Capítulo 7. Arrays

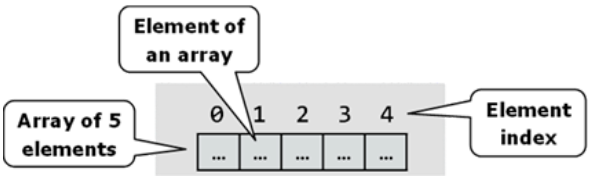
A. J. Pérez

Arrays

Un array es una colección de variables del mismo tipo...

Con más precisión:
Un array es un tipo de dato estructurado, lineal y homogéneo de naturaleza estática que agrupa a varios elementos contiguos del mismo tipo, en un sólo bloque de memoria.

Los elementos de un array en Java están numerados de `0, 1, 2, ... N-1`. Este número que localiza cada elemento se denomina *índice*. El número total de elementos de un array es su *longitud* o *tamaño*.



Hay que insistir que todos los elementos de un array son del mismo tipo (*homogeneidad*); sean de tipos *primitivos* o bien, tipos de *objetos*. **Un array es una colección de elementos homogéneos dispuestos en una secuencia contigua**.

Los arrays pueden organizarse en diferente número de dimensiones, los más utilizados son los unidimensionales conocidos como *vectores* y los bidimensionales conocidos como *tablas* o *matrices*.

En la utilización de arrays hay que distinguir los siguientes aspectos:

1. Declaración
2. Dimensionado y creación
3. Inicialización
4. Acceso a los datos

Declaración

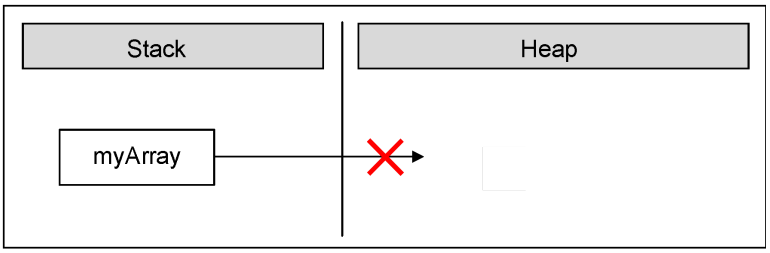
En Java los arrays tienen una longitud fija que se especifica en la inicialización, y determina el número total de elementos. Una vez que se ha definido un array de una determinada organización de índices y tamaño, no se puede cambiar.

En el siguiente ejemplo se muestra **la declaración de un vector** llamado `myArray` del tipo *array de* ***`int`***. En la sintaxis de Java se **utiliza un par de corchetes vacíos**:

```
int[] myArray;
```

La variable `myArray` es el nombre de un vector de tipo **`int[]`**, los corchetes indican que serán múltiples los elementos enteros almacenados.

El nombre `myArray` **es una referencia a un objeto array** que inicialmente apunta a **`null`**; no tiene memoria asignada para los elementos del vector.



Dimensionado y creación

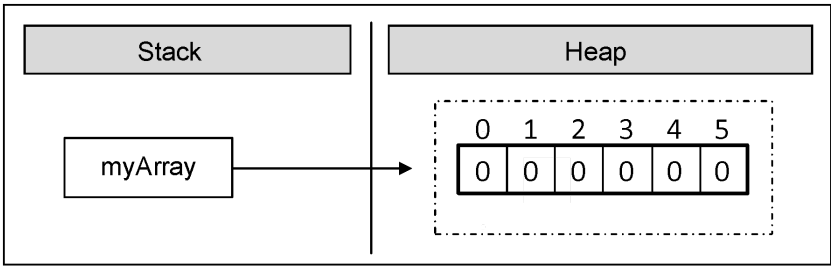
En el siguiente ejemplo se muestra cómo un array llamada `myArray`, de tipo entero, es declarado y creado. **En el proceso de creación** con el operador **`new`**; **se obtiene memoria** para los 6 elementos que lo constituyen.

```
int[] myArray = new int[6];      // Reserva bloque de 6x4 bytes en memoria.
```

Se admite también la sintaxis típica de C.

```
int myArray[];
myArray = new int[6];
```

En cualquier caso, se debe utilizar un operador especial, **`new`**, para asignar el espacio en la *memoria dinámica* (*heap*) del programa. **Este proceso de asignación de espacio, necesita información sobre el número de elementos que se agruparán en cada dimensión o índice de ese array; además de la naturaleza o tipo de cada elemento.**



El espacio de memoria reservado en el ejemplo queda asignado, pero todos los elementos están vacíos o con el valor por defecto del tipo base (**`int`**) utilizado –en este caso `0`-

El proceso de reserva de espacio y dimensionado de índices necesita información sobre el número de elementos de cada dimensión que se agruparán en ese array, y la naturaleza o tipo de cada elemento.

Tamaño de un *array*

Es importante destacar que los arrays de Java son objetos y como tales tienen una propiedad llamada `length` que indica el número de elementos que contiene en cada dimensión o índice.

Para el ejemplo visto anteriormente:

```
      System.out.println(myArray.length);   // Muestra 6
```

Inicialización de elementos con valores predeterminados

Antes de utilizar un array, probablemente se requiera que sus elementos tengan un valor inicial. En algunos lenguajes de programación no se establecen los valores iniciales por defecto, y luego, cuando se intenta acceder a un elemento, se produce algún error. **En Java, todas las variables, incluyendo los elementos de un array tienen un valor inicial predeterminado**.

Según el tipo base declarado en el array, los valores predeterminados son:

- **`byte`**, **`short`**, **`int`** y **`long`**: 0
- **`float`** y **`double`**: 0.0
- **`char`**: '\u0000'
- **`boolean`**: **`false`**
- **Referencias a objetos**: **`null`**

Los valores iniciales se pueden configurar explícitamente en tiempo de compilación y se puede hacer de diferentes maneras. Una de las alternativas es utilizar una expresión literal con los elementos del array:

```
int[] myArray = {1, 2, 3, 4, 5, 6};   // Crea un bloque de 6x4 bytes en memoria
                                      // y asigna valor a cada elemento.
```

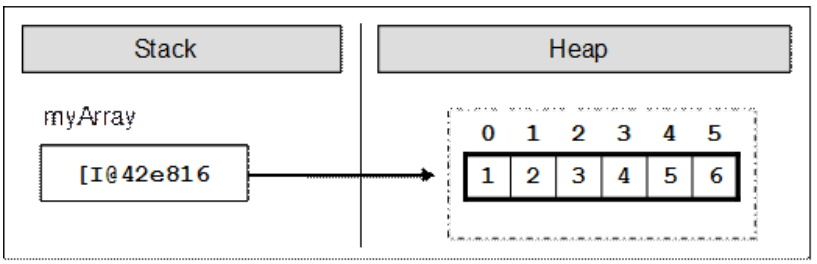
En este caso: declara , crea, dimensiona e inicializa el array simultáneamente.

Alternativamente se puede utilizar la sintaxis:

```
int[] myArray;                        // Declaración
myArray = new int[] {1, 2, 3, 4, 5, 6};

// La sintaxis siguiente  no funcionará...
myArray = {1, 2, 3, 4, 5, 6};
```

En este caso, declara previamente y después crea, dimensiona e inicializa el array.

Así es como se vería en la memoria, en los dos casos:



La sintaxis de llaves hace las veces del operador **new**, y entre ellas se listan los valores iniciales del vector, separados por comas. Los grupos de llaves y el número de valores determinan los rangos de los índices y así el total de elementos necesarios.

Otra posibilidad para la inicialización, en este caso, en tiempo de ejecución es la utilización de una estructura de control repetitiva o bucle:

```
int[] myArray = new int[6];
for (int i=0; i < myArray.length; i++) {
      myArray[i] = i+1;             // Asigna valor a cada elemento.
}
```

En este procedimiento de inicialización es útil cuando el valor de los elementos no es conocido a priori o bien son muchos elementos.

Es interesante la posibilidad, disponible en Java, de crear un array de tamaño proporcionado en tiempo de ejecución:

```
// Por teclado.
int numeroElementos = new Scanner(System.in).nextInt();
int[] myArray = new int[numeroElementos];
for (int i=0; i < myArray.length; i++) {
      myArray[i] = i+1;             // Asigna valor a cada elemento.
}
```

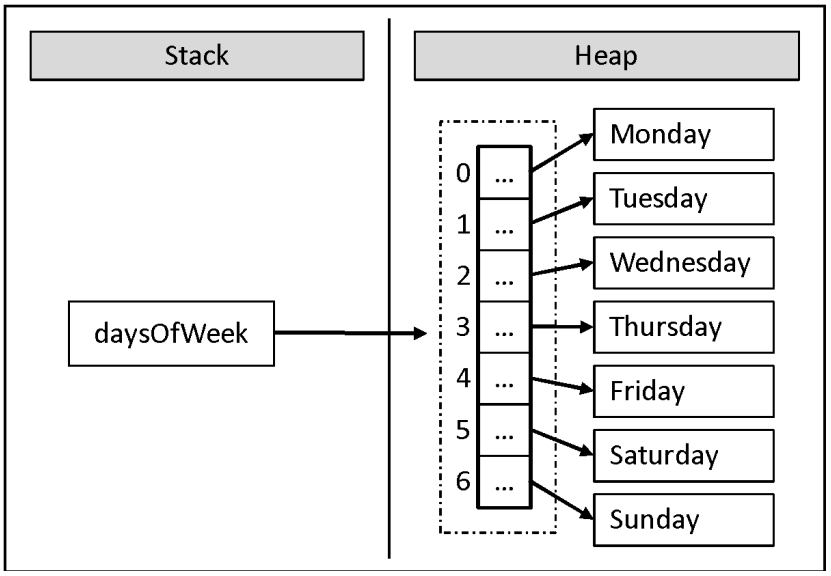
En este ejemplo el número de elementos del vector es proporcionado interactivamente durante la ejecución del programa.

Ejemplo: Vector de String

```
String[] daysOfWeek = { "Monday", "Tuesday", "Wednesday","Thursday",
                        "Friday", "Saturday", "Sunday" };
```

En este caso, el array (vector) se dimensiona con los 7 elementos de tipo `String`. Los `String`

son un tipo de referencia (objeto) y sus valores se almacenan en la memoria dinámica. Así es como el vector quedaría en la memoria:



En la *pila (Stack)* queda almacenada la variable `DaysOfWeek` , que apunta a la zona de memoria dinámica del *montículo (Heap)* que contiene los elementos del array. Cada uno de estos 7 elementos es un objeto de tipo `String`, que a su vez apunta a otra zona de la memoria donde se almacena el texto.

Acceso a los elementos de un array

El acceso a los elementos de un array es directo con un índice. Cada elemento se puede acceder con el nombre del array y su correspondiente índice colocado entre corchetes. En el acceso a un elemento de un array, éste actúa igual como si fuese una variable normal del tipo correspondiente; esto es válido tanto para la lectura como para la escritura.

Los elementos de un arrays también pueden ser accedidos utilizando una variable :

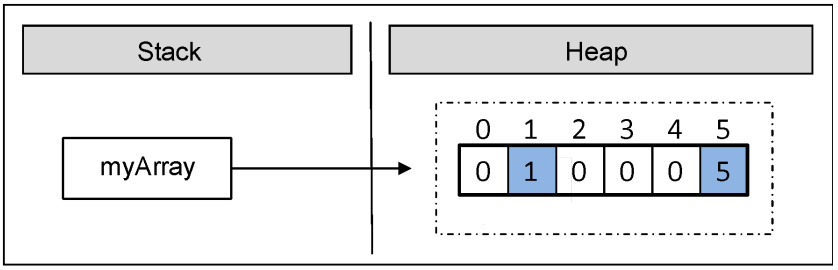
```
int[] myArray = {1, 2, 3, 4, 5, 6};
myArray[indice] = 100;
```

Se asigna un valor `100` al elemento que se encuentra en la posición `indice`, donde el valor de `indice` debe ser válido para el tamaño declarado del array.

Otro ejemplo en el que se asigna una serie de números y luego se cambian algunos de sus elementos:

```
int[] myArray = new int[6];
myArray[1] = 1;
myArray[5] = 5;
```

Después del cambio de los elementos, la memoria quedaría:



Los límites de un *array*

Los arrays en Java son por defecto de base cero, lo que significa que la enumeración de los elementos empieza desde `0`. El primer elemento tiene el índice `0`, el segundo `1`, etc. En un conjunto de `N` elementos, el último elemento tiene el índice de `N-1`.

Error en el rango de un *array*

Durante la ejecución del programa que accede a los elementos de un array, la máquina virtual Java, realiza la comprobación de rango válido y no permite ir más allá de sus límites y dimensiones. En cada acceso a los elemento del array se realiza una comprobación de si el índice es válido o no. Si no, lanza una excepción de tipo:

```
java.lang.ArrayIndexOutOfBoundsException
```

Esta comprobación tiene un coste computacional y penaliza ligeramente el rendimiento.

Un ejemplo en el que se intenta acceder a un elemento que se encuentra fuera de rango válido:

```
public class EjemploIndexOutOfRange {
	static void main(String[] args)  {

		int[] myArray = { 1, 2, 3, 4, 5, 6 };
		System.out.println(myArray[6]);
	}
}
```

En el ejemplo se crea un array que contiene `6` enteros. El primer elemento es el índice `0`, al último le corresponde un índice de `5`. Se intenta acceder al elemento de índice `6`, pero no lo existe, lo que lleva al lanzamiento de la excepción:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException : 6
	in EjemploIndexOutOfRange.main (EjemploIndexOutOfRange.java: 5)
```

Ejemplo: Inversión de un *vector*

En el siguiente ejemplo se puede ver cómo cambiar los elementos de un vector, accesibles por índice. Se utiliza un vector auxiliar en el que guardar los elementos del primero en orden inverso.

```
import java.util.Arrays;

public class VectorInverso {
             public static void main(String[] args) {
           int[] vector = { 1, 2, 3, 4, 5 };

           // Obtiene el tamaño del array.
           int longitud = vector.length;

           // Declara y crea un array para el resultado.
           int[] inverso = new int[longitud];

           // Inicializa el array inverso.
           for (int i = 0; i < longitud; indice++) {
                  inverso[longitud - i - 1] = vector[i];
           }

           // Muestra el array inverso.
           System.out.println(Arrays.toString(inverso));
      }
}
```

- En el ejemplo se crea un vector de tipo **`int`**, y lo inicializa con los números del 1 al 5.
- A continuación, se obtiene la longitud del array y se guarda en la variable entera longitud. Hay que tener en cuenta que utilizando el atributo **`length`**, que contiene el número de elementos del array. En Java cada array conoce su tamaño.
- Seguidamente, se declara un vector auxiliar llamado inverso de tamaño `longitud`, para guardar los elementos del primero, pero en orden inverso.
- Para realizar la inversión de los elementos se usa un bucle **for**, que en cada iteración el incrementa el contador `i` que proporciona acceso a cada elemento del vector . El bucle deja de iterar al llegar al final de vector.
- En la primera iteración del bucle, `i` es 0. Con `vector[i]` se obtiene acceso al primer elemento del array , pero en `inverso[longitud - i - 1]` se accede al último elemento.
- En el último elemento del array `inverso` se guarda el primer elemento del array `vector`. En cada iteración posterior, `i` se incrementa en uno, la posición en el array `vector` se incrementa en uno, pero en inverso se reduce en uno. El resultado es el vector original en orden inverso.
- Para mostrar el vector inverso se ha utilizado la clase `java.util.Arrays` y su método `toString()`, al que se le proporciona el vector inverso.

Entrada de una serie de datos desde teclado

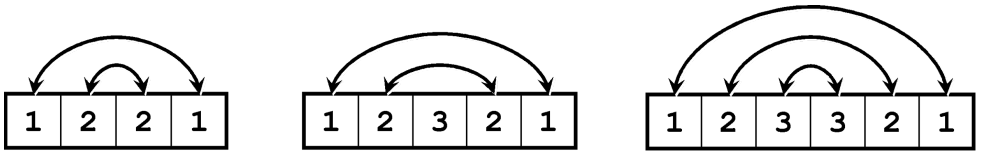
Se puede leer los valores de un array desde el teclado. Inicialmente hay que asignar memoria para el array; se puede pedir un número n por consola y utilizarlo para establecer el tamaño:

```
Scanner teclado = new Scanner(System.in);

int n = teclado.nextInt();
int[] vector = new int[n];

for (int i = 0; i < n; i++) {
  vector[i] = teclado.nextInt();
}
```

El bucle **for** permite recorrer el array. En cada iteración se asigna el valor leído desde el teclado a cada elemento del vector.

Ejemplo: Comprobación de la simetría de un vector

Una vector es simétrico, si los elementos primero y último son el mismo y el segundo y el penúltimo elemento también son los mismos, etc. Se muestran algunos ejemplos de vectores simétricos:




```
Scanner teclado = new Scanner(System.in);

int n = teclado.nextInt();
int[] vector = new int[n];
boolean simetrico = true;

for (int i = 0; i < n; i++) {
	vector[i] = teclado.nextInt();
}
for (int i = 0; i < (vector.length + 1) / 2; i++) {
	if (vector[i] != vector[n - i - 1]) {
		simetrico = false;
		break;
	}
}
System.out.println("¿Simétrico? " + simetrico);
```

- Se ha creado un vector y se han leído sus valores desde la consola. Para ver si el vector es simétrico hay que iterar sólo la mitad de elementos comprobando si es igual a `(vector.length + 1) / 2` , ya que no se sabe si el vector tiene una longitud par o impar.
- Para determinar si un vector es simétrico se usa una variable **`boolean`** asumiendo que el vector es simétrico.
- Se recorre el vector y se comparan los elementos por parejas, el primero con el último elemento de la segunda parte, etc. Si una iteración verifica que los valores de los elementos no coinciden la variable booleana se establece en **`false`**, es decir, el vector no es simétrico.
- Si no es simétrico se interrumpe el bucle con **`break`**.
- Finalmente se muestra el resultado de la variable booleana en la consola.

Salida de una serie de datos por pantalla

A menudo es necesario mostrar una serie de datos de un array por la consola, ya sea para pruebas o para otros fines.

La salida por pantalla de un array se resuelve de manera similar a la inicialización de los elementos, es decir, utilizando un bucle para recorrer y procesar los elementos del array. No hay reglas fijas para extraer los datos. Por supuesto, una buena práctica consiste en tener el formato correcto.

Un error común es tratar de enviar un array directamente a la consola de la siguiente manera:

```
String[] vector = { "uno" , "dos" , "tres" , "cuatro" };
System.out.println(vector);
```

Este código, no muestra el contenido del `vector` sino su dirección en la pila de memoria del programa (porque los arrays son referencias). Así el resultado de la ejecución del código anterior muestra algo parecido a:

```
java.lang.String;@42e816
```

Para conseguir mostrar arrays de forma correcta en la consola se puede usar el bucle:

```
String[] vector = { "uno" , "dos" , "tres" , "cuatro" };

// Procesa todos los elementos del vector.
for ( int i = 0; i < vector.length; i++) {

      // Imprime cada elemento en una línea separada.
```

```
      System.out.println("elemento[" + i + "] = " + vector[i]);
}
```

Cada iteración del bucle **for** sobre el vector ejecuta el método `System.out.printf()` que visualizar los datos en la pantalla en un formato determinado. El resultado es el siguiente:

```
elemento[0] = uno
elemento[1] = dos
elemento[2] = tres
elemento[3] = cuatro
```

Hay otra manera más fácil de mostrar el contenido de un array:

```
String[] vector = { "uno" , "dos" , "tres" , "cuatro" };
System.out. println(java.util.Arrays.toString(vector));
```

El resultado es una cadena formateada que contiene todos los elementos del array separados por comas:

```
[Uno, dos, tres, cuatro]
```

Iteración sobre los elementos de un array

Como se ha visto, la iteración en los elementos de un array es una de las operaciones básicas para su procesamiento. La iteración sobre un array puede tener acceso a cada elemento utilizando el índice. Este proceso se puede hacer con cualquier tipo de bucles, pero el más adecuado para ello es el bucle **for**.

Iteración con un bucle

Una buena práctica es usar bucles cuando se trabaja con arrays o cualquier estructura indizada. He aquí un ejemplo que duplica el valor de cada uno de los elemento de un array de números:

```
int[] vector = {1, 2, 3, 4, 5};

for (int i = 0; i < vector.length; i++) {
      vector[i] = 2 * vector[i];
}

System.out.println(Arrays.toString(vector));
```

Salida:

```
[2, 4, 6, 8, 10]
```

Con un bucle se puede manejar el contador `i` asociado al índice actual del array y acceder con precisión a cualquier otro elemento que se necesite.

Se puede recorrer parte de un array en lugar de hacerlo en su totalidad para algún fin concreto. Aquí está un ejemplo:

```
for (int i = 0; i < vector.length; i += 2) {
  vector[i] = vector[i] * vector[i];
}
```

En el ejemplo se recorren todos los elementos del array, que se encuentran en las posiciones pares y asigna en esa posición el cuadrado de su valor.

A veces es útil recorren un array hacia atrás. Se puede hacer esto de una manera análoga, excepto que el bucle debe empezar con un índice inicial igual a la posición del último elemento del array, y se irá decrementando en cada iteración. Un ejemplo sería:

```
int[] vector = {1, 2, 3, 4, 5};
System.out.print("Inverso: ");

for (int i = vector.length - 1; i >= 0; i--) {
  System.out.print(" " + vector[i] + " ");
}
```

Salida:

```
Inverso: 5 4 3 2 1
```

En el ejemplo anterior el vector se recorre de detrás hacia delante y muestra cada elemento en la pantalla.

Iteración con bucle for-each

Una variante para el bucle de recorrido completo de arrays es el llamado **`for`**-each.

La construcción de este bucle es la siguiente:

```
for (<tipo> variable : array) {
      // Proceso a realizar para cada elemento.
}
```

En este bucle el `<tipo>` caracteriza a cada elemento del array que se recorre y la `variable` representa el elemento actual en cada iteración.

El bucle **for**-each se caracteriza por el hecho de que se recorre completamente el array. No es posible recorrerlo de forma parcial, ni utilizar el índice; que permanece oculto. Este bucle se utiliza cuando se necesita un tratamiento completo de todos los elementos del array. Sólo es posible leer el elemento del array, que queda accesible en la `variable`. Es más rápido que el bucle normal para el mismo fin.

Ejemplo: bucle for-each

En el siguiente ejemplo se ve cómo utilizar **for**-each para iterar sobre arrays:

```
String[] capitales = {"Madrid", "Lisboa", "Londres", "Paris"};

for (String capital : capitales) {

      System.out.println(capital);
}
```

Una vez declarado un array de `String` `capitales` , el bucle **for**-each recorre el array y muestra los elementos por la consola. El elemento actual de cada iteración se almacena en la variable de `capital`. La salida sería:

```
Madrid
Lisboa
Londres
Paris
```

Los arrays multidimensionales

Hasta ahora se han tratado los arrays unidimensionales o vectores. Los arrays se pueden utilizan con más de una dimensión. Por ejemplo, un tablero de ajedrez estándar se puede representar como una *tabla* o *matriz* de dos dimensiones de tamaño `8x8` (8 posiciones en la dirección horizontal y 8 en una dirección vertical).

¿Qué es un array multidimensional?

Cualquier tipo de dato Java puede ser utilizado como base de los elementos de un array. Los arrays en sí, también pueden ser considerados como un tipo para otro array. Así podemos tener un array de arrays.

Un array bidimensional de enteros se declara **`int[][]`**. Se puede suponer que es un vector de vectores de enteros:

```
int[][] ArrayDosDimensiones;
```

Tales arrays se les llaman *tablas* o *matrices*.

Del mismo modo, se pueden declarar arrays de tres dimensiones:

```
int[][][] ArrayTresDimensiones;
```

En teoría no hay límite en el número de dimensiones de un array, pero en la práctica rara vez se utilizan arrays con más de dos dimensiones, por lo que serán tratadas con más detalle.

Declaración y dimensionado de arrays multidimensionales

Los arrays multidimensionales se declaran de una manera análoga a la de una sola dimensión. Cada dimensión se indicada entre corchetes:

```
int[][] matrizEnteros;
double[][] matrizReales;
String[][][] cuboCadenas;
```

➢ El ejemplo anterior muestra cómo crear arrays bidimensionales y tridimensionales. Cada dimensión corresponde a un par de corchetes `[]`.

La asignación de memoria para el tamaño multidimensional se resuelve usando la palabra clave **new** y cada dimensión entre paréntesis establecer el tamaño que se necesita:

```
int[][] matrizEnteros = New int[3][4];
double[][] matrizReales = New double[8][2];
String[][][] cuboCadenas = New String[5][5][5];
```

➢ En el ejemplo anterior `matrizEnteros` es un array de dos dimensiones con 3 elementos de tipo **`int[]`** y cada uno de estos tres elementos es de tamaño 4.

Tal como se representan los arrays bidimensionales parece difícil de entender. Es posible que resulte más fácil verlas como matrices o tablas que tienen filas y columnas:

	0	1	2	3
0	1	3	6	2
1	8	5	9	1
2	4	7	3	0

Las filas y las columnas se numeran con índices de 0 hasta el tamaño de la dimensión menos uno. Un array de dos dimensiones tiene `m*n` elementos.

Acceso a los elementos de un array multidimensional

Las matrices tienen dos dimensiones y por lo tanto cada uno de sus elementos se accede mediante dos índices; uno para las filas y otro para las columnas. Los arrays multidimensionales tienen un índice diferente para cada dimensión.

En el siguiente ejemplo:

```
int [][] matriz = {
                    {1, 2, 3, 4},
                    {5, 6, 7, 8},
                  };
```

El array `matriz` tiene 8 elementos en 2 filas y 4 columnas. Cada elemento puede ser accedido de la siguiente manera:

```
matriz[0][0]   matriz[0][1]   matriz[0][2]   matriz[0][3]
matriz[1][0]   matriz[1][1]   matriz[1][2]   matriz[1][3]
```

En arrays multidimensionales, cada elemento se identifica con tantos índices como dimensiones tenga el array:

`arrayDimensionN [indice1] ... [indiceN]`

Inicialización de arrays multidimensionales

La inicialización de arrays multidimensionales es análoga a la inicialización de una sola dimensión. Los valores de los elementos pueden aparecer inmediatamente después de la declaración:

```
int [][] matriz = {
                    {1, 2, 3, 4}, // valores de fila [0]
                    {5, 6, 7, 8}, // valores de fila [1]
                  };
// El tamaño de la matriz es 2 x 4 (2 filas, 4 colunas)
```

- En el ejemplo anterior se inicializa un array bidimensional de enteros con 2 filas y 4 columnas.
- Las llaves externas contienen los elementos de la primera dimensión, es decir, filas de la matriz de dos dimensiones.
- Cada línea representa una matriz unidimensional que se inicializa en la forma habitual.

Se puede dimensionar un array y posteriormente hacer un tratamiento que recorre la estructura y asigna arbitrariamente valores, como en este ejemplo:

```
int [][] matriz = new int[2][3]; // El tamaño de la matriz es 2x3.

// Primera fila (se podría hacer con un bucle).
matriz[0][0] = 1;
matriz[0][1] = 2;
matriz[0][2] = 3;

// Segunda fila.
matriz[1][0] = 4;
matriz[1][1] = 5;
matriz[1][2] = 6;
```

Otro caso más complejo se da cuando las filas de la tabla son de distinta longitud y se necesita dimensionar el número de elementos, dejando los valores sin asignar para hacerlo posteriormente:

```
int [][] tabla = new int[3][];  // El tamaño de tabla es 3 filas indefinidas.

tabla[0] = new int[4];          // La primera fila es de 4 elementos.
tabla[1] = new int[2];          // La segunda fila es de 2 elementos.
tabla[2] = new int[3];          // La tercera fila es de 3 elementos.

// Primera fila (se podría hacer con un bucle).
tabla[0][0] = 1;
tabla[0][1] = 2;
tabla[0][2] = 3;
tabla[0][3] = 4;

// Segunda fila.
tabla[1][0] = 5;
tabla[1][1] = 6;

// Tercera fila.
tabla[2][0] = 7;
tabla[2][1] = 8;
tabla[2][2] = 9;
```

Arrays de dos dimensiones y la memoria

En la memoria los arrays bidimensionales y multidimensionales almacenan sus valores en el montículo (Heap) con una referencia (puntero) a un área que contiene referencias a otros arrays.

Prácticamente todas las variables de tipo de array (unidimensional o multidimensional) son referencias a un lugar en la memoria dinámica que contiene los elementos del array.

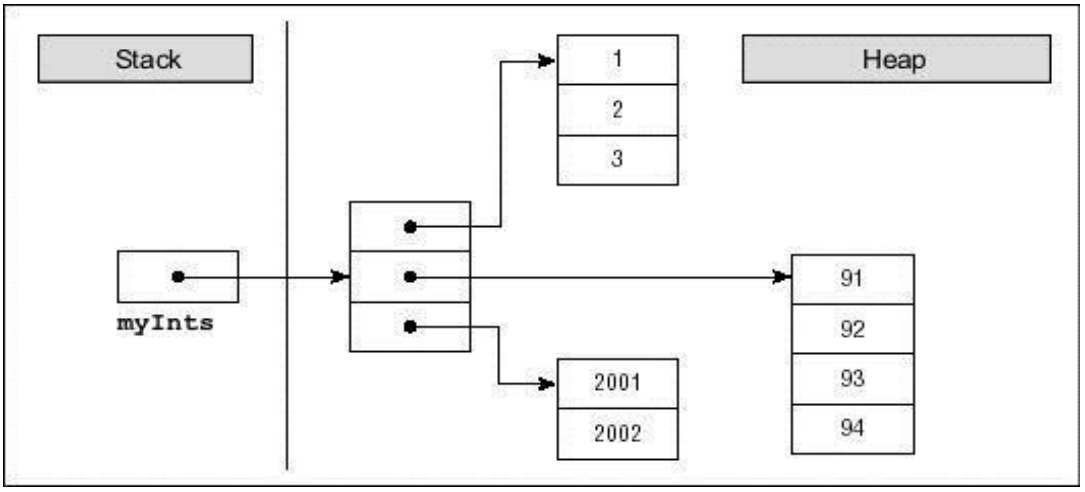
En un array de dos dimensiones, sus elementos son vectores, respectivamente, y mantienen referencias a la memoria dinámica donde se ubican sus respectivos elementos. Suponiendo el siguiente array:

```
int [][] myInts = { {1, 2, 3},
```

```
                    {91, 92, 93, 94},
                    {2001, 2002}
                  };
```

Es un array no estándar debido a que es una forma no rectangular. Se compone de 3 filas, cada una de las cuales tiene un número diferente de columnas. Esto es posible en Java.

Una vez inicializado,el array se representa de la siguiente manera en la memoria:



Longitud de un array multidimensional

Cada dimensión de un array multidimensional tiene su propia longitud, que está disponible durante la ejecución del programa. Considérese el siguiente ejemplo de una matriz de dos dimensiones:

```
int [][] matriz = {
                    {1, 2, 3, 4},
                    {5, 6, 7, 8},
                  };
```

Se puede obtener el número de filas de este array bidimensional con `matriz.length`. En la práctica, esto es la longitud del array unidimensional que contiene referencias a sus elementos (que también son arrays). La extracción de la longitud de la i-ésima fila se consigue con `matriz[i].length`.

En el siguiente ejemplo se muestra dos veces el contenido de un array no regular de enteros utilizando bucles for clásicos y **`for`**`-each`

```
int [][] datos = {
                    {1, 2, 3, 4},
                    {5, 6},
                    {7, 8, 9}
                  };
```

```
// Utiliza la propiedad length para obtener el número de filas.
for (int fila = 0; i < datos.length; fila++) {

	// Utiliza la propiedad length de cada filas.
	for (int columna; j < datos[fila].length; columna++) {
		System.out.println(datos[fila][columna]);
	}
}
System.out.println();


for (int[] f : datos) { // f es un vector que se asigna con cada fila de datos.

	for (int n : f) {  // n se asigna con cada elemento de f.
		System.out.println(n);
	}
}
System.out.println();
```

Ejemplo: submatriz máxima

En el siguiente ejemplo se va a resolver un problema interesante: Dada una matriz rectangular de números. Se puede encontrar el tamaño máximo de una submatriz de 2x2 y mostrarla por pantalla. Se entiende por submatriz máxima aquella en la que la suma de cuatro elementos tiene un valor máximo:

```
public class MaxSubmatriz2x2 {

	public static void main(String[] args) {

		// Declaración de la matriz.
		int[][] matriz = {
			{ 0, 2, 4, 0, 9, 5 },
			{ 7, 1, 3, 3, 2, 1 },
			{ 1, 3, 9, 8, 5, 6 },
			{ 4, 6, 7, 9, 1, 0 }
		};

		// Buscar la submatriz de suma máxima de tamaño de 2 x 2.
		int maxSuma = Integer.MIN_VALUE;
		int mejorFila = 0;
		int mejorColum = 0;
		for (int fila = 0; fila < matriz.length - 1; fila++) {
			for (int colum = 0; colum < matriz[0].length - 1; colum
			++){
				int suma = matriz[fila][colum] +
					matriz[fila][colum + 1] +
					matriz[fila + 1][colum]+
					matriz[fila + 1][colum + 1];
				if (suma > maxSuma) {
					maxSuma = suma;
```

```
                                mejorFila = fila;
                                mejorColum = colum;
                            }
                        }
                }

                // Muestra resultados.
                System.out.println("la submatriz máxima es: ");
                System.out.printf("  %d %d%n",
                            matriz[mejorFila][mejorColum],
                            matriz[mejorFila][mejorColum + 1]);

                System.out.printf("  %d %d%n",
                            matriz[mejorFila + 1][mejorColum],
                            matriz[mejorFila + 1][mejorColum + 1]);

                System.out.printf("la suma máxima es: %d%n", maxSuma);
        }
}
```

Salida:

```
La submatriz máxima es:
  9 8
  7 9
La suma máxima es: 33
```

- Se crea una matriz de dos dimensiones que consta de números enteros. Se declaran las variables auxiliares `maxSuma`, `mejorFila`, `mejorColum`, que se inicializan a los valores mínimos.

- En la variable `maxSuma` se mantiene la suma máxima, en `mejorFila` y `mejorColum` se mantiene la fila y la columna actuales de la submatriz actual de `2×2` correspondiente a los elementos que hacen la actual suma máxima `maxSuma`.

Para acceder a todos los elementos de la submatriz de 2x2 que hay que sumar se utiliza:

```
matriz[fila][colum]
matriz[fila][colum+1]
matriz[fila+1][colum]
matriz[fila+1][colum+1]
```

- En el ejemplo anterior, `fila` y `colum` son los índices correspondientes al primer elemento de la matriz de tamaño 2x2.
- En el recorrido de todos los elementos de la matriz principal, hay que tener en cuenta que cuando se intenta acceder a la `fila+1` y `colum+1` el índice iría más allá de los límites.
- Accediendo a los elementos contiguos de cada elemento actual de la submatriz 2x2 y sumarlos se comprueba si la suma actual es mayor que la cantidad más guardada en `maxSuma` actual.

Ejercicios

1. Escribe un programa simple que cree un array (vector) de 10 elementos de números enteros e inicialice cada elemento con el valor 7 en cada elemento. Los elementos del vector se pueden mostrar en la pantalla.
- Utiliza un vector `int[]` y un bucle **`for`**.
- Documenta el código fuente con comentarios aclaratorios adicionales.

2. Escribe un programa simple que cree un array (vector) de 20 elementos de números enteros e inicialice cada elemento con un valor igual al índice correspondiente del elemento, multiplicado por 5. Los elementos del vector se pueden mostrar en la pantalla.
- Utiliza un vector `int[]` y un bucle **`for`**.
- Documenta el código fuente con comentarios aclaratorios adicionales.

3. Escribe un método que se llame `frecuenciaNum()` que recibe un vector de enteros y un número. Devuelve la frecuencia entre los elementos del vector del número recibido.
- Prueba el método pedido desde `main()`.
- Documenta el código fuente con comentarios aclaratorios adicionales.

4. Escribe un método que se llame `vectoresIntIguales()` que recibe dos vectores de enteros. Devuelve **`true`** si son iguales.
- Dos vectores son iguales si tienen la misma longitud y los valores, elemento a elemento, son iguales. La segunda condición se puede comprobar con un bucle.
- Documenta el código fuente con comentarios aclaratorios adicionales.

5. Escribe un método que se llame `comparaVectorChar()` que recibe dos vectores de caracteres. Devuelve `0` si son iguales, `1` si el primero es mayor y `-1` si el primero es menos.
- Prueba el método pedido desde `main()`.
- El orden alfabético de vectores de caracteres requiere la comparación uno a uno de sus caracteres, comenzando desde el extremo izquierdo. El orden alfabético corresponde al orden numérico creciente del código del carácter.
- Documenta el código fuente con comentarios aclaratorios adicionales.

6. Escribe un método que se llame `maximoIntConsecutivos()` que recibe un vector de enteros. Devuelve cuantos valores son consecutivos entre sus elementos. Si devuelve `1` todos los elementos son diferentes.

Por ejemplo: `{3, 2, 3, 4 , 2, 2, 4}` devolvería `3`

- Prueba el método pedido desde `main()`.
- Recorriendo los datos de izquierda a derecha, empezando por el segundo elemento hay que comprobar si es secuencia del anterior. Se puede utilizar un contador iniciado en `1` que se incrementa al comprobarse que un elemento es consecutivo del anterior. Cada vez que se inicia nueva secuencia hay que guardar el valor del contador en una variable auxiliar con el máximo alcanzado.
- Documenta el código fuente con comentarios aclaratorios adicionales.

7. Escribe un método que se llame `buscaInt()` que recibe un vector de enteros y un valor a buscar dentro del vector. Devuelve el índice de la primera ocurrencia dentro del vector proporcionado.

Por ejemplo: `{3,2,3,4 ,7,2,4}` si se busca 4 devolvería 3

- Prueba el método pedido desde `main()`.
- Hay que recorrer el vector secuencialmente y si se encuentra hay que terminar sin llegar al final del recorrido.
- Documenta el código fuente con comentarios aclaratorios adicionales.

8. Escribe un método que se llame `ordenadoInt()` que recibe un vector de enteros. Devuelve **`true`** si está ordenado.

Por ejemplo: `{3,2,3,4 ,7,2,4}` devolvería `false`

- Prueba el método pedido desde `main()`.
- Hay que recorrer el vector secuencialmente comprobando por parejas y si se encuentra un par de elementos desordenados hay que terminar sin llegar al final del recorrido.
- Documenta el código fuente con comentarios aclaratorios adicionales.

9. Escribe un método que se llame `masFrecuenteInt()`que recibe un vector de enteros. Devuelve el valor del elemento más frecuente, si hay coincidencia devolvería el primero de ellos.

Por ejemplo: `{4,1,1,4,2,3,4,4,1,2,4,9,3}` devolvería 4

- Prueba el método pedido desde `main()`.
- Una forma sería recorrer el vector anidar un bucle que averigüe la frecuencia del valor de cada elemento, utilizando un par de variables auxiliares para retener la frecuencia máxima y el valor asociado.
- Documenta el código fuente con comentarios aclaratorios adicionales.

10. Escribe un método que se llame `buscaSecuenciaInt()` que recibe dos vectores de enteros, el segundo es una secuencia de valores a buscar en el primer vector. Devuelve el índice donde empieza la primera ocurrencia de la secuencia buscada.

Por ejemplo: `{4, 3, 1, 4, 2, 5 , 8}` y `{4, 2, 5}` devolvería 3

- Prueba el método pedido desde `main()`.
- Hay que recorrer el vector secuencialmente comprobando coincidencia con el primer elemento de la secuencia buscada. Después hay que comprobarla completamente con otro bucle. Si hay coincidencia completa termina sin llegar al final del recorrido.
- Documenta el código fuente con comentarios aclaratorios adicionales.

11. Escribe cuatro métodos que se llamen `secuenciaNaturalIntA()`, `secuenciaNaturalIntB()`, `secuenciaNaturalIntC()` y `secuenciaNaturalIntD()` que reciben respectivamente un valor que representa el lado de una matriz cuadrada de tamaño `N` que deben generar según los patrones del gráfico adjunto. Devuelven respectivamente la matriz generada. Para un tamaño de 4x4 serían:

a)

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

b)

1	8	9	16
2	7	10	15
3	6	11	14
4	5	12	13

c)

7	11	14	16
4	8	12	15
2	5	9	13
1	3	6	10

d)*

1	12	11	10
2	13	16	9
3	14	15	8
4	5	6	7

- Prueba los métodos pedido desde `main()`.
- Hay que utilizar dos bucles anidados y manipular los índices de manera que las secuencias de ejecución produzcan los patrones de distribución indicados. Para el caso d) se puede aplicar la siguiente estrategia: a partir de la posición `(0,0)` hay que bajar `N` veces. A continuación, se pasa a la derecha `N-1` veces, luego sube `N-1` veces, luego a la izquierda `N-2` veces, luego hacia abajo `N-2` veces, etc. Cada vez que se mueve un paso se produce un número de la secuencia natural `1, 2, 3, ..., NxN`.
- Documenta el código fuente con comentarios aclaratorios adicionales.

12. Escribe un método que se llame `indiceAfabetico()` que recibe una palabra como texto. Devuelve un array (vector) de números conteniendo la posición que le corresponde a cada letra de la palabra recibida según el alfabeto español.

- Prueba el método pedido desde `main()`.
- Se puede resolver con un vector de `char` que contenga todos las letras ordenadas del alfabeto español y dos bucles for anidados (uno para las letras de la palabra y otro para el array del alfabeto). También se pueden utilizar las propiedades de los sistemas de codificación ASCII o unicode.
- Documenta el código fuente con comentarios aclaratorios adicionales.

13. Escribe un método que se llame `ordenado()` que recibe un vector de enteros . Devuelve **`true`** o **`false`** indicando si los datos están ordenados o no.

- Prueba el método pedido desde `main()`.
- Consulta en el anexo 7: Ordenación de vectores
- Documenta el código fuente con comentarios aclaratorios adicionales.

14. Escribe un método que se llame `burbuja()` que recibe un vector de enteros para ordenarlos por el método de la burbuja . Devuelve `1` si ha ordenado, `0` si ya estaba ordenado y un valor negativo si no se ha podido ordenar.

- Prueba el método pedido desde `main()`.
- Consulta en el anexo 7: Ordenación de vectores
- Documenta el código fuente con comentarios aclaratorios adicionales.

15. Escribe un método que se llame `baraja()` que recibe un vector de enteros para ordenarlos por el método de la baraja . Devuelve `1` si ha ordenado, `0` si ya estaba ordenado y un valor negativo si no se ha podido ordenar.

- Prueba el método pedido desde `main()`.
- Consulta en el anexo 7: Ordenación de vectores
- Documenta el código fuente con comentarios aclaratorios adicionales.

16. Escribe un método que se llame `mergeSoft()` que recibe un vector de enteros para ordenarlos por el método *MergeSoft* . Devuelve `1` si ha ordenado, `0` si ya estaba ordenado y un valor negativo si no se ha podido ordenar.

- Prueba el método pedido desde `main()`.
- Consulta en el anexo 7: Ordenación de vectores
- Documenta el código fuente con comentarios aclaratorios adicionales.

17. Escribe un método que se llame `quickSort()` que recibe un vector de enteros para ordenarlos por el método *QuickSort* . Devuelve `1` si ha ordenado, `0` si ya estaba ordenado y un valor negativo si no se ha podido ordenar.

- Prueba el método pedido desde `main()`.
- Consulta en el anexo 7: Ordenación de vectores
- Documenta el código fuente con comentarios aclaratorios adicionales.

18. Escribe un programa que busque todos los números primos en el rango `[1 .. 10000000]`

- Buscar información en Internet para *"Criba de Eratóstenes"* .
- Documenta el código fuente con comentarios aclaratorios adicionales.

19. Escribe un método que se llame `mayorArea()` que recibe una matriz rectangular de tamaño `(n,m)` con datos. Devuelve la mayor cantidad de celdas contiguas encontradas con el mismo valor y que comparte al menos un lado. Aquí se muestra un ejemplo en el que hay un área formada por 13 elementos con el mismo valor 3:

- Este ejercicio puede ser bastante complicado. Se pueden utilizar algoritmos *DFS* o *BFS* de búsqueda en profundidad y anchura sobre grafos. Se puede buscar información y ejemplos de ellos en Internet
- Documenta el código fuente con comentarios aclaratorios adicionales.

Fuentes y bibliografía

- NAKOV, S. *Fundamentos de programación con Java*. Ed. Faber, Veliko Tarnovo, 2009. [en linea][traducción automática del búlgaro] http://www.introprogramming.info/intro-java-book/
- CARRERES, J. *Manual de Java*. [en línea] http://www.oocities.org/collegepark/quad/8901/indice.html