

Unidad 8

Estructuras Estáticas De Datos: ARRAYS

Objetivos

- Entender la necesidad de agrupar los datos en estructuras que permitan un manejo eficiente de los mismos.
- Aprender los conceptos asociados a arrays: índices, dimensiones, acceso a elementos individuales.
- Saber definir y procesar arrays paralelos.
- Saber definir y usar arrays de índices con significado.
- Saber definir y procesar un array bidimensional.
- String versus array de una dimensión de tipo carácter.
- Hacer uso de las herramientas incorporadas por Java para el uso y proceso de arrays y cadenas.
- Saber definir y usar la estructura de datos más adecuada a cada problema.

Contenidos

- INTRODUCCIÓN
- CONCEPTO DE ESTRUCTURA DE DATOS
 - Abstracción de datos
 - Clasificación de las estructuras de datos
- PERSPECTIVAS DE ANÁLISIS DE UNA ESTRUCTURA DE DATOS
- ARRAYS UNIDIMENSIONALES
 - Nivel Lógico
 - Nivel de Implementación
 - Nivel de aplicación o uso de arrays en programas.
- PASO DE ARRAYS A FUNCIONES
- ARRAYS BIDIMENSIONALES
 - Nivel lógico o abstracto
 - Nivel de implementación
 - Nivel de aplicación o de uso
- PASO DE ARRAYS DE DOS DIMENSIONES A FUNCIONES
- ARRAYS N-DIMENSIONALES
- CADENAS DE CARACTERES
- ARRAYS DE CADENAS

Introducción

- Las variables, sean simples o de instancias, son estructuras de datos simples que no son eficientes cuando se necesita procesar una cantidad importante de datos.
- Todos los lenguajes de programación incorporan estructuras de datos complejas, por ejemplo arrays, que pueden almacenar una gran cantidad de datos y que por tanto, harán los programas más sencillos y eficientes.
- Niklaus Wirth diseñador de: Euler, Algol W, Pascal, Modula, Modula-2 y Oberon
 - *Algoritmos + Estructuras de datos = Programas*

Estructura De Datos

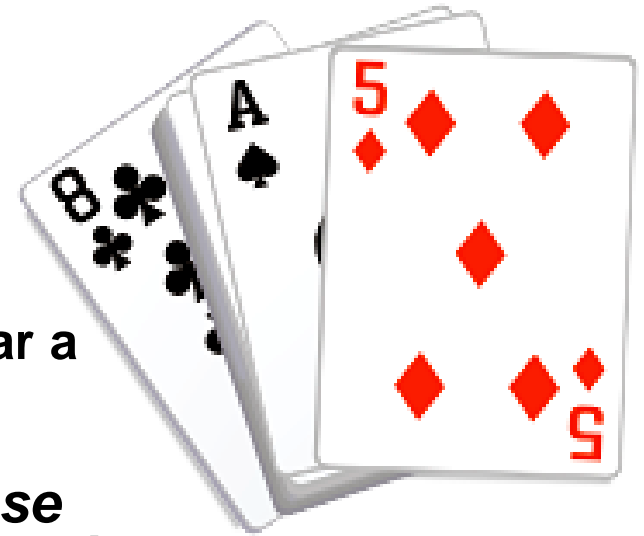
■ Ejemplo:

- Estructura **1º DAM** ⇒ para referenciar a todos los alumnos de este curso en lugar de hacerlo uno a uno.

■ **Definición:** conjunto de elementos que se caracteriza por su tipo de dato y la forma de almacenar y recuperar los elementos individuales del grupo.

■ Ejemplos:

- Estructura **Pared** ⇒ Tipo **Ladrillo** ⇔ Forma de acceder: **AlmacenarL** (poner ladrillo) y **SuprimirL** (quitar ladrillo)
- Estructura **Bombo bingo** ⇒ Tipo **Bola bingo** ⇔ Forma de acceder: **AlmacenarB** (meter bola) y **Suprimir** (sacar bola)

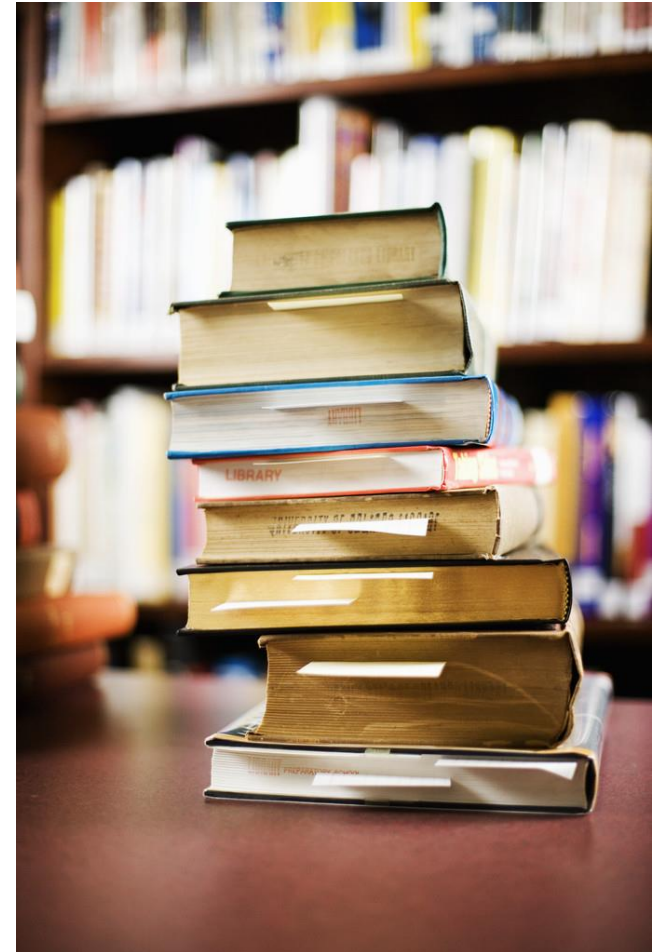


Estructura De Datos

- **Pilares fundamentales:**

Abstracción de datos: separación que existe entre la representación interna e implementación y las aplicaciones que las utilizan.

Encapsulamiento: encerrar, como en una cápsula, la definición de la estructura de datos y las funciones de acceso.



Estructura De Datos

- **Suministradas por Java:**
 - Arrays, Enumeraciones, Ficheros y clases.
 - El programador no se preocupa de su implementación.
- **No suministradas por Java :**
 - Listas, Pilas, Colas (¡ojo!, si implementadas como clases), Árboles y Grafos.
 - El programador debe implementarlas.
- **Clasificación.**
 - **Estáticas:**
 - ocupan una cantidad fija de memoria (debe ser conocida en tiempo de compilación).
 - **Dinámicas:**
 - ocupan una cantidad variable de memoria (se va decidiendo en tiempo de ejecución).

Análisis de una Estructura de datos

■ Nivel abstracto:

- Se define la organización de los datos y se especifican las formas de acceso.
- Ej. Cola de personas: Sucesión de elementos persona, donde colocar (almacenar) un elemento consiste en ponerlo tras el último que llegó y sacar (suprimir) un elemento supone sacar a todos los elementos que tiene delante.

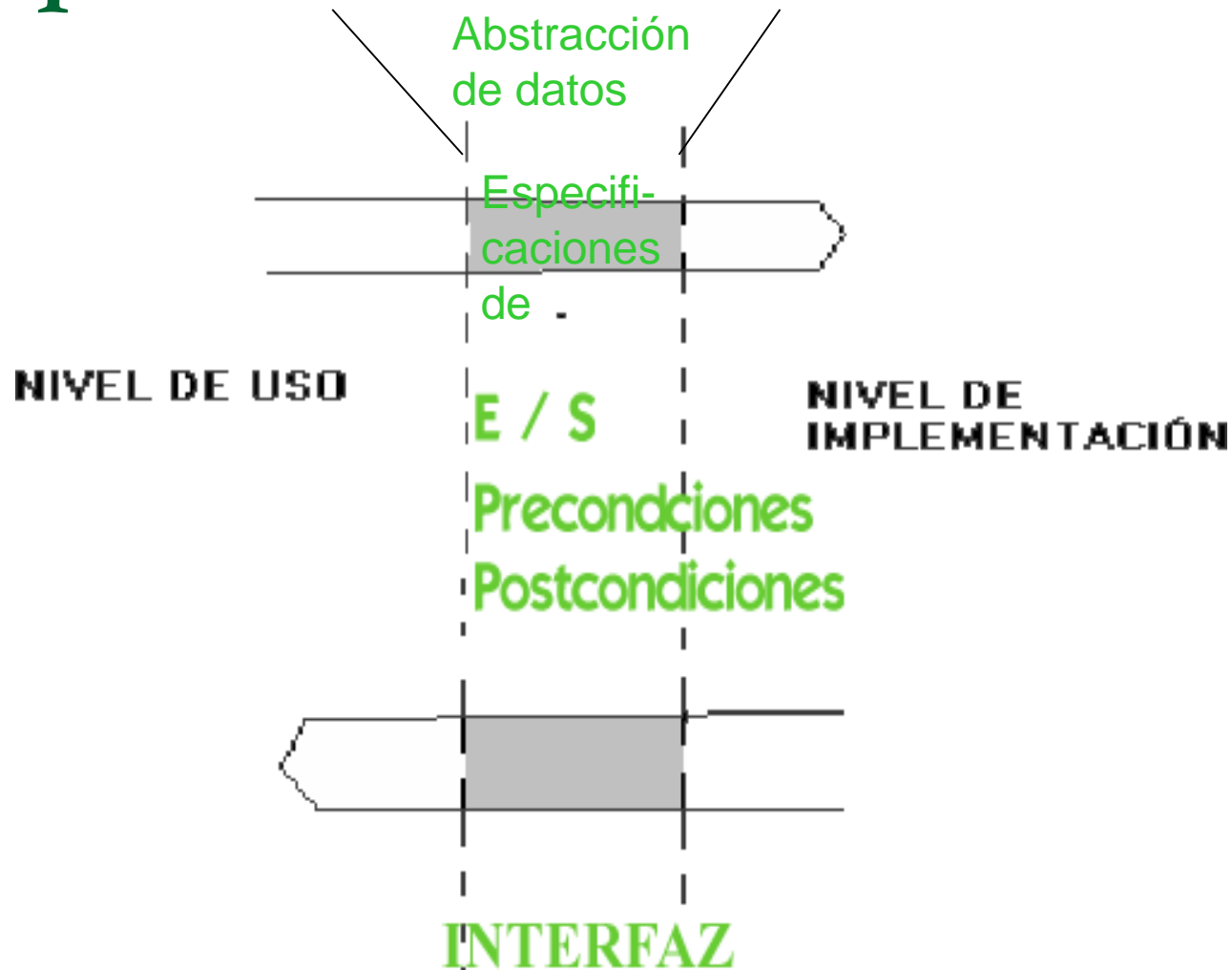
■ Nivel implementación:

- Se estudian las formas de representación de los datos en memoria y se implementan las funciones de acceso en un lenguaje de codificación.
- Ej. La estructura Cola puede ser implementada con array o dinámicamente,...

■ Nivel de uso o aplicación:

- Uso de la estructura en un contexto determinado.
- Ej. Cola del cine, Cola del autobús.

Comunicación entre nivel de uso e implementación



Arrays unidimensionales

Hay que analizarlos desde las tres perspectivas:

- Nivel lógico.
- Nivel implementación.
- Nivel de uso.

Arrays Unid.: Nivel lógico

- Definición: colección **finita** de elementos **homogéneos**, **ordenados**, que se referencian con un nombre común.
- Función de acceso: \Rightarrow **Estructura estática.**
 - Definición formal: Acceso directo a cada elemento a través de índice.
 - Operaciones definidas: Si Java no las tuviera definidas:
Las incorpora Java: **ConstruirArray**
AccederElemento
 - **Sintaxis:** ***NombreArray [expresión del índice]***
 - **Semántica:** se usa para referenciar un lugar donde:
Almacenar datos: **notas[2] = 10, notas [i] = 0** O bien,
Recuperar datos (para su tratamiento, no elimina): **var = notas[2]**
 - **Representación gráfica:** **pag. 9.**

Arrays Unid.: Nivel Implementación

1. Reservar memoria. 
2. Codificar las funciones de acceso.

Imprescindibles si la estructura no es incorporada por el lenguaje

- La reserva de memoria se hace con la declaración y creación.
En Java:

Tipo[] nombreArray = new tipo[tamaño];

o bien

Tipo nombreArray [] = new tipo[tamaño];

Sin embargo, la convención desaconseja esta última forma; los corchetes identifican el tipo contenedor *array* y deben aparecer junto al tipo de dato.

Arrays Unid.: Nivel Implementación

En Java las funciones de acceso están implementadas

■ Características :

- ❑ Cota superior = $n-1$
- ❑ Cota inferior = 0
- ❑ Base: es la dirección de la primera casilla, coincide con el *nombreArray*
- ❑ Tamaño en bytes del array.

$$\text{TamañoBytes} = \text{TamañoTipo} * \text{LongitudArray}.$$

- ❑ Localizar un elemento:

$$\text{Dirección de nombreArray} [\text{ÍndiceElemento}] = \text{Base} + \text{ÍndiceElemento} * \text{TamañoTipo}$$

ejemplo página 11

Arrays Unid.: Nivel de aplicación I

■ Operaciones prohibidas

- ❑ Inicializar o asignar de una vez

entero Array[] = 0

Tabla = 0

Tabla = Array //salvo si son objetos

- ❑ Leer o escribir de una vez

Leer (Tabla)

Escribir (“El array es:”, Tabla)

- ❑ Operaciones aritméticas de una vez

TotalAcumulado = Tabla + Array

Arrays Unid.: Nivel de aplicación I

■ Operaciones frecuentes

- ❑ Recorrido: **Ver ejemplos pag. 13**
- ❑ Lectura y escritura: **Pag 13-14**

- ❑ Inicialización

```
int [ ] array = {0, 1, 2, 3, 4};  
array[0] = 0;  
array[1] = 1;
```

....

```
for (i = 0; i < MAX; i++) // Recorrido para inicializar  
    array[ i ] = 0;
```

PruebaArray.java

→ `Tipo[] NombreArray ={lista de valores};`

ejemplos página 14

→ **Ejercicio:** programa que lea datos de teclado, los almacene en un array y a continuación pinte en pantalla los elementos que ocupan las posiciones pares y dicha posición. Se cuenta en el orden natural.

For extendido

- Aparece a partir de Java 5
- Realiza recorridos completos de colecciones
- No se necesita el número de elementos a recorrer.

- **Sintaxis:**

```
for (TipoARecorrer nomVariableTemp:nomArray)
{Instrucciones}
```

- ❑ Para cada elemento del tipo **TipoARecorrer** que se encuentre dentro de la colección **nomArray** ejecuta las **Instrucciones** que se indican.
- ❑ La variable local-temporal **nomVariableTemp**:
 - almacena en cada paso el objeto que se visita,
 - sólo existe durante la ejecución del ciclo y desaparece después.
 - Debe ser del mismo tipo que los elementos a recorrer.

Ejemplo de For extendido

```
class ForeachEjemplo{  
    public static void main(String args[]){  
        String [] arrStr = {"1", "2", "3", "4", "5"};  
        for(String elemento : arrStr) {  
            System.out.println(elemento);  
        }  
    }  
}
```

¡Ojo!, solo se utiliza si vamos a recorrer completamente el array desde la casilla primera a la última

Nivel de aplicación II

- **Índices con significado:** El índice tiene un contenido semántico. **Ejemplo PruebaArray.java**
- **Arrays paralelos:** Se procesan en orden casillas correspondientes.
- Paso de arrays a funciones:
 - Paso de la estructura completa.
 - Paso de un elemento en particular.**Pag. Ej. PasoParArrays.java**

Arrays bidimensionales

Hay que analizarlos desde las tres perspectivas:

- Nivel lógico.
- Nivel implementación.
- Nivel de uso.

Arrays Bid.: Nivel lógico

■ Definición:

■ Directa

Colección *finita* de elementos, *homogéneos*, *ordenados* en dos dimensiones, que se referencian bajo un nombre común.

⇒ Estructura estática.

■ Recursiva:

Es un array de una dimensión en el que cada elemento es a su vez un array de una dimensión.

Arrays Bid.: Nivel lógico

- Función de acceso:

- Definición: Acceso directo a través de dos índices.

Java la incorpora

Si Java no la tuviera definida:

ConstruirArray

AccederElemento


- Sintaxis: *Nombre_Array [expresiónIndice1] [expresiónIndice2]*
- Semántica: la función de acceso se usa para referenciar un lugar donde:

Almacenar datos: *notas[2] [1] = 10* o

Recuperar datos: *var = notas[2][1]*

- Representación gráfica: *ver pag. 21.*

Arrays Bid.: Nivel Implementación

1. Reservar memoria.  Imprescindibles si no las incorpora el lenguaje
2. Codificar las funciones de acceso.

- La reserva de memoria se hace con la declaración:

tipo [][]NombreArray

[= {lista de valores}];

Pag. 22

- Las funciones de acceso están implementadas, estudiaremos **características y funcionamiento interno** tenidos en cuenta para dicha implementación :
 - ❑ 2 Cotas superiores = FILAS-1,COLUMNAS-1
 - ❑ 2 Cotas inferiores = 0 y 0
 - ❑ Base: dirección de la primera casilla que coincide con el *nombreArray* que equivale a *nombreArray[0][0]*.
 - ❑ Tamaño_tipo (bytes) de un elemento.

Más sobre funcionamiento interno

- ❑ Tamaño_tipo, cantidad de bytes que ocupa un elemento.
- ❑ El tamaño en bytes de un array bid.:

$$\text{Tamaño_bytes} = \text{Filas} * \text{Columnas} * \text{Tamaño_tipo}$$

- ❑ Localizar un elemento:

$$\text{nombreArray} [\text{FIL}][\text{COL}] = \text{Base} + (\text{Columnas} * \text{FIL} + \text{COL}) * \text{Tamaño_tipo}$$

ejemplo en apuntes

Nivel de aplicación

■ Operaciones prohibidas

- Las mismas que los arrays unidimensionales.

■ Operaciones frecuentes

1. Acceso aleatorio o directo a cualquier elemento.
2. **Recorrido del array:** Acceso de forma sistemática a cada elemento por filas o por columnas.
3. Procesar una fila. **Ver página 26**
4. Procesar una columna.
5. Procesar una o las dos diagonales.

Nivel de aplicación

- **Paso de arrays bidimensionales a funciones**
 - función.
`Int [][] Func1 (int filas, int columnas)`
 - procedimiento.
`void Proced1 (int [][] Arr);`
 - La llamada sigue realizándose, como con los arrays de una dimensión, enviando el nombre del array.
`Proced1 (matriz);`
`matriz = Func1 (fil, col);`

**Ejercicio página 29
(Análisis)**

Ejercicio

- **Implementar una clase para manejo de matrices matemáticas que realice operaciones sobre matrices cuadradas:**
 - ❑ Cargar matrices aleatoriamente y desde teclado
 - ❑ Imprimir
 - ❑ Inicializar a cero
 - ❑ Crear la matriz unidad
 - ❑ Sumar matrices
 - ❑ Restar
 - ❑ Multiplicar
 - ❑ Hallar la traspuesta
 - ❑ Sumar los elementos de la diagonal principal

Cadenas/arrays de cadenas

- Una cadena es un array de tipo char y puede ser recorrido igual que el resto de arrays de cualquier otro tipo de datos, **pero no en Java.**

```
String array; //inmutable  
char[] array;
```

- Un array de cadenas puede ser tratado como un array de una dimensión y como un array de dos dimensiones.

```
String [] array //inmutable  
char[][]array
```

“Si piensas que los usuarios de tus programas son idiotas, sólo los idiotas usarán tus programas”

Linus Torvalds - "padre" de Linux