PROMETEO

Arrays

Índice

1

Introducción

PROMETEO

01

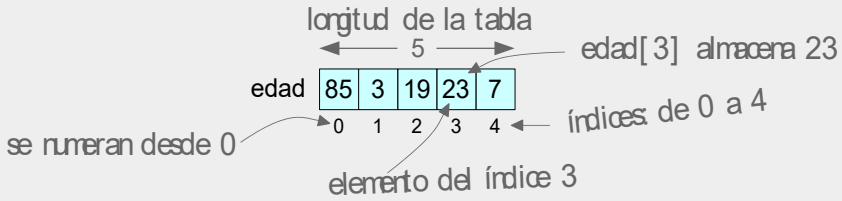
Introducción

- Una **variable escalar** es aquella que permite **almacenar** únicamente **un valor** en cada instante.
- Un **‘array’** (tabla, vector, matriz, arreglo…) es una variable que permite guardar más de un valor simultáneamente.
- Los ‘array’ se utilizan cuando se necesita almacenar y manipular **varias variables de un mismo tipo** y que están relacionadas con un mismo concepto.
- Así, se almacenan simultáneamente varios datos en una estructura que se puede entender como una lista/secuencia de valores.

01

Índices

- Para **acceder o modificar** cualquiera de los valores almacenados en un array es imprescindible considerar la posición que ocupan en la estructura.
- La **posición** que ocupa cada elemento es lo que se conoce como **índice**.
- Los índices se numeran **desde 0 hasta la longitud del array -1**.



01
Índices

- **Sintaxis de acceso** a un elemento de un array: *nombreArray*[*pos*] , donde 'pos' es el índice del elemento

estaciones =

primavera	verano	otoño	invierno
0	*1*	*2*	*3*

estaciones[1] *//verano*

```
edad[0] = 23;     //asigna el valor 23 a la primera posición del array referenciado
System.out.println (edad[2]); // imprime la tercera posición del array referenciado
```

- Hacer referencia a un índice que no existe en un array provoca una finalización abrupta del programa en ejecución. Es lo que se conoce **como índice fuera de rango** (ArrayIndexOutOfBoundsException) y es una de las principales cuestiones a tener en cuenta al utilizar arrays.

01

Construcción

1. **Decidir** qué **tipo de dato** se va a almacenar y el **número de elementos** (longitud).

2. **Declarar** el array. Las siguientes expresiones son equivalentes:

```
Tipo nomVariable[];    // int edad[];
Tipo[] nomVariable;    // int[] edad;
```

3. **Crear** el array:

a) Utilizando el operador **new**

```
int edad[]; //declara la variable edad
edad = new int[5]; //asigna a 'edad' un array de long 5
```

b) Asignando los valores en el momento de la declaración, sin utilizar new

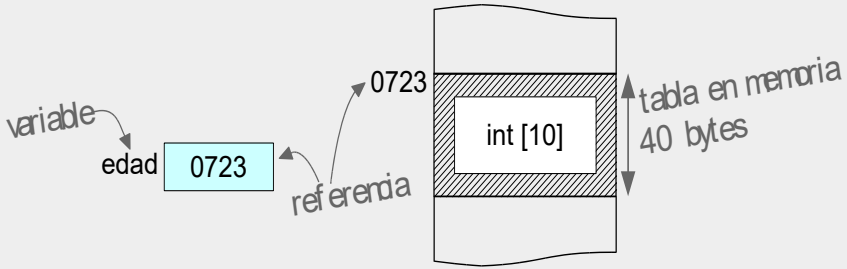
```
int edad[] = {12, 34, 78};  // utilizar solo en el mismo momento de la declaración
```

01

Referencias

Una vez creado el array con el operador ‘new’:

- Los elementos se **inicializan con valores por defecto**:
  - Numéricos : 0
  - Booleanos: false
  - Otros: null
- Se **reserva la memoria** necesaria para ubicar la nueva estructura



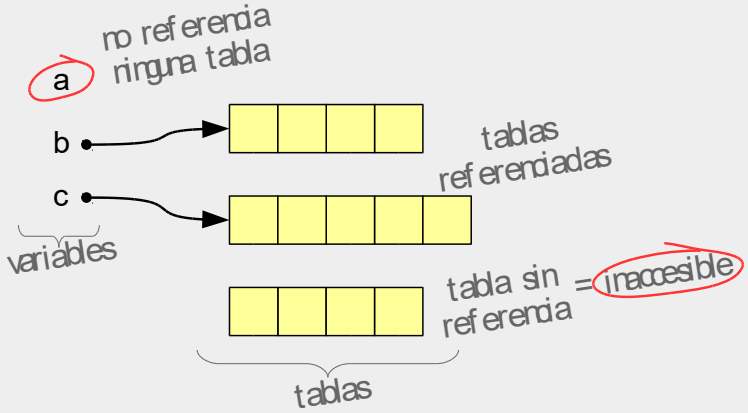

```
edad = new int[10]; //asigna  la referencia
System.out.println(edad) ; // Imprime la referencia, no el contenido del array
```

01

Referencias

- El **recolector de basura** en Java (Garbage Collector, GC) es un mecanismo automático que periódicamente revisa qué objetos son inaccesibles o no están en uso y los elimina. Este proceso libera espacio de memoria y mejora el rendimiento de los programas.

```
int a[], b[], c[]; // variables
b = new int[4];
c= new int[5];
new int[4];
```

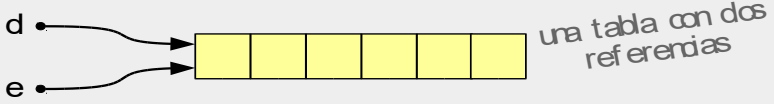


01

Referencias

- Es posible acceder a un array desde varias variables diferentes, para ello:
  - el array debe estar referenciado por dichas variables.
  - las variables deben ser del mismo tipo.

```
int d[], e[]; // variables
d = new int[6]; // construye array referenciado por d
e= d; // la variable e referencia el mismo array que d.
```



2

Uso de arrays

PROMETEO

02

Técnicas de manipulación

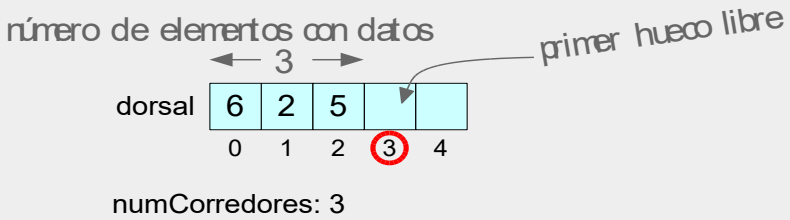
Estrategias para trabajar con arrays:

1. **Todos los elementos contienen información útil** y se adapta la longitud del array a las necesidades de cada momento.
    - Esta opción simplifica la resolución de los problemas y permite aprovechar las herramientas que proporciona java.
    - Será la técnica elegida habitualmente.
    - Para **modificar la longitud** de un array:
        - Crear un **nuevo array** con la longitud adecuada
        - **Copiar** los datos del array original en el nuevo
        - **Referenciar el nuevo array** con la misma variable que referencia el array original

02

Técnicas de manipulación

2. **Solo algunos elementos contienen información útil**
   - El array se divide en 2 partes, una con datos útiles y otra con datos no relevantes, o posiciones vacías.
   - Para manipular el array es necesario utilizar una **variable auxiliar** (entero) que almacene el número de elementos útiles, valor que coincide con la primera **posición 'libre'**.
   - No es necesario modificar la longitud al insertar o eliminar elementos, únicamente se tiene que actualizar la variable auxiliar



02

length() – toString()

Clase Arrays y métodos/propiedades útiles:

- Para obtener la **longitud de un array:** *variableArray.length*

```
int longitud = estaciones.length;      //length es un entero
```

- Clase **Arrays**: tiene métodos estáticos que facilitan el tratamiento de arrays. Se ubica en el paquete java.utils.Arrays

- **Arrays.toString**: método estático para mostrar el contenido de un array

```
int t[] = {1,2,3,4}; //declara t y asigna array de números enteros
System.out.println(Arrays.toString(t));  // imprime [1,2,3,4]
```

02

foreach

foreach: Para recorrer un array podemos utilizar un for, pero también existe una sintaxis específica para recorrer todos los elementos de un array.

```
for (tipo variableAuxiliar : Nombrearray){
        Bloque de instrucciones
}
```

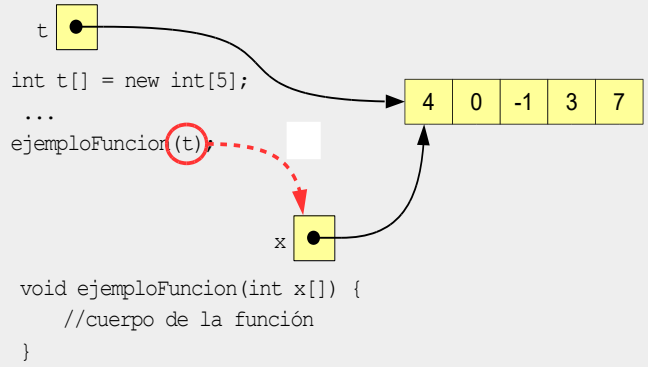
- Se utiliza cuando se quiere analizar el contenido de un array ya inicializado, por ejemplo, para búsquedas, cuenta de elementos, etc...
- Con el uso del for-each, se avanza una posición del array en cada iteración, hasta llegar al final.

```
double arraySueldos[] = {1000, 2000, 2500.38};
double sumaSueldos = 0;
for (double auxSueldo : arraySueldos){ sumaSueldos += auxSueldo;}
```

02

array como parámetro

- El mecanismo de paso de parámetros es siempre el mismo: el valor de la variable (sea escalar o array) se copia en el parámetro.
- La variable **t**, que referencia al array de datos, **copia su referencia a** la variable **x** (parámetro de la función).
- **Tanto t como x referencian al mismo array**



** La modificación de un elemento del array desde la función es visible para la variable externa. Ambas referencian el mismo array.*

3

La clase Arrays

PROMETEO

03

Inicialización

fill: Método estático de la clase **Arrays** que inicializa los elementos de un array.

- Este método está sobrecargado, se puede utilizar con cualquier tipo de variable primitivo y con diferente número de parámetros:

a) Todos los elementos con un mismo valor

*static void **fill**(tipo[] t, tipo valor)*

b) Algunos elementos con un valor determinado. Actualiza los elementos del array t, comprendidos entre los **índices** 'desde' y 'hasta', sin incluir este último, con valor

*static void **fill**(tipo t[], int desde, int hasta, tipo valor)*

Ejemplo:

```
Arrays.fill(sueldos, 3, 7, 1234.56); //inicializa el rango 3..6 con valor 1234,56
```

03

Comparación

equals: Método estático de la clase **Arrays** que compara el contenido de 2 arrays.

*static boolean **equals**(tipo a[], tipo b[])*

- Compara los arrays elemento a elemento.
- Devuelve 'true' si son iguales, 'false' en caso contrario.

Ejemplo:

```
boolean sonIguales = Arrays.equals(array1, array2);
System.out.println(sonIguales?"Son iguales":"No son iguales");
```

03

Ordenación

sort: Método estático de la clase **Arrays** que ordena los elementos de un array.

- Ordena los elementos del array de forma creciente.
- El método está sobrecargado para cualquier tipo primitivo

*static void **sort**(tipo t[])*

Ejemplo:

```
int edad = {85, 19, 3, 23, 7}; //tabla desordenada
Arrays.sort(edad); //ordena la tabla. Ahora edad = [3, 7, 19, 23, 85]
```

03

Búsqueda

La **eficiencia** del algoritmo de **búsqueda** de un elemento en un array depende de si los elementos están ordenados o no.

a) Búsqueda **secuencial** en un array **no ordenado**

```
/* búsqueda secuencial */
int indiceBusqueda = 0; //índice que usamos para recorrer la tabla
while (indiceBusqueda < t.length && //no es el último elemento
   t[indiceBusqueda] != claveBusqueda) { //y no encontrado
   indiceBusqueda++; //incrementamos el índice de búsqueda
}
if (indiceBusqueda < t.length) {
   ... //claveBusqueda se encuentra en la posición indiceBusqueda
} else { //el índice se ha salido de rango
   ... //no encontrado
}
```

Se sale del bucle cuando se encuentra la primera coincidencia o cuando se llega al final del array.

En el peor de los casos, se recorre el array completo.

03

Búsqueda

b) Búsqueda **binaria (dicotómica)** en un array **ordenado.** Tener los elementos ordenados permite aplicar un algoritmo más eficiente que el de la búsqueda secuencial:

i. Se consulta el elemento central del array y se compara su valor con el valor que se está buscando.

i. Si el valor central es mayor que el elemento que se busca, se reduce la búsqueda a la mitad de la izquierda del array.

ii. Si el valor central es menor que el elemento que se busca, se reduce la búsqueda a la mitad de la derecha del array.

Este proceso se repite con la mitad seleccionada de forma sucesiva hasta localizar la clave de búsqueda o hasta que ya no queden posiciones en el array.

03

Búsqueda

binarySearch: Método estático de la clase **Arrays** que implementa la búsqueda binaria (dicotómica).

*static int **binarySearch**(tipo t[], tipo claveBusqueda)*

- Devuelve a) el índice donde se encuentra la primera ocurrencia del valor buscado
  b) un valor negativo* si no se encuentra el valor

- El valor negativo que devuelve binarySearch está identificando a su vez la posición en la que tendría que insertarse dicho valor para incluirlo y mantener el array ordenado:

Ejemplo:

```
int a[] = {2, 4, 5, 6, 9};
int pos = Arrays.binarySearch(a, 3); //pos vale -2
int indiceInsercion = -pos - 1; //-(-2)-1 = 1 -> el 3 iría en la 2ª posición.
```

03

Búsqueda

binarySearch está **sobrecargado** y también se puede utilizar para **buscar** un valor en un **rango de posiciones** del array.

*static int **binarySearch**(tipo t[], int desde, int hasta, tipo claveBusqueda)*

- Busca en el array t el valor claveBusqueda en el rango de posiciones [desde, hasta**-1**]

03

Copia

copyOf: Método estático de la clase **Arrays** que crea un array copia de otro dado.

*static tipo[] **copyOf**(tipo origen[], int longitud)*

- Construye y devuelve una **copia** del array origen **con la longitud indicada**
  - Si longitud < longitud original: solo se copian los elementos que caben.
  - Si es mayor: quedan posiciones vacías

Ejemplo:

```
int t[] = {1, 2, 1, 6, 23}; //array origen
int a[], b[]; // referencias a los array destino
a = Arrays.copyOf(t, 3); //a = [1, 2, 1]
b = Arrays.copyOf(t, 10); //b = [1, 2, 1, 6, 23, 0, 0, 0, 0, 0]
```

03

Copia

copyOfRange: Método estático de la clase **Arrays** construye y **devuelve un array** copia de un rango de elementos de un array original.

*static tipo[] **copyOfRange**(tipo origen[], int desde, int hasta)*

- Devuelve un array donde se han copiado los elementos de origen comprendidos entre los índices [desde, hasta**-1**]

Ejemplo: Copia desde los índices 1 al 3 de 't' al array 'a'

```
int t[] = {7, 5, 3, 1, 0, -2};
int a[] = Arrays.copyOfRange(t, 1, 4); //a = [5, 3, 1]
```

03

Copia

arraycopy: Método de la clase **System** copia un rango de elementos consecutivos de un array origen a un array destino.

void arraycopy(Object arrayOrig, int posOrig, Object arraydest, int posDest, int longitud)

- Se **sobrescriben** los elementos en el array destino
- Hay que controlar que no se intentará escribir en posiciones fuera de rango en el destino
- Hay que controlar que no se intentará copiar/leer de posiciones fuera de rango en el origen
- Los arrays deben ser de **tipos compatibles** y **deben existir** antes de invocar al método

03

Copia

Ejemplo: arraycopy

```
// origen
int[] arrayOrigen = {10, 20, 30, 40, 50};

// destino (debe tener espacio suficiente)
int[] arrayDestino = {0, 0, 0, 0, 0, 0};


// Copiando 3 elementos desde origen a destino
System.arraycopy(arrayOrigen, 1, arrayDestino, 2, 3);

// imprime destino
System.out.println(Arrays.toString(arrayDestino)); //[0, 0, 20, 30, 40, 0]
```


PROMETEO