11. Algunas clases estándar de Java (II)

Objetivos:
a) Presentar la clase predefinida en Java para trabajar con arrays
b) Interpretar el código fuente de una aplicación Java donde aparecen arrays
c) Construir una aplicación Java sencilla, convenientemente especificada, que emplee arrays.

Otra clase estándar muy emplea en Java es la clase *array* o *vector*. El término español *vector* para referirse a esta clase de objetos puede dar lugar a confusión ya que, como se verá más adelante, existe una clase `Vector` predefinida en Java. En este caso y sin que sirva de precedente, es conveniente emplear el término inglés *array*.

11.1. Objetos de la clase array

Un **objeto** de la clase predefinida *array* permite representar una secuencia lineal y finita de elementos del mismo tipo. Estos elementos pueden ser de tipo primitivo o pertenecientes a otras clases. Los *arrays* son objetos con características propias. Además del conjunto de elementos del mismo tipo, un objeto de tipo *array* almacena en memoria una constante numérica entera que representa el número de elementos o tamaño del *array*.

La declaración de un puntero o referencia a un *array* se lleva a cabo de la siguiente manera:

```
tipoElemento [] identificadorInstancia;
```

Por ejemplo, la declaración de un puntero o referencia a un *array* de números enteros:

```
int [] p;    // se declara el puntero o referencia del array de enteros
```

Como ocurre con los demás objetos, la ejecución de la sentencia anterior sólo crea la referencia del *array* (Figura 11.1), pero no el *array* en sí.

Figura 11.1. Creación de la referencia p

En la declaración del identificador de la variable tampoco se especifica el tamaño del *array* a referenciar. El tamaño del *array* se declara y establece cuando se crea la instancia *array*. La creación de un vector o *array*, una vez declarado su referencia se hace de la siguiente forma:

```
identificadorInstancia = new tipoElemento [numeroElementos];
```

Por ejemplo:

```
p = new int [5];      // se crea el array de 100 enteros referenciado por p
```

De esta manera, se reserva espacio para todos los elementos del *array*: 5 valores enteros (Figura 11.2).

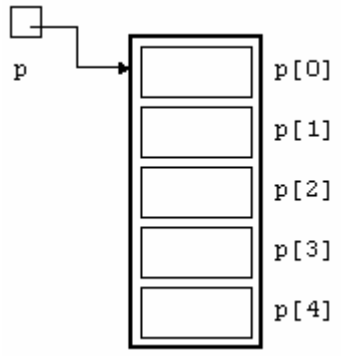

Figura 11.2. Creación del *array* de enteros cuya dirección de memoria es almacenada en la referencia `p`

La creación de una referencia/puntero y del *array* de enteros al que apunta puede llevarse a cabo de forma simultánea en la misma sentencia:

```
int [] p = new int [5];       // se declara el puntero y se crea el vector
```

11.2. Operaciones con arrays

Puede accederse al tamaño en el código fuente del programa mediante el **atributo** `length` que devuelve el número de elementos de un *array* (*`identificadorArray`*`.length`).

El acceso a cada elemento del *array* se realiza con el identificador de la instancia, la pareja de corchetes y su índice (un valor numérico **entero**), considerando que el primer elemento tiene índice 0:

```
p[0] = -15;   // Al primer elemento del array se le asigna el valor -15
p[2] = 26;    // Al segundo elemento del array se le asigna el valor 26
p[2] = 34;    // Al tercer elemento del array se le asigna el valor 34
```

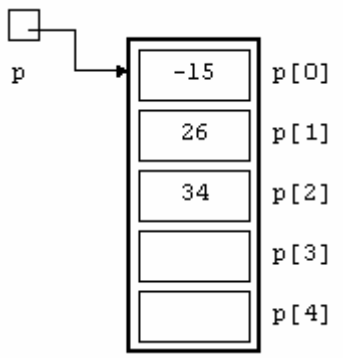

Figura 11.3. Identificación y asignación de valores a algunos elementos del *array*.

Por lo tanto, a un *array* de n elementos le corresponde siempre un índice válido dentro del intervalo [0 ... n-1]. Un error muy común es intentar acceder al elemento de índice `n`, que no existe (se producirá un error de ejecución *out of bounds*). Para especificar el índice de un elemento pueden emplearse tanto constantes como variables de tipo entero.

Otro error es tratar de emplear el identificador del *array* como si fuera un dato de tipo primitivo. Por ejemplo, la sentencia

```
p = p + 34;
```

generará un error de compilación.

Se suelen emplear bucles cuando se quiere trabajar con todos los elementos de un *array*. El siguiente ejemplo muestra como trabajar con todos los elementos de dos *arrays* de números (uno de valores enteros y otro de valores reales). El segundo de ellos almacena la raiz cuadrada de los valores almacenados en el primero:

```
/**
 *  ArrayRaices: Ejemplo de uso de arrays
 *  A. Garcia-Beltran - abril, 2007
 */
public class ArrayRaices {
    public static void main (String [] args ) {
        int [] numero = new int[10];           // Array de valores enteros
        double [] raiz = new double[10];       // Array de valores reales
        for (int i=0; i<numero.length; i++) {
            numero[i] = i+1;
            raiz[i] = Math.sqrt(numero[i]);
            System.out.println(numero[i] + " : " + raiz[i]);
        }
    }
 }
```

Es importante ver que el bucle `for` emplea la expresión `i<numero.length` como condición de término. El código anterior puede ejecutarse, mostrando la siguiente salida por pantalla:

```
$>java ArrayRaices↵

1 : 1.0
2 : 1.4142135623730951
3 : 1.7320508075688772
4 : 2.0
5 : 2.23606797749979
6 : 2.449489742783178
7 : 2.6457513110645907
8 : 2.8284271247461903
9 : 3.0
10 : 3.1622776601683795
```

También puede llevarse a cabo la declaración e inicialización simultánea de un *array* (con los valores correspondientes separados por comas y entre llaves) en la misma sentencia de código. Por ejemplo:

```
double [] x = {1.5, 2.3, -0.6, 4.8};
```

Este tipo de inicialización sólo es válida en la sentencia de declaración de la referencia del *array*. El resultado de la ejecución de la sentencia anterior se muestra en la Figura 11.4.

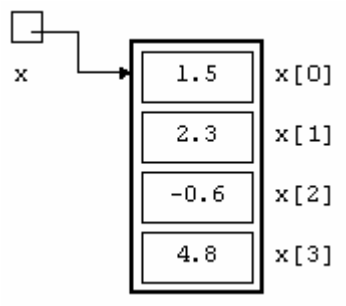

Figura 11.4. Creación de la referencia, del *array* de números reales e inicialización de los valores de sus elementos al ejecutarse una única sentencia.

A continuación se muestra un programa más completo que calcula la media aritmética de una serie de valores reales introducidos vía teclado.

```
/**
 *  NotaMedia: Ejemplo de empleo de vector de reales
 *  A. Garcia-Beltran - abril, 2007
 */
import java.io.*;
public class NotaMedia {
   public static void main (String [] args ) {
       double [] notas = leerNotas();
       if (notas.length==0)
          System.out.println("No hay notas");
          else {
               int i;
               double suma=0;
               for (i=0; i<notas.length; i++) {
                   System.out.println("Nota " + (i+1) + ": " + notas[i]);
                   suma+=notas[i];
                   }
               System.out.println("La nota media es: " + suma/i);
       }
   }
   public static double [] leerNotas( ) {
       BufferedReader in = new BufferedReader (new
                                InputStreamReader(System.in));
       double valor = 0;
       double [] vector = new double [20];
       int indice = 0;
       String cadena;
       System.out.println("Introduce una nota por linea");
       try {
           while ( (cadena = in.readLine()) != null ) {
             valor = Double.parseDouble(cadena);
             vector[indice] = valor;
             indice++;
             }
           }
       catch ( Exception e ) { }  // No se procesan los errores
    }
}
```

El código anterior puede compilarse y ejecutarse, mostrando la siguiente salida por pantalla:

```
$>java NotaMedia↵

Introduce una nota por linea
```

```
8↵
5.25↵
9↵

Nota 1: 8.0
Nota 2: 5.25
Nota 3: 9.0
La nota media es: 7.416666666666667
```

11.3. Array de objetos

El uso de vectores no tiene por qué restringirse a elementos de tipo primitivo. Por ejemplo, pueden crearse sendas referencias a *arrays* de referencias de la clase `precio` (tal y como se declaró esta clase en un ejemplo anterior) o de la clase `String`:

```
precio [] catalogo;            // Creacion de la referencia
catalogo = new precio [5];     // Creación del array de referencias
catalogo[0] = new precio();    // Creación del primer elemento/instancia precio
```

El resultado de la ejecución de las sentencias anteriores se muestra en la Figura 11.5. Primero se crea la referencia al *array* de punteros, luego se crea el array de punteros y, finalmente, se crea la instancia de la clase `precio` y se almacena su dirección de memoria en el primer elemento del *array* de punteros.

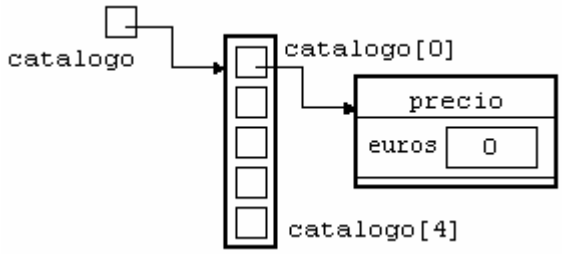

Figura 11.5. Creación de la referencia, del *array* de referencias y de la instancia de la clase precio referenciada por el primer elemento del *array*.

En el caso de trabajar con un *array* de `String` entonces podrían emplearse las siguientes sentencias:

```
String [] Tabla;               // Creacion de la referencia
Tabla = new String [10];       // Creación del array de referencias
Tabla[0] = new String();       // Creación del primer elemento/instancia String
```

En el siguiente código se muestra un ejemplo de cómo trabajar con el *array* de objetos de la clase `precio`:

```
/**
 *  ArrayPrecios: Ejemplo de uso de arrays de objetos
 *  A. Garcia-Beltran - abril, 2007
 */
public class ArrayPrecios {
  public static void main (String [] args) {
    precio [] catalogo;                 // Creacion de la referencia
    catalogo = new precio [5];          // Creacion array de referencias precio

    for (int i=0; i<catalogo.length; i++) {
       catalogo[i] = new precio();      // Creacion del elemento i-esimo
```

```
       catalogo[i].pone(10*Math.random());
       System.out.println("Producto "+ i + " : " + catalogo[i].da());
       }

    // Busqueda del maximo precio
    double maximo=catalogo[0].da();
    for (int i=1; i<catalogo.length; i++) {
       if (maximo<catalogo[i].da()) maximo=catalogo[i].da();
       }
    System.out.println("El mas caro vale "+ maximo +" euros");

  }
}
```

La ejecución y salida por pantalla del código anterior ya compilado es la siguiente:

```
$>java ArrayPrecios↵

Producto 0 : 1.92170958406773
Producto 1 : 9.995382834390597
Producto 2 : 2.794443755715136
Producto 3 : 5.366258891245966
Producto 4 : 1.0105636724630673
El mas caro vale 9.995382834390597 euros
```

Otro ejemplo: los argumentos de la linea de ejecución de un programa son un *array* de `String`s:

```
/**
 *  Ecos: Ejemplo de manipulacion de cadenas
 *  A. Garcia-Beltran - marzo, 2007
 */
public class Ecos {
  public static void main (String [] args) {
    int i=0;
    System.out.println("Eco de palabras:");
    while (i < args.length) {
      System.out.print(args[i] + " ");
      i++;
      };
    System.out.println();
    System.out.println("Eco inverso de palabras:");
    for (i=0; i < args.length; i++)
        System.out.print(args[args.length-i-1] + " ");
    System.out.println();
    System.out.println("Eco inverso de caracteres:");
    for (i=args.length-1; i >= 0; i--) {
        for (int k=0; k<args[i].length(); k++)
            System.out.print(args[i].charAt(args[i].length()-k-1));
        System.out.print(" ");
        }
    }
}
```

Ejemplo de ejecución y salida por pantalla:

```
$>java Ecos Esto es una prueba↵

Eco de palabras:
Esto es una prueba
Eco inverso de palabras:
```

```
prueba una es Esto
Eco inverso de caracteres:
abeurp anu se otsE

$>
```

11.4. Arrays multidimensionales

En Java también se puede trabajar con arrays multidimensionales, llamados comúnmente matrices. Por ejemplo, la declaración de un puntero (Figura 11.6) a una matriz bidimensional de números enteros se puede realizar de la siguiente manera incluyendo un par de corchetes adicionales en la declararación de la referencia:

```
int [][] m;
```

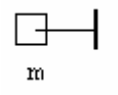

Figura 11.6. Creación de la referencia `m`

La creación de la matriz (una vez declarado su puntero) puede hacerse según muestra la siguiente sentencia:

```
m = new int [5][4];    // Crea una matriz para almacenar 5 x 4 enteros
```

En este caso se indican en el par de cochetes adicional el número de elementos según la segunda dimensión. El resultado de la ejecución de la sentencia anterior se muestra en la Figura 11.7.

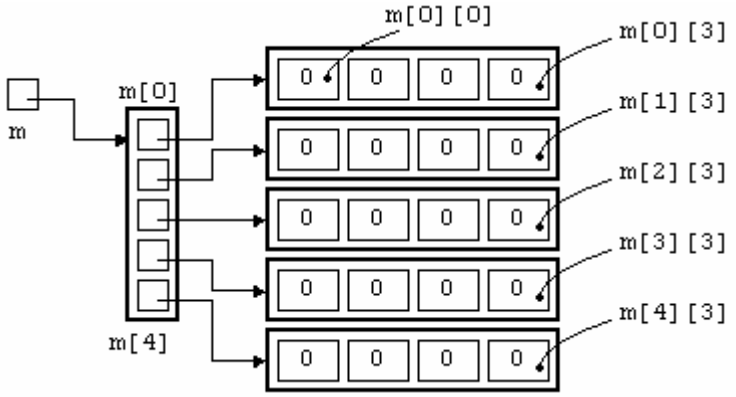

Figura 11.7. Creación de un *array* bidimensional de 5 x 4 enteros

La ejecución del código anterior reserva espacio en memoria durante la ejecución del programa para una matriz de 5 x 8 valores numéricos enteros. En realidad se reserva espacio en memoria para un array de cinco punteros que, a su vez, almacenan la dirección de memoria de otros tantos arrays de cuatro números enteros. Para referenciar a cada uno de los elementos de los arrays de enteros es necesario utilizar el identificador del puntero seguido de los correspondientes índices entre corchetes. Por ejemplo, como en las siguientes sentencias:

```
m[0][0] = 34;
m[0][1] = 46;
m[0][2] = 13;
m[0][3] = -8;
```

```
m[1][0] = 5;
m[1][1] = 56;
m[1][2] = -3;
```

Por otro lado, también puede llevarse a cabo la declaración de la referencia, creación del array multidimensional e inicialización simultánea de sus elementos. Por ejemplo, la siguiente sentencia genera una matriz 2 x 3 de números enteros:

```
// Matriz de dos filas y tres columnas
int [][] a = { {1, 2, 3}, {4, 5, 6} };
// Por ejemplo, el elemento a[1][0] contiene el valor 4
```

El resultado se muestra en la Figura 11.8.

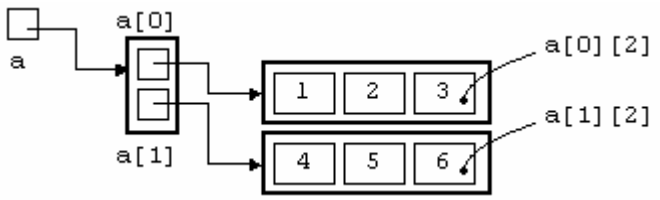

Figura 11.8. Creación de un *array* bidimensional de 2 x 3 números enteros e inicialización de sus elementos.

El número de elementos de los *arrays* referenciados puede especificarse para cada uno de ellos y ser diferente entre sí. Por ejemplo, la ejecución de la sentencia:

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

genera la estructura de datos representada en la Figura 11.9. En dicha estructura no existen, por ejemplo, los elementos `b[0][3]` ni `b[2][2]`.

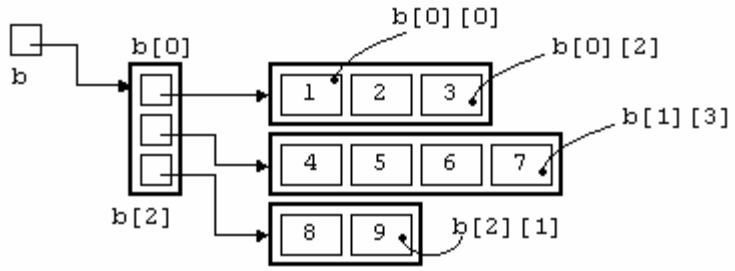

Figura 11.9. Creación de un *array* bidimensional no rectangular

En Java es posible trabajar con estructuras de tipo *array* con más de dos dimensiones añadiendo simplemente los índices (y corchetes o llaves) que sean necesarios. Por ejemplo, la siguiente sentencia crea una matriz tridimensional 2 x 2 x 3 de números enteros:

```
int [][][] c = { { {1, 2, 3}, {4, 5, 6} }, { {7, 8, 9}, {10, 11, 12} } }
```

El número de pares de **corchetes** en el término de la izquierda y de **niveles de llaves** en el término de la derecha debe coincidir e indica el de dimensiones de la matriz. En principio, en Java no hay límite para el número de dimensiones que pueden utilizarse en una matriz.

El siguiente programa genera dos matrices reales con elementos de valor aleatorio, visualiza dichas matrices, las suma y, finalmente visualiza la matriz suma.

```
/**
 *  SumaMatrices: Ejemplo de arrays multidimensionales
 *  A. Garcia-Beltran - marzo, 2007
 */
public class SumaMatrices {
    public static void main (String [] args ) {
       double [][] a = new double[3][3];
       double [][] b = new double[3][3];
       double [][] c = new double[3][3];
       // Asignar valores aleatorios a las elementos de las matrices sumando
       for (int i=0; i<3; i++)
            for (int j=0; j<3; j++) {
                a[i][j] = Math.random();
                b[i][j] = Math.random();
                }
       // Visualizar los valores de los elementos por pantalla
       System.out.println("Primera matriz sumando:");
       muestra(a);
       System.out.println("Segunda matriz sumando:");
       muestra(b);
       // Realizar la suma
       for (int i=0; i<3; i++) {
            for (int j=0; j<3; j++)
                c[i][j] = a[i][j]+b[i][j];
            }
       // Visualizar por pantalla la matriz suma
       System.out.println("Matriz suma:");
       muestra(c);
       }
    public static void muestra (double [][] m) {
       for (int i=0; i<3; i++) {
            for (int j=0; j<3; j++)
                 System.out.print(m[i][j] + "  ");
            System.out.println();
            }
       return;
       }
}
```

Ejemplo de ejecución y salida por pantalla:

```
$>java SumaMatrices↵

Primera matriz sumando:
0.6813532584445761  0.5184944306800717  0.5907331172325986
0.12223454631468555  0.9551053431516023  0.6331217713823304
0.8597845852228786  0.7187706185540627  0.5622659371857616
Segunda matriz sumando:
0.17929777458376628  0.38589355399665426  0.9181828632082493
0.7474050764717326  0.45146577347486194  0.06754244312639623
0.49367466805515114  0.14680828058239193  0.4933199328308062
Matriz suma:
0.8606510330283423  0.904387984676726  1.5089159804408478
0.8696396227864182  1.4065711166264643  0.7006642145087266
1.3534592532780296  0.8655788991364546  1.055585870016568
```

Se propone modificar el programa anterior creando dos métodos que, respectivamente, almacenen valores aleatorios en una matriz y sumen dos matrices.