Capítulo 3

Arreglos y Cadenas de Carácteres

Arreglos en Java

Los arreglos en Java son objetos especiales. Cuando se crea un arreglo lo que internamente realiza Java es crear un objeto de tipo arreglo. Se dice que un arreglo es un objeto especial dado que el programador nunca puede usar directamente una clase Arreglo (no hacemos un "**new"** de alguna clase tipo Arreglo), tampoco podemos crear una clase que herede de la clase Arreglo que internamente implementa Java.

Los arreglos en Java son estáticos en el sentido de qué una vez creados éstos, no pueden redimensionarse. Un objeto arreglo, una vez creado, conserva su tamaño durante todo su tiempo de vida.

Arreglos Unidimensionales

El formato de declaración de un arreglo es:

```
<tipo de los elemento> <nombre del arreglo> [ ];
```

Por ejemplo, para declarar una referencia a un arreglo de enteros podríamos utilizar:

```
int ArregloEnteros[ ];
```

ArregloEnteros es el nombre de una referencia que puede ser utilizada para referenciar a cualquier objeto arreglo de enteros. La referencia ArregloEnteros se declara pero aún no se inicializa. Como todo objeto, una referencia no inicializada tiene el valor null.

Para inicializar un arreglo se usa el operador **new**, indicando entre corchetes el tamaño del arreglo. Por ejemplo, para referenciar nuestra variable anterior ArregloEnteros a un objeto arreglo de 10 enteros se puede utilizar:

```
ArregloEnteros = new int [ 10 ];
```

La sentencia anterior crea un objeto arreglo de enteros y asigna a ArregloEnteros su referencia. Otra forma de hacer lo mismo sería:

```
int ArregloEnteros[ ] = new int [ 10 ];
```

Al igual que C++, un arreglo puede inicializarse utilizando un inicializador, por ejemplo:

```
int ArregloEnteros[ ] = { 10, 5, 2, 3 };
```

Donde el tamaño del arreglo creado y asignado corresponde al número de elementos en el inicializador.

Cuando se crea un objeto arreglo (con el operador **new**), los elementos del arreglo son inicializados según las mismas reglas de inicialización automática:

- Datos primitivos numéricos son inicializados en cero (0).
- Datos primitivos booleanos son inicializados en false.
- Datos tipo referencia son inicializados en null.

En Java, una referencia a un arreglo no puede declarase e inicializarse al estilo de C/C++. Por ejemplo, la siguiente sentencia arrojaría un error de compilación:

```
int ArregloEnteros[ 10 ];
```

Como en cualquier declaración de variables, puede inicializarse más de una referencia a variables de tipo arreglo durante su declaración:

```
int Arreglo1[ ] = new int [ 10 ],
    Arreglo2[ ] = { 10, 5, 2, 3 };
```

Al igual que el resto de datos miembro de una clase, los datos miembros que son referencias a objetos de tipo arreglo también pueden declarase e inicializarse a la vez. Por ejemplo, el siguiente código es correcto:

```
class MiClaseConArreglos {
       …
       double Arreglo[ ] = new double [ 205 ];
       …
}
```

Arreglos Multidimensionales

Java no soporta originalmente la declaración directa de arreglos de múltiples dimensiones. En contraparte, se pueden crear arreglos de arreglos. Por ejemplo, para crear un arreglo bidimensional de enteros usaríamos:

```
int b [ ] [ ];
b = new int [ 10 ] [ ];
b[ 0 ] = new int [ 5 ];
b[ 1 ] = new int [ 5 ];
```

En el ejemplo anterior se declara un arreglo bidimensional de 10x5. También podemos utilizar un inicializador:

```
int b [ ] [ ] = { { 1, 2, 3 },  { 4, 7, 10, 6 } };
```

Donde tendríamos un arreglo de 2 elementos en el cual:

- b[0] refiere a un arreglo de 3 elementos
- b[1] refiere a un arreglo de 4 elementos

Nótese que, dado que se crea realmente un arreglo de arreglos, cada elemento de la primera dimensión puede estar refiriendo a un arreglo de dimensiones distintas.

Si lo que deseamos crear es un arreglo bidimensional, por ejemplo “b”, donde cada elemento de la primera dimensión de “b” sea un arreglo con el mismo tamaño, podemos usar la sintaxis:

```
int b[ ] [ ] = new int [ 5 ] [ 3 ];
```

Donde crearíamos un arreglo bidimensional de 5x3.

Acceso a los Elementos de un Arreglo

La primera posición de un arreglo, al igual que C++, es cero. Los índices que se utilizan para referirse a los elementos de un arreglo deben ser valores enteros o expresiones que, al ser evaluadas, produzcan un valor entero.

Dado que los arreglos son objetos en Java, cada arreglo creado conoce cuál es su tamaño. El programador puede averiguar el tamaño de un arreglo mediante la variable miembro "**length"**.

Para referirnos a los elementos de un arreglo utilizamos la misma sintaxis que en C++. Por ejemplo:

```
int a[ ] = new int [ 4 ];
int b[ ] [ ] = new int [ 5 ] [ 3 ];
int iValor = 65;
a[ 0 ] = 400;
a[ 1 ] = iValor;
a[ 2 ] = 30;
a[ 3 ] = 555;
b[ 2 ] [ 1 ] = a[ 0 ] + a[ 1 ];
```

Si quisiéramos recorrer los elementos de un arreglo podemos utilizar la variable miembro "**length"** para averiguar su longitud. Por ejemplo:

```
void MostrarArreglo( int a[ ] [ ] ) {
       for( int i = 0; i < a.length; i++ )
              for( int j = 0; j < a[ i ].length; j++ )
                     System.out.println( "Valor = " + a[ i ] [ j ] );
}
```

Como último ejemplo, la creación de un arreglo de objetos String podría ser:

```
String a[ ] = new String [ 2 ];
a[ 0 ] = "Hola";
a[ 1 ] = "Adiós";
```

Recuerde que, como los elementos en este arreglo son referencias, éstas son inicializadas automáticamente a null cuando se crea el arreglo. Luego, el siguiente código generaría un error en tiempo de ejecución:

```
String a[ ] = new String [ 10 ];
System.out.println( a[ 0 ] );
```

Dado que el elemento de índice cero del arreglo se está utilizando antes de ser inicializado.

Paso de un Arreglo a un Método

Los arreglos, al igual que todos los objetos, son pasados a un método por referencia. Por ejemplo:

```
class MiClaseConArreglos {
       …
       void Metodo1( ) {
              …
              double Arreglo[ ] = new double [ 205 ];
              Metodo2( Arreglo );
              …
       }
       …
```

```
        void Metodo2( double Arr [ ] ) {
                …
        }
        …
}
```

El método Metodo2 recibe un parámetro tipo arreglo en la referencia **Arr**. Tanto **Arreglo** en Metodo1 como **Arr** en Metodo2 son referencias al mismo objeto arreglo, lo cual significa que al modificar los valores de los elementos del arreglo mediante la referencia **Arr** se estaría también modificando los valores a los que refiere **Arreglo**. Sin embargo, si a la variable **Arr** en Metodo2 se le referencia a otro arreglo, la variable **Arreglo** en Metodo1 seguirá referenciando al arreglo original.

Preguntas de Repaso

- Cuando se pasa un arreglo a un método, si se modifica el parámetro correspondiente dentro del método (por ejemplo, se le asigna otro arreglo), ¿se modificará también la referencia que se usó en la llamada al método?

- Cuando asigno a una referencia de un arreglo el valor de otra referencia de otro arreglo, ¿estoy sacando una copia? Si no, ¿cómo se sacaría una copia?

- ¿Cómo se declara un método que debe recibir como parámetro un elemento de un arreglo multidimensional?

- Si tengo un arreglo multidimensional y paso uno de sus elementos a un método, si modifico el parámetro referencia correspondiente, ¿modifico el elemento del arreglo multidimensional también?

Arreglos en C#

Los arreglos en C# son un tipo de clase predefinida especial, dado que su creación y manipulación difiere de las clases estándar. Aunque los programadores tienen acceso a la clase base de todos los arreglos, la clase Array, no puede derivar directamente de ésta, sino indirectamente a través de los formatos de declaración de arreglos.

El siguiente es el formato de declaración de una variable de tipo arreglo unidimensional:

```
[modificadores] <tipo de los elementos> [ ] <nombre de la variable>;
```

A diferencia de C++, los corchetes van entre el tipo de los elementos y el nombre de la variable, no especificándose las dimensiones del arreglo. El siguiente código muestra un ejemplo del uso de un arreglo unidimensional.

```
using System;
class MainClass {
        public static void Main(string[] args) {
                int[] a;
                a = new int[10];
                a = new int[3];
                a[0] = 10;
                a[1] = 20;
                a[2] = 30;
                for(int i = 0; i < a.Length; i++)
                        a[i] += i;
                int suma = 0;
                foreach(int elemento in a)
                        suma += elemento;
                Console.WriteLine("Suma total = " + suma);
```

```
        }
}
```

Nótese que la variable "a" es de tipo referencia, dado que los arreglos son objetos. En la primera asignación, "a" recibe una referencia a un objeto de tipo arreglo de enteros de diez elementos. En la segunda asignación, "a" recibe una referencia a un nuevo objeto arreglo, esta vez de 3 elementos, quedando el primer objeto arreglo sin modificación, aunque dado que ya no es referenciado por ninguna variable será eliminado de memoria por el recolector de basura. Como se ve, los objetos arreglo, una vez creados, no pueden modificar sus dimensiones.

Dado que un arreglo es un objeto, contiene datos miembro y métodos. El dato miembro "**Length"** (más adelante veremos que realmente se trata de una propiedad, un concepto definido en C#) retorna el número de elementos del arreglo.

Los objetos arreglo se crean en el **heap** gestionado, a diferencia de los arreglos de C++, que se crean en el **stack**. En C++, los arreglos en el **heap** se crean mediante punteros.

C# permite manejar arreglos de arreglos, así como arreglos multidimensionales. Para el primer caso la sintaxis es la siguiente:

```
[modificadores] <tipo de los elementos> [ ][ ] …   <nombre de la variable>;
```

El siguiente código muestra un ejemplo del uso de un arreglo tridimensional.

```
using System;
class MainClass {
        public static void Main(string[] args) {
                int[][][] b;
                b = new int[2][][];
                b[0] = new int[3][];
                b[1] = new int[3][];
                b[0][0] = new int[2];
                b[0][1] = new int[3];
                b[0][2] = new int[4];
                b[1][0] = new int[7];
                b[1][1] = new int[8];
                b[1][2] = new int[9];
                for(int i = 0; i < b.Length; i++)
                        for(int j = 0; j < b[i].Length; j++)
                                for(int k = 0; k < b[i][j].Length; k++)
                                        b[i][j][k] += i+j+k;
                int acumulado = 0;
                foreach(int[][] e1 in b)
                        foreach(int[] e2 in e1)
                                foreach(int e3 in e2)
                                        acumulado += e3;
                Console.WriteLine("Acumulado total = " + acumulado);
        }
}
```

El arreglo "b" se dice que es "**ortogonal"**, debido a que no todos los arreglos, dentro de una misma dimensión, tienen el mismo número de elementos. La contraparte es un arreglo **rectangular**. Si lo que deseamos es crear un arreglo de arreglos rectangular debemos asegurarnos que todos los arreglos creados, para una misma dimensión, tengan el mismo número de elementos.

La sintaxis para el caso de un arreglo unidimensional es la siguiente:

```
[modificadores] <tipo de los elementos> [<tantas comas como dimensiones menos 1>]
<nombre de la variable>;
```

El siguiente código muestra un ejemplo del uso de un arreglo multidimensional de tres dimensiones:

```
using System;
class MainClass
{
	public static void Main(string[] args)
	{
		// arreglo tridimensional de enteros
		int[,,] c;
		c = new int[5,10,8];
		for(int i = 0; i < c.GetLength(0); i++)
			for(int j = 0; j < c.GetLength(1); j++)
				for(int k = 0; k < c.GetLength(2); k++)
					c[i,j,k] += i+j+k;
		int resultado = 0;
		foreach(int e in c)
			resultado += e;
		Console.WriteLine("Resultado total = " + resultado);
		Console.WriteLine("Para el arreglo 'c'");
		Console.WriteLine("= " + c.Length);
		Console.WriteLine("= " + c.Rank);
	}
}
```

En el ejemplo, el método "**GetLength"** reemplaza al uso de **Length**, dado que si bien **Length** también se define para arreglos multidimensionales, retorna el total de elementos del arreglo multidimensional sumados los elementos de todas sus dimensiones. Para el ejemplo anterior, **Length** retorna 24. El método **GetLength** recibe como parámetro el número de la dimensión de la que se desea saber su tamaño. Para el ejemplo anterior GetLength(0) retorna 5, GetLength(1) retorna 10, GetLength(2) retorna 8.

Un objeto tipo arreglo de arreglos se diferencia de un arreglo ortogonal en que el primero tiene su memoria dispersa, mientras que el segundo, junta. La figura 3.1 muestra esta diferencia.

La segunda diferencia es que todos los arreglos **multidimensionales** son forzosamente **rectangulares**, debido a que el tamaño de todas las dimensiones debe especificarse al momento de crear el objeto arreglo.

También existe la posibilidad de mezclar ambos tipos de arreglos al declarar una variable. Por otro lado, la clase base Array ofrece métodos estáticos que permiten realizar operaciones comunes sobre arreglos, como son ordenamiento, inversión, etc. Estos aspectos escapan de los alcances de la presente introducción.

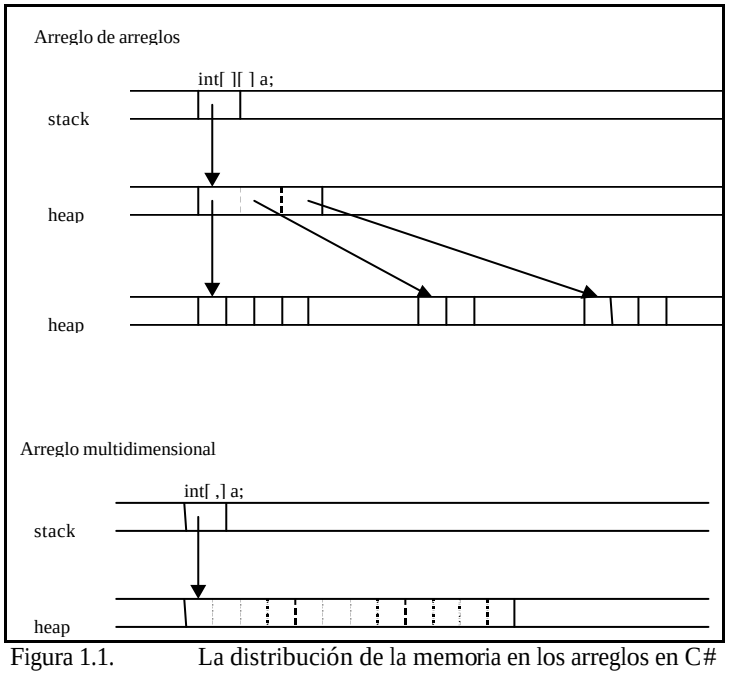

Figura 1.1. La distribución de la memoria en los arreglos en C#

Cadenas de Carácteres en Java

Java tiene una clase especial (como parte del lenguaje, por lo que está definido en el paquete java.lang) para manejar cadenas: "**String"**. Las referencias de tipo **String** pueden concatenarse utilizando el operador **"+"**. La declaración de un literal de texto provoca que Java cree un objeto String sin nombre de forma automática, la que comúnmente suele asignarse a una referencia para su manipulación. Por ejemplo:

```
String Cadena1 = "Hola";
String Cadena2 = "Mundo";
String Cadena3;
Cadena3 = Cadena1 + " " + Cadena2;
System.out.println( Cadena1 );
System.out.println( Cadena2 );
System.out.println( Cadena3 + "!!!" );
System.out.println( Cadena3 );
```

Provocará la salida:

```
Hola
Mundo
Hola Mundo!!!
Hola Mundo
```

La clase **String** provee adicionalmente una serie de constructores para la creación de cadenas.

La concatenación de 2 cadenas provoca la creación de un nuevo objeto **String**, el objeto original no es modificado. Como ejemplo, el siguiente código:

```
String Cadena3;
Cadena3 = "Hola";
Cadena3 += " Mundo";
```

Realizará primera la creación de un objeto **String** que contiene la cadena "Hola", luego se creará un nuevo objeto **String**, en base al primero y a otro objeto **String** " Mundo", y es asignada a la referencia Cadena3. Dado que los objetos **String** "Hola" y " Mundo" ya no son referenciados, en algún momento Java los eliminará de memoria (mediante el sistema de Recolección de Basura que se explicará más adelante).

Se puede acceder a los carácteres individuales de una cadena utilizando el método "**charAt**". El siguiente ejemplo muestra esto:

```
String s = "Hola";
for(int i = 0; i < s.length(); i++)
System.out.println("caracter " + i + " = " + s.charAt(i));
```

Es común que en los programas se realicen conversiones entre cualquier tipo de dato a texto. Para los datos primitivos, esta conversión es automática cuando se concatenan con por lo menos una referencia a alguna cadena. Por ejemplo el siguiente código:

```
int Valor = 10;
String Cadena = "Valor = ";
System.out.println( Cadena + Valor );
String Cadena2;
Cadena2 = "Otra concatenación: " + 50.2;
System.out.println( Cadena2 );
```

Provocará la salida:

```
Valor = 10
Otra concatenación: 50.2
```

Sin embargo, cuando no se realiza una concatenación y se desea convertir un número a una cadena se pueden utilizar las clases especiales de Java para esta labor (también en java.lang), las que encapsulan a los datos primitivos en clases y brindan también métodos estáticos (podemos llamarlos sin necesidad de crear un objeto) que nos facilitan acciones comunes como la conversión desde y hacia cadenas de texto (String). Por ejemplo:

```
int Valor = Integer.parseInt( "10" );
String Cadena = Integer.toString( Valor );
```

La clase **Integer** encapsula un dato primitivo **int** y contiene métodos utilitarios que permiten realizar acciones comunes. Para el resto de datos primitivos existen clases equivalentes, por ejemplo la clase Double, Boolean, etc.

La clase **String** también cuenta con el método estático "**valueOf"** que permite la conversión de cualquier dato de tipo primitivo a una cadena, por ejemplo:

```
String Cadena = String.valueOf( 10 );
```

La clase String está diseñada para ser eficiente en el manejo de textos no modificables. Si se desea manejar un texto grande que se modificará repetidamente, es más eficiente el uso de la clase "**StringBuffer"**. Ésta provee métodos para la modificación de su cadena. El siguiente ejemplo muestra el uso de esta clase.

```
StringBuffer sb = new StringBuffer();
sb.append("hola ");
sb.append("mundo");
System.out.println("sb=" + sb);
```

Como puede verse en el ejemplo, sólo un objeto **StringBuffer** es creado y manipulado, lo que es más eficiente que la creación constante de nuevos objetos cuando se concatenan objetos **String**.

(Falta la comparación con cadenas utilizando equals)

(Falta agregar una seccion sobre los constructores de la clase String)

Cadenas de Carácteres en C#

Al igual que todos los tipos nativos de C#, el tipo "**string"** es un alias del tipo CTS System.String. Un objeto **string** se crea automáticamente cuando se ejecuta alguna sentencia que incluya literales de texto. Por ejemplo, el siguiente código crea un objeto **string** conteniendo la palabra "hola" y se le asigna su referencia a una variable.

```
string cad = "hola";
```

También puede crearse objetos **string** utilizando cualquiera de sus constructores. Algunos de estos son:

```
public string(char[]);// Crea un string en base a un arreglo de carácteres.
public string(char, int);  // Crea un string de n caráteres iguales.
// Crea un string en base a un conjunto de valores de un arreglo de carácteres.
public string(char[], int, int);
```

El siguiente ejemplo utiliza estos constructores:

```
char[] arr = {'H','o','l','a'};
string cad = new string(arr);
Console.WriteLine("cad = " + cad);

string cad2 = "Adios";
Console.WriteLine("cad2 = " + cad2);
```

```
string cad3 = new string('1',5);
Console.WriteLine("cad3 = " + cad3);

cad3 = new string(arr, 1, 2);
Console.WriteLine("cad3 = " + cad3);
```

Una cadena puede ser trabajada como un arreglo, por lo que el siguiente código es válido:

```
string cad = "Como un arreglo";
for(int i = 0; i < cad.Length; i++)
Console.WriteLine("  cad(" + i + ") = " + cad[i]);
```

Una vez creada una cadena, esta no es modificable, por lo que la siguiente instrucción no es válida:

```
cad[i] = 'P';
```

Si bien es de esperarse que la comparación de dos variables tipo referencia corresponda a la comparación de los valores de referencia y no de los contenidos de cada objeto referenciado, la clase **string** sobrecarga el operador de comparación, de manera que lo que se compare sea el contenido. Adicionalmente la clase sobrescribe el método **"Equals"** heredado de **System.Object** (la clase base de todos los tipos de datos) e implementa los métodos "**Compare"** y **"CompareOrdinal"**. El siguiente código muestra algunos casos de comparación entre cadenas utilizando estos métodos.

```
using System;
class Principal {
	public static void Main(string[] args) {
		string cad1 = "Hola Mundo";
		for(int idx = 0; idx < cad1.Length; idx++)
			Console.WriteLine(cad1[idx]);
		string cad2;
		string cad3;
		Console.Write("Primera cadena: ");
		cad2 = Console.ReadLine();
		Console.Write("Segunda cadena: ");
		cad3 = Console.ReadLine();
		if(cad2 == cad3)
			Console.WriteLine("cad2 y cad3 iguales");
		else
			Console.WriteLine("cad2 y cad3 diferentes");
		if(cad2.Equals(cad3))
			Console.WriteLine("cad2 y cad3 iguales");
		else
			Console.WriteLine("cad2 y cad3 diferentes");
		if(String.Compare(cad2, cad3) == 0)
			Console.WriteLine("cad2 y cad3 equivalentes");
		else
			Console.WriteLine("cad2 y cad3 no-equivalentes");
	}
}
```

La clase **string** implementa adicionalmente otros métodos que permiten crear un texto con formato (Format), buscar un carácter o conjunto de carácteres (IndexOf, IndexOfAny, LastIndexOf, LastIndexOfAny), completar una cadena con carácteres de relleno (PadRight, PadLeft), reemplazar carácteres (Replace), partir una cadena en dos cadenas (Split), obtener una sección de una cadena (SubString), colocar los carácteres a mayúsculas o minúsculas (ToLower, ToUpper), eliminar los carácteres de relleno a la izquierda y derecha (Trim).

Debe tenerse siempre en consideración que la clase **string** esta diseñada para ser eficiente en el manejo de cadenas de longitud pequeña y media. Cada método de la clase que produciría una modificación de la cadena original, retorna como resultado una nueva cadena con las modificaciones, dejando la cadena original intacta. Por esto, se dice que los objetos **string** no son modificables luego de su creación. Si se desea manejar textos de gran tamaño, se puede utilizar la clase

```
System.Text.StringBuilder
```

Algunos constructores de esta clase son:

```
StringBuilder( )
StringBuilder( int cap )
StringBuilder( string cad )
StringBuilder( int cap, int capmax )
StringBuilder( string cad, int cap )
```

Algunas propiedades son:

```
Length (para lectura y escritura)
Capacity     (para lectura y escritura)
```

Algunos métodos son:

```
Append, AppendFormat, Insert, Remove, Replace, ToString
```

Es común que en los programas se realicen conversiones entre cualquier tipo de dato a texto. Para los datos primitivos, esta conversión es automática cuando se concatenan con por lo menos una referencia a alguna cadena. Por ejemplo el siguiente código:

```
int Valor = 10;
string Cadena = "Valor = ";
Console.WriteLine( Cadena + Valor );
string Cadena2;
Cadena2 = "Otra concatenación: " + 50.2;
Console.WriteLine( Cadena2 );
```

Provocará la salida:

```
Valor = 10
Otra concatenación: 50.2
```

Sin embargo, cuando no se realiza una concatenación y se desea convertir un número a una cadena se pueden utilizar el método ToString. Dado que en C# todos los tipos de datos son clases, inclusive los tipos primitivos, todos los tipos de datos poseen un método ToString heredado de la clase base object. Por ejemplo:

```
string Cadena = 10.ToString( );
```

Para convertir una cadena a un tipo primitivo se puede utilizar el método Parse incluido en todos los tipos primitivos. Por ejemplo:

```
bool valorBool = bool.Parse("true");
```

También se pueden utilizar los métodos de la clase Convert. Por ejemplo:

```
bool valorBool = System.Convert.ToBoolean(“true”);
```