) - Overhead extra por nodos enlazados

## Stack

**Ventajas:** - LIFO, ideal para recursión y backtracking
**Desventajas:** - Solo acceso al tope - No es thread-safe sin sincronización adicional

## Queue

**Ventajas:** - FIFO, útil para procesamiento por orden de llegada
**Desventajas:** - No permite acceso aleatorio - Operaciones limitadas a cabeza y cola

# Comparativa rápida

| Necesitas... | Usa |
|---|---|
| Acceso rápido por índice | `ArrayList` |
| Insertar/eliminar mucho | `LinkedList` |
| Evitar duplicados | `HashSet` |
| Buscar por clave | `HashMap` |
| Orden natural | `TreeSet` o `TreeMap` |
| Último en entrar, primero en salir | `Stack` |
| Primero en entrar, primero en salir | `Queue` |

# Notas adicionales

- Para usar estas estructuras debes importar sus paquetes, por ejemplo:

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
```

- En lugar de tipos primitivos (como `int`, `double`), usa sus clases envoltorio: `Integer`, `Double`, etc.