Multidimensional Arrays

Aunque el array unidimensional es el array más comúnmente utilizada en la programación, los arrays multidimensionales (matrices de dos o más dimensiones) ciertamente no son raros. En Java, una array multidimensional es una array de arrays.

Two-Dimensional Arrays

The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array **table** of size 10, 20 you would write

```
int table[][] = new int[10][20];   // 10 filas por 20 columnas en cada fila
```

To access point 3, 5 of array **table**, you would use **table[3][5]**.

```
public class DosDimensiones {
   public static void main(String args[]) {
      int t, i;
      int table[][] = new int[3][4];
      for (t = 0; t < 3; ++t) {
         for (i = 0; i < 4; ++i) {
            table[t][i] = (t * 4) + i + 1;
            System.out.print(table[t][i] + "\t");//tabulador \t
         }
         System.out.println();
      }
   }
}
```

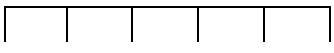
Ejercicios:

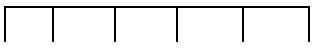
i. Realiza un programa que cargue desde teclado una tabla bidimensional de 3x2 de números enteros. Posteriormente el programa pedirá un número a buscar, y en caso de encontrarlo indicará el número de fila y columna donde se ha encontrado por primera vez. El programa ofrecerá al usuario la opción de seguir buscando más números.

ii. Realiza un programa que cargue una tabla de 5 filas y 7 columnas con los siguientes números, para finalmente mostrarla (NOTA: Se debe rellenar con los valores del 1 al 35 controlados mediante un contador, no se debe inicializar la matriz al declararla).

35	30	25	20	15	10	5
34	29	24	19	14	9	4
33	28	23	18	13	8	3
32	27	22	17	12	7	2
31	26	21	16	11	6	1

iii. Crea una tabla bidimensional de 5 * 5, rellénala con los números del 1 al 25 y muestra su contenido en pantalla. Posteriormente, guarda sobre la misma matriz y sin utilizar una matriz auxiliar, el resultado de su matriz transpuesta. Vuelve a mostrar su contenido en pantalla:





1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24
5	10	15	20	25

iv. Realiza un programa que dibuje un cuadrado mágico de orden impar. Un cuadrado mágico es aquel en el que sin repetir ningún número, todas las filas, columnas y las dos diagonales suman lo mismo.

v. Realiza un programa que cargue una tabla aleatoriamente de enteros de dimensión 3x4. El programa mostrará la fila en la que la suma de sus elementos sea mayor.

Arrays of Three or More Dimensions

Java allows arrays with more than two dimensions. Here is the general form of a multidimensional array declaration:

type name[ ][ ]...[ ] = new *type*[*size1*][*size2*]...[*sizeN*];

For example, the following declaration creates a 4 × 10 × 3 three-dimensional integer array.

int multidim[][][] = new int[4][10][3];

Initializing Multidimensional Arrays

A multidimensional array can be initialized by enclosing each dimension's initializer list within its own set of curly braces. For example, the general form of array initialization for a two-dimensional array is shown here:

type-specifier array_name[ ] [ ] = { { *val, val, val, ..., val* },
{ *val, val, val, ..., val* },
. . .
{ *val, val, val, ..., val* } };

Here, *val* indicates an initialization value. Each inner block designates a row. Within each row, the first value will be stored in the first position of the subarray, the second value in the second position, and so on. Notice that commas separate the initializer blocks and that a semicolon follows the closing **}**.

Assigning Array References

Al igual que con otros objetos, cuando asigna una variable de referencia de matriz a otra, simplemente está cambiando a qué objeto se refiere esa variable. No está haciendo una copia de la matriz, ni está haciendo que el contenido de una matriz se copie a la otra. Por ejemplo, considere este programa:

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

```
        int i;
        int nums1[] = new int[10];
        int nums2[] = new int[10];
        for (i = 0; i < 10; i++) {
           nums1[i] = i;       }
        for (i = 0; i < 10; i++) {
           nums2[i] = -i;       }

        System.out.print("Here is nums1: ");
        for (i = 0; i < 10; i++) {
           System.out.print(nums1[i] + " ");       }
        System.out.println();

        System.out.print("Here is nums2: ");
        for (i = 0; i < 10; i++) {
           System.out.print(nums2[i] + " ");       }
        System.out.println();

        nums2 = nums1; // Ahora num2 apunta a num1

        System.out.print("Here is nums2 after assignment: ");
        for (i = 0; i < 10; i++) {
           System.out.print(nums2[i] + " ");       } //Los mismos valores que num1
        System.out.println();

        nums2[3] = 99;  //También estoy almacenando el nº 99 en num1
        System.out.print("Here is nums1 after change through nums2: ");
        for (i = 0; i < 10; i++) {
           System.out.print(nums1[i] + " ");       }
        System.out.println();
    }
}
```

```
Here is nums1: 0 1 2 3 4 5 6 7 8 9
Here is nums2: 0 -1 -2 -3 -4 -5 -6 -7 -8 -9
Here is nums2 after assignment: 0 1 2 3 4 5 6 7 8 9
Here is nums1 after change through nums2: 0 1 2 99 4 5 6 7 8 9
BUILD SUCCESSFUL (total time: 0 seconds)
```

Using the length Member

Debido a que las matrices se implementan como objetos, cada matriz tiene asociada una variable de longitud que contiene la cantidad de elementos que la matriz puede contener. (En otras palabras, la longitud contiene el tamaño de la matriz). Aquí hay un programa que muestra esta propiedad:

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

```
        int list[] = new int[10];
        int nums[] = {1, 2, 3};
        int table[][] = { // a variable-length table
                {1, 2, 3},
                {4, 5},
                {6, 7, 8, 9}
                };
        System.out.println("length of list is " + list.length);//10
        System.out.println("length of nums is " + nums.length);//3
        System.out.println("length of table is " + table.length);//3
        System.out.println("length of table[0] is " + table[0].length);//3
        System.out.println("length of table[1] is " + table[1].length);//2
        System.out.println("length of table[2] is " + table[2].length);//4
        System.out.println();

        for (int i = 0; i < list.length; i++) {
            list[i] = i * i;
        }
        System.out.print("Here is list: ");

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

The For-Each Style for Loop

Automáticamente recorre toda la matriz, obteniendo un elemento a la vez, en secuencia, de principio a fin.

```
public class ForEachPrueba {
    public static void main(String args[]) {
        int nums[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int sum = 0;
        for (int x : nums) {
            System.out.println("Value is: " + x);
            sum += x;
        }
        System.out.println("Summation: " + sum);
    }
}
```

There is one important point to understand about the for-each style **for** loop. Its iteration variable is "read-only" as it relates to the underlying array. An assignment to the iteration variable has no effect on the underlying array. In other words, **you can't change the contents of the array** by assigning the iteration variable a new value.(En otras palabras, no puede cambiar el contenido de la matriz asignando un nuevo valor a la variable de iteración).

```
public class ForEachPrueba2 {
   public static void main(String args[]) {
      int nums[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
      for (int x : nums) {
         System.out.print(x + " ");
         x = x * 10;  // Esta instrucción no tiene efecto,
                              //no cambia el valor de la celda del array
      }
      System.out.println();
      for (int x : nums) {
         System.out.print(x + " ");
      }
      System.out.println();
   }
}
```

Iterating Over Multidimensional Arrays with "for each"

Notice how **x** is declared. It is a reference to a one-dimensional array of integers. This is necessary because each iteration of the **for** obtains the next *array* in **nums**, beginning with the array specified by **nums[0]**.

```
public class ForEachMultidimensional {
   public static void main(String args[]) {
      int sum = 0;
      int nums[][] = new int[3][5];
      for (int i = 0; i < 3; i++) {
         for (int j = 0; j < 5; j++) {
            nums[i][j] = (i + 1) * (j + 1);
         }
      }
      for (int x[] : nums) {
         for (int y : x) {
            System.out.println("Value is: " + y);
            sum += y;
         }
      }
      System.out.println("Summation: " + sum);
   }
}
```

Strings

En programación, uno de los tipos de datos más importantes de Java es String. Cadena define y soporta cadenas de caracteres. En muchos otros lenguajes de programación, una cadena es una matriz de caracteres. Este no es el caso con Java. En Java, las cadenas son objetos.

En realidad, hemos usando la clase String desde el Capítulo 1. Cuando creas una cadena literal, en realidad estás creando un objeto String. Por ejemplo, en la declaración

System.out.println ("En Java, las cadenas son objetos.");

Constructing Strings

Puedes construir un String como lo haces con cualquier otro tipo de objeto: utilizando new y llamando al constructor de String. Por ejemplo:

```
String str = new String("Hello");
```

Esto crea un objeto String llamado str que contiene la cadena de caracteres "Hola".
También puedes construir una cadena a partir de otra cadena. Por ejemplo:

```
String str = new String("Hello");
String str2 = new String(str);
```

After this sequence executes, **str2** will also contain the character string "Hello".

Another easy way to create a **String** is shown here:

```
String str = "Java strings are powerful.";
```

Ejemplo:

```
class StringDemo {
   public static void main(String args[]) {
      String str1 = new String("Los Strings son objetos.");
      String str2 = "Se pueden crear de diferentes formas.";
      String str3 = new String(str2);
      System.out.println(str1);
      System.out.println(str2);
      System.out.println(str3);
   }
}
```

Operating on Strings

The **String** class contains several methods that operate on strings.

boolean equals(*str*)	*Devuelve verdadero si la cadena que invocas contiene la misma secuencia de caracteres que str.*
int length( )	Obtains the length of a string.
char charAt(*index*)	Obtains the character at the index specified by *index.*
int compareTo(*str*)	Devuelve menos que cero si la cadena que invocas es menor que str, mayor que cero si la cadena que invocas es mayor que str, y cero si las cadenas son iguales.
int indexOf(*str*)	Busca en la cadena de invocación la subcadena especificada por str. Devuelve el índice de la primera coincidencia o -1 en caso de error.
int lastIndexOf(*str*)	Busca en la cadena de invocación la subcadena especificada por str. Devuelve el índice de la última coincidencia o -1 en caso de error.
String toUpperCase()	Converts all of the characters in this String to upper case
String toLowerCase()	Converts all of the characters in this String to lower case.
String substring(int begin)	Returns a new string that is a substring of this string.
String substring(int begin, int end) desde posicion comienzo hasta posicion fin	Returns a new string that is a substring of this string.
boolean contains(CharSequences)	Returns true if and only if this string contains the specified sequence of char values.

Ejemplos:

```
class OperacionesStrings {
   public static void main(String args[]) {
    String str1 = " trabajar con Strings ...";
        String str2 = new String(str1);

        String str3 = "Los Strings en Java cuentan con muchos métodos";

        int result, idx;

        char ch;

        System.out.println("Longitud de str1: " + str1.length());

        //Mostrar carácter a carácter

        for (int i = 0; i < str1.length(); i++) {

           System.out.print(str1.charAt(i));

        }

        System.out.println();

        //Comparando Strings

        if (str1.equals(str2)) {

           System.out.println(str1 +"igual que"+ str2);

        } else {

           System.out.println(str1 +"no es igual que "+str2);

        }
```

```
            //De otra forma
            result = str1.compareTo(str3);
            if (result == 0) {
               System.out.println(str1+" y "+str3 +" son iguales");
            } else if (result < 0) {
               System.out.println(str1 +" es más pequeño que "+str3);
            } else {
               System.out.println(str1 +" es mayor que  "+str3);
            }
            // Buscar un String en otro
            str2 = "Uno dos tres tres tres";
            idx = str2.indexOf("tres");
            System.out.println("La primera aparición de \"tres\" en "+ str2 +" empieza en: " +
idx);
            idx = str2.lastIndexOf("tres");
            System.out.println("La última aparición de \"tres\" en "+ str2 +" empieza en: " +
idx);
            //Las tres primeras letras de str1
            str3 = str1.substring(0, 3);
            System.out.println("Las tres primeras letras de "+ str1+ " "+ str3);
            //Prueba del método contains
            if (str1.contains(str3))
               System.out.println("El String "+str1 +" contiene al string "+str3);
         }
      }
   }
```

Ejercicios:

vi. Dada una cadena de caracteres leída a través de teclado, muestra cada una de sus palabras en una línea diferente de pantalla.

vii. Realiza un programa que lea una cadena y diga cuántas vocales hay.

viii. Realiza un programa que lea una frase. El programa nos dirá si la frase tiene todas las vocales o no.

ix. Realiza un programa que lea una frase y nos la muestre en mayúsculas y sin espacios.

x. Realiza un programa que lea una frase y nos diga si se trata de un palíndromo:

Ejemplos: DABALE ARROZ A LA ZORRA EL ABAD
LA RUTA NOS APORTO OTRO PASO NATURAL

xi. Realiza un programa que lea un número entero, y nos diga por pantalla si es capicúa o no. (NOTA: convierte el número en String para solucionar el programa)

STRING a un INT `int numEntero = Integer.parseInt(numCadena);`

INT a STRING `String numCadena= String.valueOf(numEntero);`

Arrays of Strings

```
class ArrayDeStrings {
   public static void main(String args[]) {
      String strs[] = {"Esto", "es", "un", "ejemplo."};
      System.out.println("Original: ");
      for (int i=0; i<strs.length; i++) {
         System.out.print(strs[i] + " ");
      }
      System.out.println("\n");
      // Cambiar valore
      strs[1] = "era";
      strs[3] = "ejemplo, también!";
      System.out.println("Modified array: ");
      for (String s : strs) {
         System.out.print(s + " ");
      }
   }
}
```

Strings Are Immutable

El contenido de un objeto String es inmutable. Es decir, una vez creada, la secuencia de caracteres que conforma la cadena no puede modificarse. Esta restricción permite a Java implementar cadenas de manera más eficiente. Cuando necesite una cadena que sea una variación de una que ya existe, simplemente cree una cadena nueva que contenga los cambios deseados.

Using a String to Control a switch Statement

```
public class SwitchConStrings {
   public static void main(String args[]) {
      String command = "cancel";
      switch (command) {
```

```
            case "connect":
               System.out.println("Connecting");
               break;
            case "cancel":
               System.out.println("Canceling");
               break;
            case "disconnect":
               System.out.println("Disconnecting");
               break;
            default:
               System.out.println("Command Error!");
               break;
         }
      }
   }
```

Using Command-Line Arguments

Ahora que conoce la clase String, puede comprender el parámetro args a main () que ha estado en todos los programas mostrados hasta ahora. Muchos programas aceptan lo que se llaman argumentos de línea de comando. Un argumento de línea de comando es la información que sigue directamente al nombre del programa en la línea de comando cuando se ejecuta. Para acceder a los argumentos de la línea de comandos dentro de un programa Java es bastante fácil: se almacenan como cadenas en la matriz de cadenas que se pasa a main (). Por ejemplo, el siguiente programa muestra todos los argumentos de la línea de comandos con los que se llama:

```
public class StringArgs {
   public static void main(String args[]) {
      System.out.println("There are " + args.length
            + " command-line arguments.");
      System.out.println("They are: ");
      for (int i = 0; i < args.length; i++) {
         System.out.println("arg[" + i + "]: " + args[i]);
      }
   }
}
```

Ejercicio:

xii. Realiza un programa que sume todos los números pasados por la línea de comando:

```
>java suma  4  67  9  2
>La suma es 82
```

sacarlo del paquete comentando la linea

Ejercicio final: Desarrolla el juego del ahorcado (tienes 6 jugadas para acertar , no hay que dibujar

nada,solo ir restando jugadas)
Crea un array de palabras y el jugador jugará contra la maquina.