

MP0485
Programación
UF4. Estructuras de datos compuestos

4.1. Arrays

Índice

☰	Objetivos	3
☰	Concepto de array	4
☰	Trabajar con arrays en Java	9
☰	Los arrays son objetos	16
☰	Búsqueda en arrays	21
☰	Ordenación de arrays	26
☰	Resumen	27

Objetivos

Con esta unidad perseguimos los siguientes objetivos:

1

Implementar programas Java que utilicen *arrays* de tipo vector, matriz o cubo.

2

Realizar las típicas tareas de procesamiento de *arrays*: carga de datos, recorrido, búsqueda y ordenación.

¡Ánimo y adelante!

Concepto de array

Recuerda, un *array* es un conjunto de elementos organizados en memoria en posiciones contiguas. Todos los elementos son del mismo tipo y se puede acceder a ellos a partir de subíndices.

La **dimensión** del *array* es el número de subíndices que son necesarios para acceder a un elemento.

La **longitud** de un *array* es el número de elementos que puede almacenar.

Existen distintos tipos de *arrays* en función del número de subíndices necesarios para acceder a cada elemento, es decir, según su dimensión.

Aunque ya estudiaste los *arrays* en una lección anterior, ahora tendrás oportunidad de repasarlos y ver cómo se definen en Java.

Vector (*array* de una dimensión)

Los **vectores** son *arrays* de una dimensión porque para acceder a cada uno de los elementos solo necesitamos **un subíndice**. Para acceder al elemento nº 7 (Lucas) lo hacemos así:

nombre[7]

nombre	
0	Pepe
1	Juan
2	Luis
3	Alicia
4	Rosa
5	Carlos
6	Pedro
7	Lucas
8	María
9	Carlota

Vector

Los *arrays* en Java son objetos y los objetos se crean con el operador *new*.

En Java puedes declarar el *array nombre* con capacidad para 10 elementos de cualquiera de estas dos formas:

```
String nombre[] = new String[10];  
String []nombre = new String[10];
```

Declarar vector en Java.

Matriz (*array* de dos dimensiones)

La siguiente imagen muestra una matriz que representa un parking con 10 plantas (10 filas) y 7 plazas en cada planta (7 columnas).

		plaza						
		0	1	2	3	4	5	6
0	libre	libre						
1	134BJK	libre	libre	libre	libre	libre	libre	libre
2	libre	libre						
3	libre	2345HPJ	libre	libre	libre	libre	libre	libre
4	libre	libre	libre	7654ABJ	libre	5429PJA	libre	libre
5	libre	libre						
6	3395RJK	libre	libre	libre	3357BHA	libre	libre	libre
7	libre	libre	9954PRJ	libre	libre	libre	libre	libre
8	libre	libre	libre	libre	libre	libre	5594PJR	libre
9	libre	libre						

Matriz.

Las **matrices** son *arrays* de dos dimensiones porque para acceder a cada uno de los elementos necesitamos **2 subíndices**.

Para acceder al elemento de la fila 4 y la columna 3 (matrícula 7654ABJ) lo hacemos así:

plaza[4][3]

Puedes crear la matriz de cualquiera de estas dos formas:

```
String plaza[][] = new String[10][7];
String [][]plaza = new String[10][7];
```

Matriz en Java.

Cubos (*array de tres dimensiones*)

plaza										
	0	1	2	3	4	5	6	0	1	2
0	libre									
1	134BJK	libre	libre	libre	libre	libre	libre			
2	libre									
3	libre	2345HPJ	libre	libre	libre	libre	libre			
4	libre	libre	libre	7654ABJ	libre	5429PJA	libre			
5	libre									
6	3395RJK	libre	libre	libre	3357BHA	libre	libre			
7	libre	libre	9954PRJ	libre	libre	libre	libre			
8	libre	libre	libre	libre	libre	libre	5594PJR			
9	libre									

Cubo

Los **cubos** son *arrays de tres dimensiones* porque para acceder a cada uno de los elementos necesitamos **3 subíndices**.

Para acceder al elemento de la fila 4, la columna 3 y profundidad 0 (matrícula 7654ABJ) lo hacemos así:

plaza[4][3][0]

Puedes crear el cubo de cualquiera de estas dos formas:

```
String plaza[][][] = new String[10][7][3];
String [][] []plaza = new String[10][7][3];
```

Arrays de más de tres dimensiones

Java permite crear *arrays* de más de 3 dimensiones, aunque no sería posible dibujarlas.

Cualquiera de estas dos sentencias sería correcta para declarar un *array* de cuatro dimensiones:

```
String plaza[][][][] = new String[10][7][5][3];  
String [][]][]plaza = new String[10][7][5][3];
```

Array de 4 dimensiones en Java.

Y este sería un ejemplo para acceder a uno de los elementos:

plaza[4][3][2][1]

Trabajar con arrays en Java

Java almacena los *arrays* como un tipo especial de objeto con sus propiedades y métodos. En este apartado veremos un ejemplo de vector en Java y otro ejemplo de matriz. Los subíndices de *array* en Java siempre comienzan con el valor 0.

Ejemplo de vector en Java

```
public class Principal {  
    public static void main(String[] args) {  
  
        // Declaración del vector.  
        String nombre[] = new String[10];  
  
        // Inicialización de los valores del vector.  
        nombre[0] = "Pepe";  
        nombre[1] = "Juan";  
        nombre[2] = "Luis";  
        nombre[3] = "Alicia";  
        nombre[4] = "Rosa";  
        nombre[5] = "Carlos";  
        nombre[6] = "Pedro";  
        nombre[7] = "Lucas";  
        nombre[8] = "María";  
        nombre[9] = "Carlota";  
  
        // Recorrido del vector.  
        for (int i=0; i<nombre.length; i++) {  
            System.out.println(nombre[i]);  
        }  
    }  
}
```

Vector en Java.

Para recorrer los elementos de un vector necesitamos una estructura repetitiva de tipo *para* (*for*) con un contador que tome los valores de las distintas posiciones del *array*.

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

Recorrido de un vector en Java.

La propiedad *length* contiene el **número total de elementos del array**, en nuestro ejemplo contendrá el valor 10. Como las posiciones van del 0 al 9, la condición *i<nombre.length* llegará hasta el 9.

También es posible declarar el vector asignándole directamente los valores o elementos encerrándolos entre llaves de la siguiente forma:

```
public class Principal {  
    public static void main(String[] args) {  
        // Declaración del vector.  
        String nombre[] = {"Pepe", "Juan", "Luis", "Alicia", "Rosa"  
  
        // Recorrido del vector.  
        for (int i=0; i<nombre.length; i++) {  
            System.out.println(nombre[i]);  
        }  
    }  
}
```

Otra forma de declarar un vector.

Ejemplo de matriz en Java

Antes de nada es conveniente que comprendas que en Java realmente solo hay vectores. **Una matriz es internamente un vector donde cada uno de los elementos contiene otro vector interno.** El siguiente gráfico muestra de manera más certera el formato interno de una matriz en Java:

plaza							
	0	1	2	3	4	5	6
0	libre						
1	0	1	2	3	4	5	6
1	1234BJK	libre	libre	libre	libre	libre	libre
2	0	1	2	3	4	5	6
2	libre	libre	libre	libre	9948_IP	libre	libre
3	0	1	2	3	4	5	6
3	libre	2345HPJ	libre	libre	libre	libre	libre
4	0	1	2	3	4	5	6
4	libre	libre	libre	7654ABJ	libre	5429PJA	libre
5	0	1	2	3	4	5	6
5	libre						
6	0	1	2	3	4	5	6
6	3395RJK	libre	libre	libre	3357BHA	libre	libre
7	0	1	2	3	4	5	6
7	libre	libre	9954PRJ	libre	libre	libre	libre
8	0	1	2	3	4	5	6
8	libre	libre	libre	libre	libre	libre	5594PJR
9	0	1	2	3	4	5	6
9	libre						

Matriz en Java.

En conclusión, lo que existe en Java son vectores, cuyos elementos contienen otros vectores.

La propiedad ***length***, aplicada al nombre del vector, contiene el **número de elementos de nivel superior**, es decir, de la primera dimensión.

Prueba el siguiente código:

```
String plaza[][] = new String[10][7];
System.out.println(plaza.length);
```

Averiguar la longitud del vector: propiedad *length*.

Obtendrás como resultado un 10, el número de filas o de elementos del array *plaza*. Dentro de cada una de esas filas o elementos hay otro array con 7 elementos.

Añade ahora esta línea:

```
System.out.println(plaza[2].length);
```

En este caso estás obteniendo el número de elementos del array interno que está dentro de la posición 2, es decir, un 7.

Ahora prueba un ejemplo completo:

```
public class Principal {  
    public static void main(String[] args) {  
        // Construcción de la matriz (vector de vectores).  
        String plaza[][] = new String[10][7];  
  
        // Primero rellenamos todas las plazas como libres.  
        for (int f=0; f<plaza.length; f++) {  
            for (int c=0; c<plaza[0].length; c++) {  
                plaza[f][c]="libre";  
            }  
        }  
  
        // Ahora entran coches.  
        // Asigno a las plazas ocupadas la matricula correspondiente  
        plaza[1][0]="1234BJK";  
        plaza[2][4]="9948LIP";  
        plaza[3][1]="2345HPJ";  
        plaza[4][3]="7654ABJ";  
        plaza[6][0]="3395RJK";  
        plaza[6][4]="3357BHA";  
        plaza[7][2]="9954PRJ";  
        plaza[8][6]="5594PJR";  
  
        // Recorremos el array (las plazas).  
        for (int f=0; f<plaza.length; f++) {  
            for (int c=0; c<plaza[0].length; c++) {  
                System.out.println("Plaza "+f+". "+c+  
": "+plaza[f][c]);  
            }  
        }  
    }  
}
```

Arrays dentro de otros arrays

Ahora que has comprendido el funcionamiento interno de los *arrays* y que dentro de un *array* puede haber otro *array*, es momento de que veas que cada uno de esos *arrays* internos puede ser de distinta longitud. Veamos un ejemplo:

```
public class Principal {  
    public static void main(String[] args) {  
  
        /*  
         Construimos el array sin especificar número de columnas.  
         Datos es un array que contiene referencias a otros arrays  
         que tendrán que ser construidos en otro momento.  
        */  
        int datos[][] = new int[5][];  
  
        // Construcción arrays internos con longitudes diferentes.  
        datos[0] = new int[4];  
        datos[1] = new int[5];  
        datos[2] = new int[8];  
        datos[3] = new int[3];  
        datos[4] = new int[9];  
  
        // Rellenamos los elementos con números al azar entre 0 y 9  
        for (int f=0; f<datos.length; f++) {  
            for (int c=0; c<datos[f].length; c++) {  
                datos[f][c]= (int) (Math.random()*10);  
            }  
        }  
  
        // Recorremos el array mostrando los valores en pantalla.  
        for (int f=0; f<datos.length; f++) {  
            for (int c=0; c<datos[f].length; c++) {  
                System.out.print(datos[f][c]+" ");  
            }  
            System.out.println(); // Retorno de carro.  
        }  
    }  
}
```

Veamos ahora otra forma de crear una matriz asignando los valores directamente:

```
public class Principal {  
    public static void main(String[] args) {  
  
        int datos[][] = {  
            {1,3,5,7},  
            {3,5,7,9,11,13},  
            {2,4},  
            {8,9,10}  
        };  
  
        // Recorremos el array mostrando los valores en pantalla.  
        for (int f=0; f<datos.length; f++) {  
            for (int c=0; c<datos[f].length; c++) {  
                System.out.print(datos[f][c]+" ");  
            }  
            System.out.println(); // Retorno de carro.  
        }  
    }  
}
```

Los arrays son objetos

Los *arrays* de Java son objetos y por consiguiente, las variables utilizadas para declarar *arrays* son referencias a objetos.

Observa el siguiente programa Java:

```
public class Principal {  
    public static void main(String[] args) {  
        int valor[];  
        valor = new int[4];  
  
        valor[0]=3;  
        valor[1]=7;  
        valor[2]=25;  
        valor[3]=12;  
  
        int numeros[] = valor;  
        numeros[2]=252;  
  
        System.out.println(valor[2]);  
    }  
}
```

Crea un *array* de 4 elementos denominado *valor*. La variable *valor* es una referencia a un objeto y el objeto es el *array*. Cuando se ejecuta esta sentencia:

```
int numeros[] = valor;
```

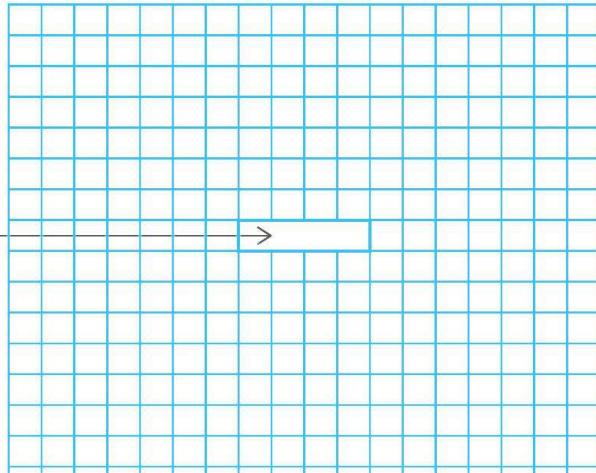
Lo que tenemos es otra referencia que apunta al mismo objeto, es decir, al mismo *array*.

Vamos a ver paso a paso lo que ocurre en la memoria RAM cuando ejecutamos el anterior programa:

int valor[];

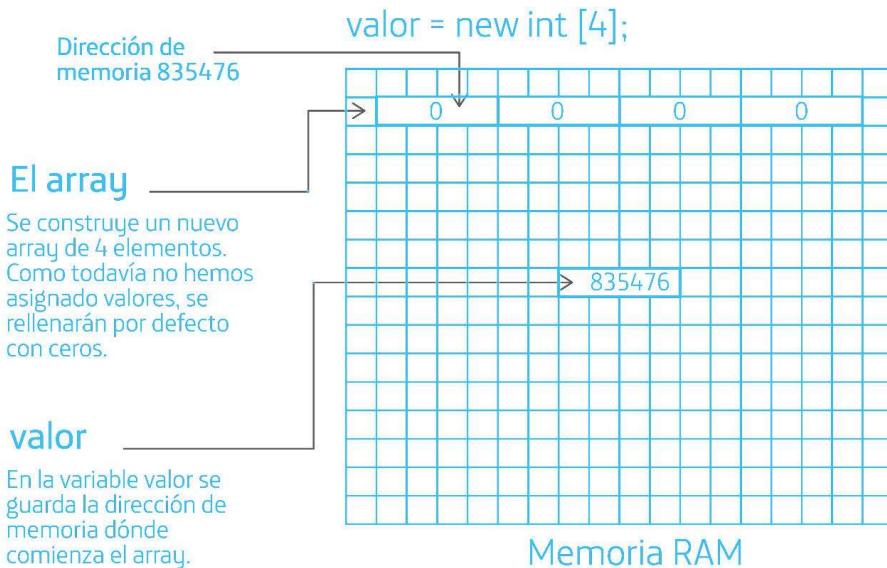
valor

Se reserva un espacio en memoria para la variable *valor* que contendrá la dirección de memoria en un futuro array de números enteros.

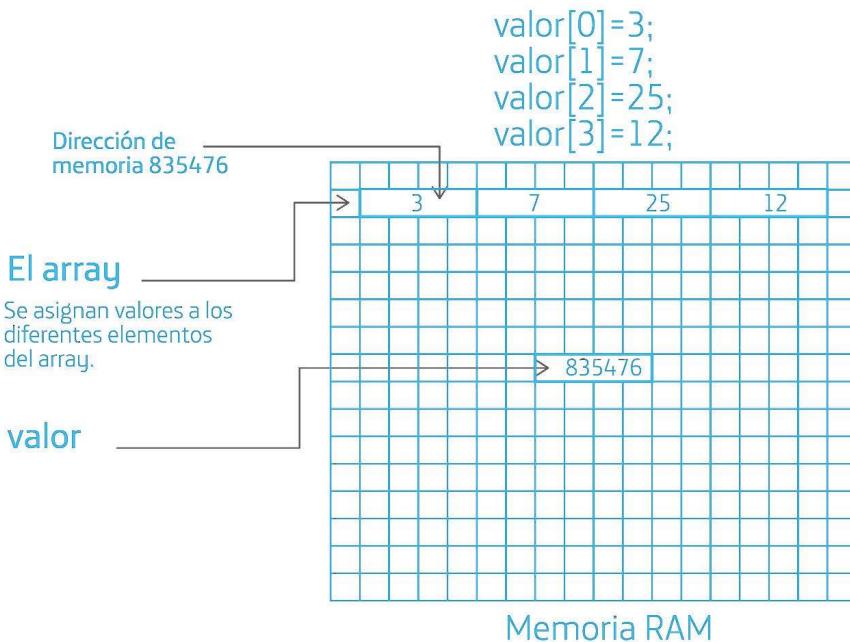


Memoria RAM

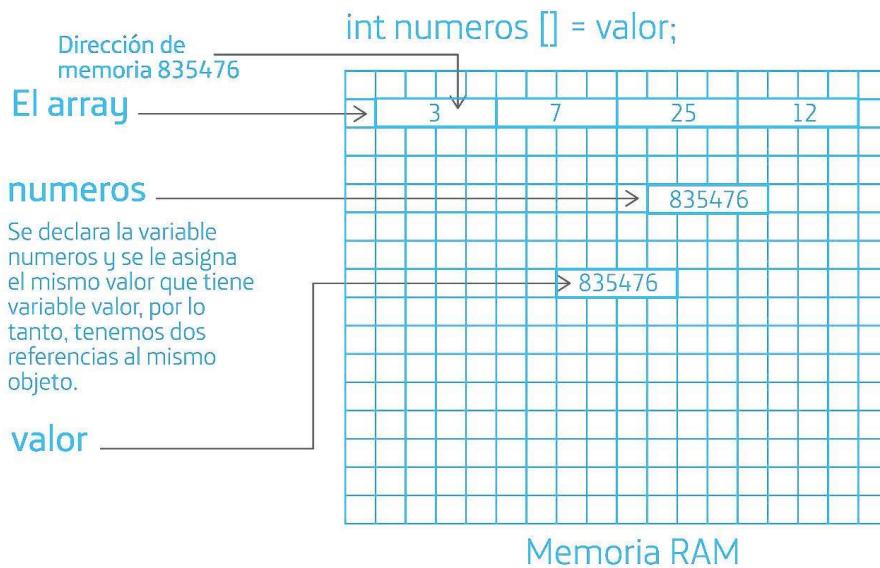
Paso 1: declaración de la variable *valor* como una futura referencia a un *array* de números enteros.



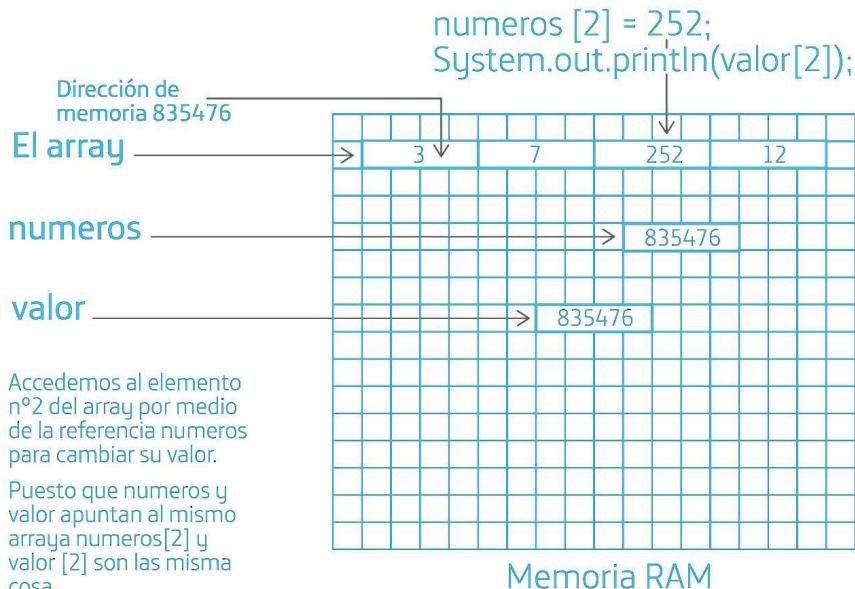
Paso 2: construcción del nuevo *array* con el operador *new*. La variable *valor* apunta al nuevo *array*.



Paso 3: asignación de valores a los elementos del *array*.



Paso 4: declaración de una nueva referencia a un *array* que igualamos con la variable *valor*. Las variables *numeros* y *valor* son referencias que apuntan al mismo objeto (un *array*).



Paso 5: accedemos al elemento 2 del *array* por medio de la referencia *numeros*. Como *valor* y *numeros* son referencias al mismo objeto (al mismo *array* en este caso), resulta que *numeros[2]* y *valor[2]* acceden exactamente al mismo elemento.

Si escribes en pantalla el valor de las variables *valor* y *numeros*, como son referencias de memoria, escribirá la dirección de memoria donde se encuentra el objeto al que apuntan:

```
System.out.println(valor);
System.out.println(numeros);
```

Estas dos líneas muestran la misma dirección de memoria porque apuntan al mismo objeto.

El resultado mostrado en pantalla será similar a este, aunque la dirección de memoria puede variar.

[I@2a139a55

[I@2a139a55

La *I* que aparece delante del símbolo @ significa que se trata de un *array* de números enteros, después tenemos un número en hexadecimal que representa la dirección de memoria donde se encuentra el *array* al que apuntan.

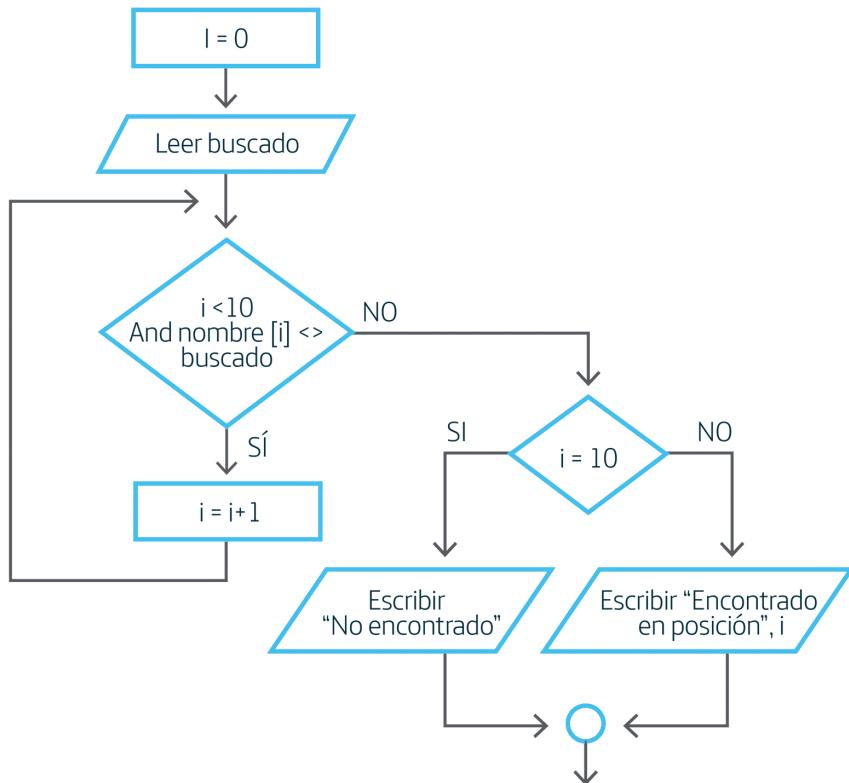
Búsqueda en arrays

En este apartado aprenderás dos técnicas de búsqueda en un vector: la **búsqueda secuencial** y la **búsqueda binaria**.

Algoritmo de búsqueda secuencial

Se trata de buscar un elemento determinado en el vector recorriendo todos los elementos desde la posición 0 hasta que lo encontramos, o bien hasta que lleguemos al final del vector sin encontrarlo.

Sirve igualmente para buscar en vectores ordenados que desordenados.



Búsqueda secuencial en vector *nombre*.

Programa Java de búsqueda secuencial

Aquí tienes un programa Java completo de búsqueda secuencial, incluyendo la carga de datos.

```
public class Principal {  
    public static void main(String[] args) {  
        // Declaración del vector.  
        String nombre[] = new String[10];  
  
        // Inicialización de los valores del vector.  
        nombre[0] = "Pepe";  
        nombre[1] = "Juan";  
        nombre[2] = "Luis";  
        nombre[3] = "Alicia";  
        nombre[4] = "Rosa";  
        nombre[5] = "Carlos";  
        nombre[6] = "Pedro";  
        nombre[7] = "Lucas";  
        nombre[8] = "María";  
        nombre[9] = "Carlota";  
  
        // Algoritmo de búsqueda para localizar a Alicia.  
        String buscado = "Alicia";  
        int i=0;  
        while (i<10 && !nombre[i].equals(buscado)) {  
            i=i+1;  
        }  
  
        if (i==10)  
            System.out.println("No encontrado");  
        else  
            System.out.println("Encontrado en posición "+i);  
    }  
}
```

Búsqueda secuencial en Java.

La expresión "`nombre[i].equals(buscado)`" compara la cadena situada en la posición *i* del array con el contenido de la variable *buscado*. Si son iguales retorna *true* y si son distintas retorna *false*.

Algoritmo de búsqueda binaria o dicotómica

Solo puede utilizarse en **vectores ordenados**. Consiste en determinar el elemento central del vector y utilizarlo para compararlo con el elemento buscado. Si el elemento buscado es menor que el central, sabremos que el elemento buscado está en la mitad inferior. Si el elemento buscado es mayor, sabremos que está en la mitad superior. Si el elemento buscado es igual al central habremos terminado nuestra búsqueda.

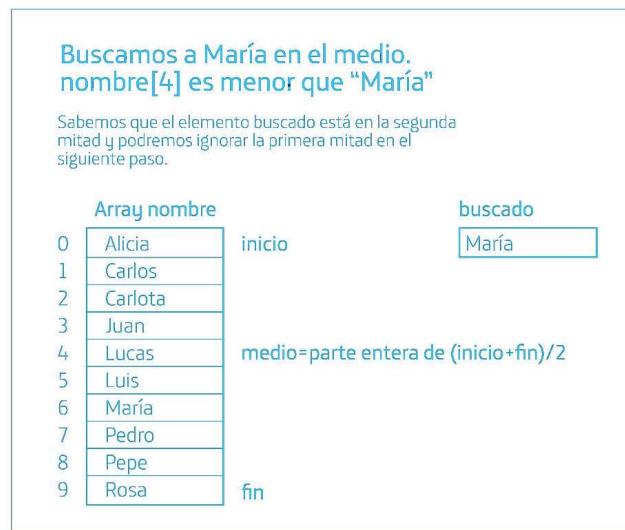
Por lo explicado anteriormente llegamos a la conclusión de que hay que dividir el *array* en dos *subarrays* más pequeños, acotando así la búsqueda a la mitad del *array*. Después volveremos a dividir el *subarray* en otros dos y así sucesivamente hasta completar la búsqueda. Este procedimiento resulta mucho más rápido que la búsqueda secuencial, pero no hay que olvidar que es imprescindible que el *array* esté ordenado.

```
public class Principal {  
    public static void main(String[] args) {  
        // Declaración del vector.  
        String nombre[] = new String[10];  
  
        // Inicialización de los valores del vector en orden alfabé  
        nombre[0] = "Alicia";  
        nombre[1] = "Carlos";  
        nombre[2] = "Carlota";  
        nombre[3] = "Juan";  
        nombre[4] = "Lucas";  
        nombre[5] = "Luis";  
        nombre[6] = "María";  
        nombre[7] = "Pedro";  
        nombre[8] = "Pepe";  
        nombre[9] = "Rosa";  
  
        // Búsqueda binaria  
        String buscado = "María";  
  
        int inicio = 0;  
        int fin = 9;  
        int posicion = -1;  
  
        while (posicion== -1 && inicio <= fin) {  
            int medio = (int)((inicio+fin)/2);  
            if (nombre[medio].equals(buscado)) {  
                posicion = medio;  
            }  
            else {  
                if (buscado.compareTo(nombre[medio])<0)  
                    fin=medio-1;  
                else  
                    inicio = medio+1;  
            }  
        }  
  
        if (posicion > -1)  
            System.out.println("Encontrado en posición "+posici  
        else  
            System.out.println("No encontrado");  
  
    }  
}
```

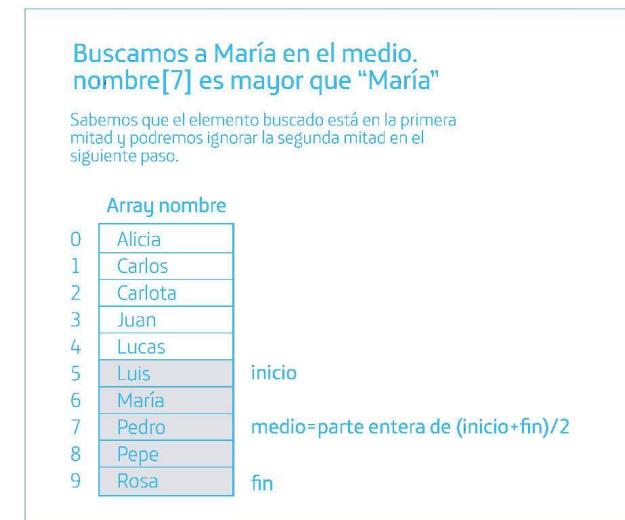
Algoritmo de búsqueda binaria.

La expresión "`buscado.compareTo(nombre[medio])`" compara el contenido de la variable `buscado` con la posición del `array` determinada por el valor de la variable `medio`. Si las dos cadenas son iguales, retorna 0. Si la primera cadena es menor que la segunda retorna un número negativo y si la primera cadena es mayor que la segunda retorna un número positivo.

Observando la siguiente galería de imágenes podrás entender mejor el algoritmo anterior:



Paso 1



Paso 2

Buscamos a María en el medio.
nombre[5] es menor que "María"

Sabemos que el elemento buscado está en la segunda mitad y podremos ignorar la primera mitad en el siguiente paso.

Array nombre	
0	Alicia
1	Carlos
2	Carlota
3	Juan
4	Lucas
5	Luis
6	María
7	Pedro
8	Pepe
9	Rosa

inicio y medio
fin

Paso 3

Buscamos a María en el medio.
nombre[6] es igual a "María"

Hemos encontrado el elemento buscado.

Array nombre	
0	Alicia
1	Carlos
2	Carlota
3	Juan
4	Lucas
5	Luis
6	María
7	Pedro
8	Pepe
9	Rosa

inicio, fin y medio

Paso 4

Ordenación de arrays

En este apartado conocerás el algoritmo para ordenar un *array* por el **método de la burbuja**.

El **método de la burbuja** consiste en ir comparando cada elemento del *array* con el siguiente e intercambiarlos si no están ordenados de la manera correcta. Este proceso tendrá que repetirse varias veces hasta que el *array* esté completamente ordenado.

Método de la burbuja

La [Wikipedia](#) explica de una manera gráfica el funcionamiento del método de la burbuja.

```
public class Principal {  
    public static void main(String[] args) {  
  
        // Creamos un array desordenado.  
        String nombre[] = new String[5];  
        nombre[0] = "Lucas";  
        nombre[1] = "María";  
        nombre[2] = "Alicia";  
        nombre[3] = "Rosa";  
        nombre[4] = "Carlos";  
  
        // Ordenamos por el método de la burbuja  
        for (int i = 0; i <= 4; i++) {  
            for (int j = 0; j <= 3; j++) {  
                if (nombre[j].compareTo(nombre[j + 1]) > 0) {  
                    String aux = nombre[j];  
                    nombre[j] = nombre[j + 1];  
                    nombre[j + 1] = aux;  
                }  
            }  
        }  
  
        // Muestra en pantalla los 5 nombres ya ordenados  
        for (int i = 0; i <= 4; i++) {  
            System.out.println(nombre[i]);  
        }  
    }  
}
```

Ordenación por el método de la burbuja.

Resumen

Has terminado la lección, vamos a ver los puntos más importantes que hemos tratado.

- Un *array* es un conjunto de elementos organizados en memoria en posiciones contiguas. Todos los elementos son del mismo tipo y se puede acceder a ellos a partir de subíndices.



PROEDUCA