

Programación orientada a objetos

Capítulo 7

Colecciones de tamaño fijo

Tutor: Manuel Fernández Barcell

Centro Asociado de Cádiz

<http://prof.mfbarcell.es>

7.1 Colecciones de tamaño fijo: “arreglos” o “matrices”

Concepto

Un arreglo es un tipo especial de colección que puede almacenar un número fijo de elementos.

A veces el tamaño máximo de una colección puede estar predeterminado. Para ello los lenguajes de programación suelen ofrecer un tipo de colección de tamaño fijo:
matrices

- El acceso a los elementos de un arreglo es generalmente más eficiente que el acceso a los elementos de una colección de tamaño flexible.
- Los arreglos son capaces de almacenar objetos o valores de tipos primitivos. Las colecciones de tamaño flexible sólo pueden almacenar objetos³.

7.3.1 Declaración de variables matriz

```
private int[ ] contadoresPorHora;
```



La característica distintiva de la declaración de una variable de tipo arreglo es un par de corchetes que forman parte del nombre del tipo: `int[]`. Este detalle indica que la variable `contadoresPorHora` es de tipo *arreglo de enteros*. Decimos que `int` es el tipo base de este arreglo en particular. Es importante distinguir entre una declaración de una variable arreglo y una declaración simple ya que son bastante similares:

```
int hora;
```

```
int[ ] contadoresPorHora;
```

7.3.2 Creación de objetos “matriz”

La forma general de la construcción de un objeto arreglo es:

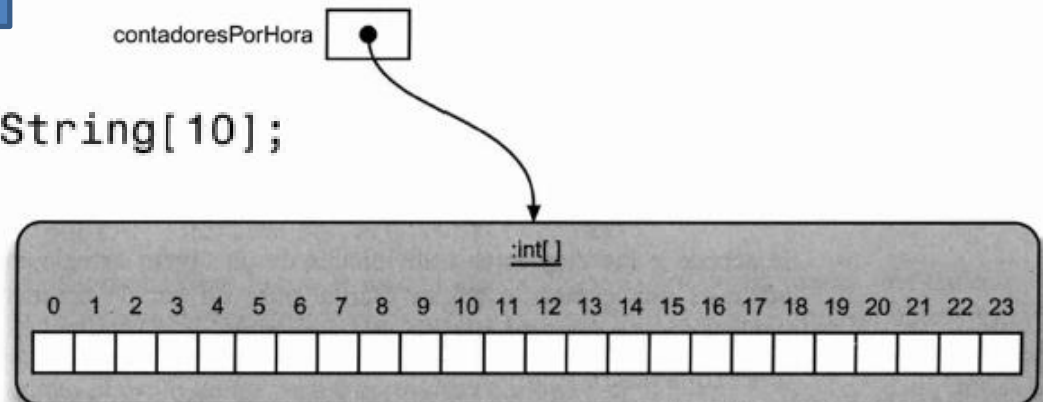
```
new tipo[expresión-entera]
```

La elección del tipo especifica de qué tipo serán todos los elementos que se almacenarán en el arreglo. La expresión entera especifica el tamaño del arreglo, es decir, un número fijo de elementos a almacenar.

```
int[ ] contadoresPorHora;  
contadoresPorHora = new int[24];
```

En un solo paso

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



7.3.3 Usar objetos “matriz”

Se accede a los elementos individuales de un objeto arreglo mediante un *índice*. Un índice es una expresión entera escrita entre un par de corchetes a continuación del nombre de una variable arreglo. Por ejemplo:

```
etiquetas[6];  
maquinas[0];  
gente[x + 10 - y];
```

Los valores válidos para una expresión que funciona como índice depende de la longitud del arreglo en el que se usarán. Los índices de los arreglos siempre comienzan por cero y van hasta el valor uno menos que la longitud del arreglo. Por lo que los índices válidos para el arreglo `contadoresPorHora` son desde 0 hasta 23, inclusive.

```
etiqueta[5] = "Salir";  
double mitad = lecturas[0] / 2;  
System.out.println(gente[3].getNombre());  
maquinas[0] = new MaquinaDeBoletos(500);
```

Acceso a los elementos de una matriz

- El primer elemento de una matriz tiene el índice 0
- Se comprueba automáticamente los límites de la matriz
- Si se intenta acceder fuera de los límites de la matriz(entre 0 y length-1), se produce la excepción *IndexOutOfBoundsException*
- Tamaño de un array: miembro.length

```
A.length           // correcto
A.length()         // ERROR: no se usan paréntesis

for (int i = 0; i < A.length; i++)  A[i]=i;
```

El tamaño de un array se especifica al crearlo con el operador **new**

```
int A[] = new int[10];    // array de 10 enteros:  
                          // A[0]..A[9]
```

```
String[] S;
```

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

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

Array de arrays

```
int[][] A;
```

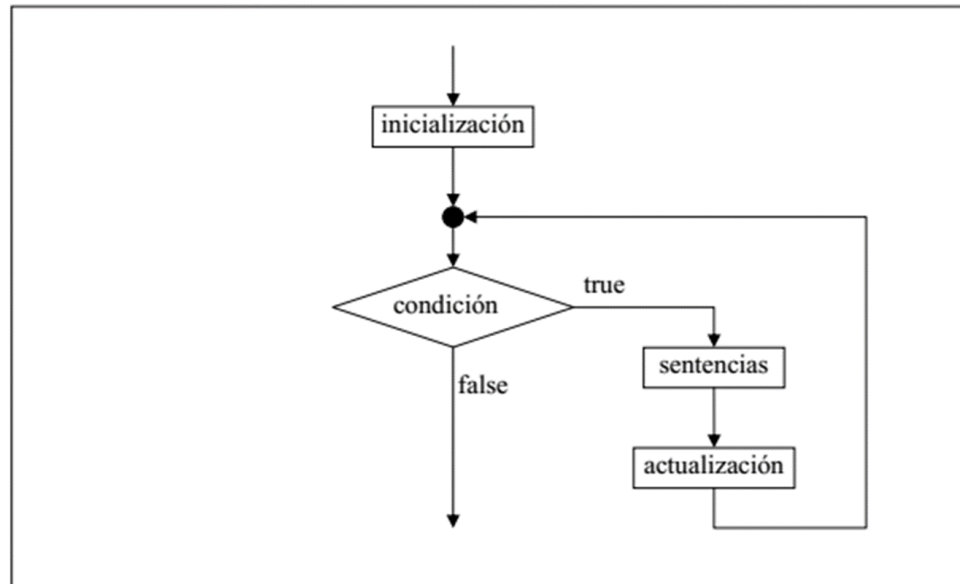
```
A = new int[2][3];
```

7.4 El ciclo “for”

Un *ciclo for* tiene la siguiente forma general:


```
for (inicialización; condición; acción modificadora) {  
    setencias a repetir  
}
```

- La inicialización se realiza sólo una vez, antes de la primera iteración
- La condición se comprueba cada vez antes de entrar al bucle. Si es cierta, se entra. Sino, se termina.
- La actualización se realiza siempre al terminar de ejecutar la iteración, antes de volver a comprobar la condición
- Los bucles **for** son una alternativa a los bucles **while** cuando el número de repeticiones es conocido



Variable local al bucle

for loop version



```
for(int hour = 0; hour < hourCounts.length; hour++) {  
    System.out.println(hour + ": " + hourCounts[hour]);  
}
```

while loop version

```
int hour = 0;  
while(hour < hourCounts.length) {  
    System.out.println(hour + ": " + hourCounts[hour]);  
    hour++;  
}
```

Número de Línea	Código
4	class Examen {
5	private int i;
6	public void ejemplo () {
7	for (int i=0; i<5; i++)
8	System.out.print (this.i++);
9	}
10	}

- a. Imprime 00000.
- b. Imprime 01234.
- c. Imprime infinitos ceros.
- d. Se producirá un error en tiempo de compilación por no estar inicializada la propiedad i.

Comparación de “for” con “while” y “for-each”

```
for(int hora = 0; hora < contadoresPorHora.length; hora++)  
{  
    System.out.println(hora + ": " +  
contadoresPorHora[hora]);  
}
```

```
int hora = 0;  
while (hora < contadoresPorHora.length) {  
    System.out.println(hora + ": " + contadoresPorHora[hora]);  
    hora++  
}
```

```
for(int valor : contadoresPorHora) {  
    System.out.println(": " + valor);  
}
```

7.4.2 El bucle for y los iteradores

Para eliminar elementos de una colección

```
for(Iterator<Track> it = tracks.iterator(); it.hasNext(); ) {  
    Track t = it.next();  
    if(t.getArtist().equals(artist)) {  
        it.remove();  
    }  
}
```

```
// Imprime múltiplos de 3 que sean menores que 40  
for(int num = 3; num < 40; num = num + 3) {  
    System.out.println(num);  
}
```

7.5.1 El operador condicional

Calcular el mayor de dos números con Java es una cosa realmente sencilla con Java. Si preguntas a cualquier programador te dirá que puedes utilizar un simple if-then-else. Quedando un programa de pocas líneas.

```
1. if (x>y)
2.     mayor = x;
3. else
4.     mayor = y;
```

Pero lo que, sorprendentemente, muchos programadores Java no saben, es que existe un operador condicional ternario ?: que nos ayuda a realizar estas operaciones con mucho menor código.

La estructura del operador ternario ?: es la siguiente:

```
resultado = (condicion)?valor1:valor2;
```

Así, si queremos calcular el mayor de dos números tendremos el siguiente código:

```
1. mayor=(x>y)?x:y;
```

7.6 matriz de mas de una dimensión

```
1  public class PruebaApp {
2
3      public static void main(String[] args) {
4
5          //Definimos un array de 3 filas x 5 columnas
6          int array[][]=new int [3][5];
7
8          //Asignamos un 5 al array, en la fila 0 columna 1
9          array[0][1]=5;
10
11      }
12
13 }

1  public class PruebaApp {
2
3      public static void main(String[] args) {
4
5          //Definimos un array de 3 filas x 5 columnas
6          int array[][]={{1,2,3,4,5}, {6,7,8,9,10}, {11,12,13,14,15}};
7
8      }
9
10 }
```

Para recorrer un array, debemos usar un bucle anidado, donde el primer bucle controla las filas y el de dentro las columnas. Veamos un ejemplo:

```
1 public class PruebaApp {
2
3     public static void main(String[] args) {
4
5         //Definimos un array de 3 filas x 5 columnas
6         int array[][]={{1,2,3,4,5}, {6,7,8,9,10}, {11,12,13,14,15}};
7
8         //Recorremos el array multidimensional
9         for (int i=0;i<array.length;i++){
10             for(int j=0;j<array[0].length;j++){
11                 System.out.println(array[i][j]);
12             }
13         }
14     }
15
16 }
```

Como vemos en el ejemplo anterior, vemos **array.length**, devuelve la longitud de las filas (3) y **array[0].length** devuelve la longitud de las columnas (5).

Pregunta: Según el texto de la bibliografía básica de la asignatura, indique cuál de las siguientes afirmaciones es correcta:

- a. El lenguaje Java tiene tres variantes del ciclo for : for-each, for y for-do.
- b. Un ciclo while es similar en su estructura y propósito que el ciclo for-each.
- c. El tipo de la variable de ciclo no tiene porqué ser el mismo que el tipo del elemento declarado para la colección que estamos recorriendo con un ciclo.
- d. Un índice es un objeto que proporciona funcionalidad para recorrer todos los elementos de una colección.

Pregunta: Según el texto de la bibliografía básica de la asignatura, indique cuál de las siguientes afirmaciones es correcta:

- a. Las colecciones de objetos son objetos que pueden almacenar un número predeterminado e invariable de otros objetos.
- b. Un iterador es un objeto que proporciona funcionalidad para recorrer todos los elementos de una colección.
- c. Un ciclo consiste en la escritura repetida de un bloque de sentencias.
- d. Un arreglo (array) es un tipo especial de colección que puede almacenar un número variable de elementos.

Pregunta: Según el texto de la bibliografía básica de la asignatura, indique cuál de las siguientes afirmaciones es correcta en relación a la clase Vector de Java:

- a. Es Final
- b. Implementa java.util.List
- c. Es serializable
- d. Dispone de un solo constructor

¿Qué ciclo debo usar? Hemos hablado sobre tres ciclos diferentes: el *ciclo for*, el *ciclo for-each* y el *ciclo while*. Como habrá visto, en muchas situaciones el programador debe seleccionar el uso de alguno de estos ciclos para resolver una tarea. Generalmente, un ciclo puede ser rescrito mediante otro ciclo. De modo que, ¿cómo puede hacer para decidir qué ciclo usar en una situación? Ofrecemos algunas líneas guías:

- Si necesita recorrer todos los elementos de una colección, el *ciclo for-each* es, casi siempre, el ciclo más elegante para usar. Es claro y conciso (pero no provee una variable contadora de ciclo).
- Si tiene un ciclo que no está relacionado con una colección (pero lleva a cabo un conjunto de acciones repetidamente), el *ciclo for-each* no resulta útil. En este caso, puede elegir entre el *ciclo for* y el *ciclo while*. El *ciclo for-each* es sólo para colecciones.
- El *ciclo for* es bueno si conoce anticipadamente la cantidad de repeticiones necesarias (es decir, cuántas vueltas tiene que dar el ciclo). Esta información puede estar dada por una variable, pero no puede modificarse durante la ejecución del ciclo. Este ciclo también resulta muy bueno cuando necesita usar explícitamente una variable contadora.
- El *ciclo while* será el preferido si, al comienzo del ciclo, no se conoce la cantidad de iteraciones que se deben realizar. El fin del ciclo puede determinarse previamente mediante alguna condición (por ejemplo, lee una línea de un archivo (repetidamente) hasta que alcanza el fin del archivo).

```
while (expresión_booleana)  
    instrucción
```

```
do  
    instrucción  
while (expresión_booleana)
```

```
for (instruccion_inicialización;  
    expresión_booleana;  
    instrucción_incremento)  
    instrucción
```

```
for(ElementType elemento : colección) {  
    cuerpo del bucle  
}
```

break	permite salir del ciclo
continue	salta a la siguiente iteración

Términos introducidos en este capítulo

colección, arreglo, iterador, *ciclo for-each*, *ciclo while*, *ciclo for*, índice, sentencia import, biblioteca, paquete, objeto anónimo

Resumen de conceptos

- **colecciones** Las colecciones de objetos son objetos que pueden almacenar un número arbitrario de otros objetos.
- **ciclo** Un ciclo se usa para ejecutar un bloque de sentencias repetidamente sin tener que escribirlas varias veces.
- **iterador** Un iterador es un objeto que proporciona funcionalidad para recorrer todos los elementos de una colección.
- **null** Se usa la palabra reservada de Java **null** para indicar que «no hay objeto» cuando una variable objeto no está haciendo referencia a un objeto en particular. Un campo que no ha sido inicializado explícitamente contendrá por defecto el valor **null**.
- **arreglo** Un arreglo es un tipo especial de colección que puede almacenar un número fijo de elementos.

Vídeos

- Matrices

- https://www.youtube.com/watch?v=UID_EKKfpcE&list=PLU8oAlHdN5BktAXdEVCLUYzvDyqRQJ2Ik&index=23
- https://www.youtube.com/watch?v=NwztwM_xGgU&list=PLU8oAlHdN5BktAXdEVCLUYzvDyqRQJ2Ik&index=24

- Matrices bidimensionales

- <https://www.youtube.com/watch?v=tUncS0AsNE&list=PLU8oAlHdN5BktAXdEVCLUYzvDyqRQJ2Ik&index=25>
- <https://www.youtube.com/watch?v=xEHkuRApCno&list=PLU8oAlHdN5BktAXdEVCLUYzvDyqRQJ2Ik&index=26>