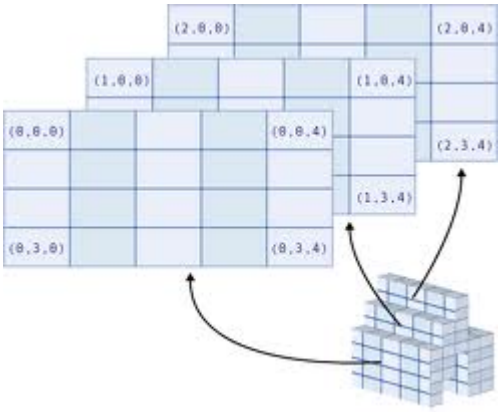
GRUPO STUDIUM FORMACIÓN

Tema 3: Cadenas y Tablas



3.1 Introducción

Como ya hemos visto anteriormente, las variables nos sirven para guardar información de alguno de los tipos como enteros, reales, caracteres, etc. Son muy útiles para guardar un único dato, aunque podemos cambiar su valor tantas veces como queramos, pero sólo mantiene el último valor asignado. Si necesitamos guardar muchos datos a la vez podemos declarar muchas variables lo cual puede ser un poco tedioso y largo.

Para evitar este inconveniente se crearon las **Estructuras Estáticas de Datos**. Son fundamentalmente variables con la diferencia que pueden guardar varios datos a la vez, además con el mismo nombre. Son estáticas porque al crearlas debemos darles un tamaño que ya no podrá ser alterado a lo largo del programa. Aunque en JAVA también podemos crearlos inicialmente sin tamaño y dárselo a la hora de usarlos. Estas estructuras son conocidas como **Arrays** o **Tablas** o **Matrices** o **Vectores**. En realidad se trata del mismo elemento pero que se diferencia por su uso.

En los siguientes apartados veremos cómo trabajar con estos elementos en JAVA y toda la potencia de programación que nos van a proporcionar.

También trataremos con las **Cadenas**, que no son más que Tablas de Caracteres. Pero al estar trabajando con Programación Orientada a Objetos, se han creado Clases para trabajar con todos ellos.

3.2 Cadenas

En JAVA, las cadenas son Objetos. Existen dos tipos de cadenas: **String** y **StringBuffer**. Un String consiste en una tabla o array de caracteres y viene implementada por la clase String:

```
String[] args;
```

Este código declara explícitamente un array, llamado args, que contiene objetos del tipo String. Los corchetes vacíos indican que la longitud del array no se conoce en el momento de la compilación, porque el array se pasa en el momento de la ejecución.

El segundo uso de String es el uso de cadenas literales (una cadena de caracteres entre comillas " y "):

```
"Hola mundo!"
```

El compilador asigna implícitamente espacio para un objeto String cuando encuentra una cadena literal. Los objetos String no se pueden modificar una vez que han sido creados.

Constructores

Como con todas las otras clases, se pueden crear instancias de String con el operador new:

```
String s = new String();
```

Este ejemplo creará una instancia de String sin caracteres en ella. Para crear un String inicializado con caracteres hay que pasarle una matriz de tipo char al constructor. Veamos un ejemplo:

```
char chars[] = {'a','b','c'};
String s = new String(chars); // s es la cadena "abc"
```

Si se tiene una matriz de la que sólo un rango nos interesa existe un constructor que permite especificar el índice de comienzo y el número de caracteres a utilizar:

```
char chars[] = {'a','b','c','d','e','f'};
String s = new String(chars, 2, 3); // s es la cadena "cde"
```

Dado que los Strings son valores constantes, Java incluye un atajo para un literal de cadena estándar, en el que un valor de cadena se puede encerrar entre comillas dobles:

```
String s = "abad";
```

El paquete java.lang proporciona una clase diferente, StringBuffer, que se podrá utilizar para crear cadenas que pueden ser manipuladas después de ser creadas.

Operaciones con Cadenas

Concatenar

Java permite concatenar cadenas fácilmente utilizando el operador +. El siguiente fragmento de código concatena tres cadenas para producir su salida.

```
"La entrada tiene " + contador + " caracteres."
```

Dos de las cadenas concatenadas son cadenas literales: "La entrada tiene " y " caracteres.". La tercera cadena - la de enmedio- es realmente un entero que primero se convierte a cadena y luego se concatena con las otras.

Longitud de una cadena

Uno de los métodos mas habituales que se utilizan en un String es length, que devuelve el número de caracteres de una cadena:

```
String s = "abc";
System.out.println(s.length()); // imprimiría 3
```

Un punto interesante en Java es que se crea una instancia de objeto para cada literal String, por lo que se puede llamar a los métodos directamente con una cadena entre comillas, como si fuera una referencia a objeto, con este ejemplo se volvería a imprimir un 3:

```
String s = "abc";
System.out.println("abc".length());
```

Extracción de caracteres

Para extraer un único carácter de una cadena, se puede referir a un carácter indexado mediante el método charAt:

```
"abc".charAt(1) // devolverá 'b'
```

Si se necesita extraer más de un carácter a la vez, puede utilizar el método getChars, que le permite especificar el índice del primer carácter y del último más uno que se desean copiar, además de la matriz char donde se desean colocar dichos caracteres.

```
String s = "Esto no es una canción";
char buf[] = new char[2];
s.getChars(5, 7, buf, 0); // buf ahora tendrá el valor 'no'
```

También existe una función útil llamada toCharArray, que devuelve una matriz de char que contiene la cadena completa.

Comparación

Si se desean comparar dos cadenas para ver si son iguales, puede utilizar el método equals de String. Devolverá true si el único parámetro está compuesto de los mismos caracteres que el objeto con el que se llama a equals. Una forma alternativa de equals llamada equalsIgnoreCase ignora si los caracteres de las cadenas que se comparan están en mayúsculas o minúsculas.

La clase String ofrece un par de métodos útiles que son versiones especializadas de equals. El método reginMatches se utiliza para comparar una región especifica que se parte de una cadena con otra región de otra cadena. Hay dos variantes de regionMatches, una le permite controlar si es importante la diferenciación entre mayúsculas/minúsculas; la otra asume que si lo es.

```
boolean regionMatches (int toffset, String otra,
           int ooffset, int longitud); // si importa la diferencia
boolean regionMatches (boolean ignorarMaysc,
           int toffset, String otra,
           int ooffset, int longitud); // no importa la diferencia
```

En estas dos versiones de regionMatches, el parámetro toffset indica el desplazamiento en caracteres en el objeto String sobre el que estamos llamando el método. La cadena con la que estamos comparando se llama otra, y el desplazamiento dentro de esa cadena se llama ooffset. Se comparan longitud caracteres de las dos cadenas comenzando a partir de los dos desplazamientos.

Igualdad

El método equals y el operador == hacen dos pruebas completamente diferentes para la igualdad. Mientras que el método equals compara los caracteres contenidos en una String, el operador == compara dos referencias de objeto para ver si se refieren a la misma instancia, es decir, el mismo objeto.

Ordenación

A menudo no basta con conocer si dos cadenas son idénticas o no. Para aplicaciones de ordenación, necesitamos conocer cuál es menor que, igual que o mayor que la siguiente. El método de String compareTo se puede utilizar para determinar la ordenación. Si el resultado entero de compareTo es negativo, la cadena es menor que el parámetro, y si es positivo, la cadena es mayor. Si compareTo devuelve 0, entonces las dos cadenas son iguales. Ahora ordenaremos una matriz de cadenas utilizando compareTo para determinar el criterio de ordenación mediante una Ordenación en burbuja.

```
class SortString
{
      static String arr[]={"Ahora","es","el","momento","de","actuar"};
      public static void main(String args[])
      {
            for (int j = 0; j < arr.length; j++)
            {
                  for (int i = j + 1; i < arr.length; y++)
                  {
                        if (arr[i].compareTo(arr[j] < 0)
                        {
                           String t = arr[j];
                           arr[j] = arr[i];
                           arr[i] = t;
                        }
                  }
                  System.out.println(arr[j]);
            }
      }
}
```

valueOf

Si se tiene algún tipo de dato y se desea imprimir su valor de una forma legible, primero hay que convertirlo a String. El método valueOf está sobrecargado en todos los tipos posibles de Java, por lo que cada tipo se puede convertir correctamente en una String. Cualquier objeto que se le pase a valueOf devolverá el resultado de llamar al método toString del objeto. De hecho, se podría llamar directamente a toString y obtener el mismo resultado.

StringBuffer

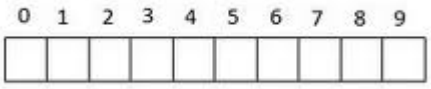
StringBuffer es una clase gemela de String que proporciona gran parte de la funcionalidad de la utilización habitual de las cadenas. StringBuffer representa secuencias de caracteres que se pueden ampliar y modificar. Java utiliza ambas clases con frecuencia, pero muchos programadores sólo tratan con String y permiten que Java manipule StringBuffer por su cuenta mediante el operador sobrecargado '+'.

append

Al método append de StringBuffer se le llama a menudo a través del operador +. Tiene versiones sobrecargadas para todos los tipos. Se llama a String.valueOf para cada parámetro y el resultado se añade al StringBuffer actual. Cada versión de append devuelve el propio buffer.

3.3 Tablas

Como se ha comentado anteriormente, si necesitásemos guardar 50 números enteros, tendría que crear 50 variables de tipo entero. Gracias a las tablas podemos crear una única variable, pero en la que podemos guardar muchos valores. La forma de diferenciar un valor de otro se hará por la posición que ocupan:



En el caso anterior tenemos un array de 10 valores. Las posiciones empiezan en el 0, de manera que el último siempre es uno menos que el tamaño total del array. Para acceder a un valor, debemos indicar el nombre del array y la posición exacta a la que queremos acceder incluido entre corchetes:

```
Tabla[0], Tabla[1], …Tabla[9]
```

Creación

En Java, los Array pueden ser de cualquier tipo de dato, incluidos objetos. El tipo de dato Array es, a su vez, un objeto. Las variables del tipo Array se declaran utilizando [], del siguiente modo:

```
tipo_basico[]  nombre_variable
```

Por ejemplo,

```
int[]  fila;
```

declara la variable *fila* del tipo Array de datos del tipo int. Estas variables almacenarán la referencia al objeto. Para crear el objeto, se utiliza el operador new de la forma:

```
new tipo_basico[numero_de_elementos]
```

Por ejemplo,

```
fila = new int[100];
```

crea el objeto que puede almacenar 100 enteros. Se puede declarar la variable y crear el objeto en la misma instrucción:

```
int[] fila = new int[100];
```

Inicializar los arrays

Para inicializar un array en Java tenemos dos opciones: darle sus valores en la propia creación o bien realizar una recorrido de sus posiciones e ir metiéndole valores como se verá en el siguiente apartado.

Para inicializar en la propia creación le daremos los valores a continuación del igual, dentro de llaves y separados por comas:

```
double[] primos20 = {2.0, 3.0, 5.0, 7.0, 11.0, 13.0, 17.0, 19.0};
```

En este caso se ha creado un array llamado primos20 de tipo double con los 8 primeros números primos. Tendríamos:

```
primos20[0] = 2.0
primos20[1] = 3.0
…
primos20[7] = 19.0
```

Recorrer un array

Pero no siempre sabemos los valores que vamos a meter en el array o se tienen que pedir por teclado. En dicho caso debemos recorrer la tabla. Para ello haremos uso de un bucle for:

```
package es.studium.Tablas1;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Tablas1
{

	public static void main(String[] args) throws IOException
	{
		int tabla[] = new int[5];
		int i;
		BufferedReader lectura = new BufferedReader(new
InputStreamReader(System.in));
		// Recorrido para rellenar la tabla
		for(i=0;i<5;i++)
		{
			System.out.println("Ingrese el valor de la posición:
"+i);
			tabla[i] = Integer.parseInt(lectura.readLine());
		}
		// Recorrido para mostrar los valores de la tabla
		for(i=0;i<5;i++)
		{
			System.out.println("Tabla["+i+"]="+tabla[i]);
		}
	}
}
```

En el ejemplo anterior hemos recorrido la tabla una primera vez para introducir sus valores y otra vez para mostrarlos. Si hubiéramos tenido que hacer

una tratamiento de los datos leídos, tendríamos que haber metido otro recorrido (otro bucle for) tras la lectura para tratar cada elemento de la tabla, antes del último recorrido para mostrar los resultados.

En el siguiente ejemplo, leeremos una tabla de 10 elementos, cambiaremos cada uno por su doble, y luego mostraremos la tabla con los resultados ya calculados:

```
package es.studium.Tablas2;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Tablas2
{

	public static void main(String[] args) throws IOException
	{
		int tabla[] = new int[10];
		int i;
		BufferedReader lectura = new BufferedReader(new
InputStreamReader(System.in));
		// Recorrido para rellenar la tabla
		for(i=0;i<10;i++)
		{
			System.out.println("Ingrese el valor de la posición:
"+i);
			tabla[i] = Integer.parseInt(lectura.readLine());
		}
		// Recorrido para tratar los elementos de la tabla
		for(i=0;i<10;i++)
		{
			tabla[i] = tabla[i]*2;
		}

		// Recorrido para mostrar los valores de la tabla
		for(i=0;i<10;i++)
		{
			System.out.println("Tabla["+i+"]="+tabla[i]);
		}
	}
}
```

Arrays multidimensionales

Hasta ahora hemos trabajado con tablas de una única dimensión. Pero también podemos utilizar tablas de más de una dimensión: bidimensionales, tridimensionales,... Habitualmente con las tablas unidimensionales y bidimensionales tendremos suficiente.

La forma de trabajar con ellos es similar teniendo en cuenta que ahora tendremos un bucle for por cada dimensión, anidados unos dentro de otro.

Por ejemplo, para dos dimensiones:

```
package es.studium.Tablas3;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Tablas3
{

	public static void main(String[] args) throws IOException
	{
		int tabla[][] = new int[5][5];
		int i,j;
		BufferedReader lectura = new BufferedReader(new
InputStreamReader(System.in));
		// Recorrido para rellenar la tabla
		for(i=0;i<5;i++)
		{
			for(j=0;j<5;j++)
			{
				System.out.println("Ingrese el valor de la
posición: ("+i+","+j+")");
				tabla[i][j] =
Integer.parseInt(lectura.readLine());
			}
		}
		// Recorrido para mostrar los valores de la tabla
		for(i=0;i<5;i++)
		{
			for(j=0;j<5;j++)
			{

	System.out.println("Tabla["+i+"]["+j+"]="+tabla[i][j]);
			}
		}
	}
}
```

Y así para más dimensiones.

Para finalizar, se resumen las acciones a realizar para trabajar con Tablas o Arrays.

1º Crear el array definiendo tamaño y el tipo de dato a contener

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

2º Inicializar la tabla, bien de forma Automática o bien de forma Manual leyendo los valores por teclado. Para este proceso hay que recorrer todos los elementos.

```
for(i=0;i<10;i++)
{
	System.out.println("Ingrese el valor de la posición: "+i);
	tabla[i] = Integer.parseInt(lectura.readLine());
}
```

3º Trabajar con la tabla, operando con cada elemento. Para este proceso hay que recorrer todos los elementos.

```
for(i=0;i<10;i++)
{
      tabla[i] = …
}
```

4º Por último, mostrar los resultados. Estos pueden ser un único valor, en cuyo caso se mostrará simplemente por pantalla. Pero también se puede mostrar la tabla para ver cómo ha quedado tras las operaciones del paso 3, en cuyo caso también hay que recorrer de nuevo el array mostrando cada valor.

```
for(i=0;i<10;i++)
{
      System.out.println("Tabla["+i+"]="+tabla[i]);
}
```

3.4 Vectores

Un Vector es parecido a una Tabla o Array con la diferencia que los elementos pueden ser de diferentes tipos. Se encuentra en la librería java.util. Los vectores son menos eficientes que las tablas. Los métodos que podemos utilizar (entre otros) son:

- vector.addElement () → Añade un elemento al vector
- vector.elementaAt() → Para recuperar el elemento del vector
- vector.elements() → Indica el número de elementos del vector.

```
import java.util.*;
public class vectores
{
      public static void main(String args[])
      {
            int i;
            Vector v=new Vector();
            for(i=0;i<10;i++)
            {
                  v.addElement(i);
                  System.out.println(v);
            }
      }
}
```

21/11/2015