Entornos de desarrollo
Bloque 2
Tema 3: Manejo de arrays y cadenas

por Nacho Iborra

2.3.1. Utilización de Arrays

Como en muchos otros lenguajes, en Java también se usan arrays para tratar conjuntos de datos cuyo tamaño está prefijado y todos los elementos son del mismo tipo. Para acceder a cada elemento lo hacemos mediante su índice o posición en el array.

2.3.1.1. Declaración de los Arrays

En Java, se declaran de una forma muy similar a otros lenguajes como C#. Sin embargo, Java sólo contempla el uso de dos tipos de arrays:

Arrays Unidimensionales

Se declaran y usan de la misma forma que hacíamos en C#:

```
int[] data = new int[10];  // Array to store 10 integer values
data[0] = 3;
```

Los corchetes se pueden poner también después del nombre de la variable en la declaración:

```
int data[] ...
```

Arrays bidimensionales y multidimensionales

En este caso, Java solo permite arrays multidimensionales o arrays de arrays, es decir, que cada uno de los subarrays puede tener diferente tamaño. Se utilizan igual que hacíamos en C# (y también podemos poner los corchetes después del nombre de la variable)

```
int[][] data2 = new int[5][]; // Array with 5 rows
data2[0] = new int[3];        // Row 1 has 3 columns
data2[1] = new int[10];       // Row 2 has 10 columns
...
data2[0][0] = 14;
```

Si todas las filas del array van a tener el mismo número de columnas, podemos inicializar el array entero con una sola instrucción:

```
int[][] data2 = new int[5][10];  // 5 rows, 10 columns
```

2.3.1.2. Declarar un array con valores iniciales

Un array se puede declarar al mismo tiempo que se le asignan sus valores iniciales, simplemente igualando la declaración con sus valores entre llaves. El número de valores que pongamos entre las llaves será la longitud de dicho array.

```
int[] someData = {10, 11, 12}; // Array size is 3
```

Para los arrays multidimensionales, se especificarán los elementos de cada fila de la misma forma:

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

2.3.1.3. Recorrer arrays

Se puede usar la propiedad `length` para determinar la longitud del array y acotar el bucle para su recorrido.

```
for (int i = 0; i < data.length; i++)
        ...

for (int i = 0; i < data2.length; i++)
        for (int j = 0; j < data2[i].length; j++)
                ...
```

Ejercicios propuestos:

2.3.1.1. Crea un programa llamado *MatrixAddition* que pida al usuario que introduzca 2 matrices bidimensionales de 3x3 y muestre el resultado de sumarlas. Recordad que para sumar dos matrices se deben sumar sus celdas una a una:

```
result[i][j] = matrixA[i][j] + matrixB[i][j]
```

2.3.1.2. Crea un programa llamado *MarkCount* que pida al usuario que introduzca 10 notas (enteros entre 0 y 10). El programa deberá mostrar cuantas notas de cada tipo se han introducido. Por ejemplo si se introducen estas notas: 1, 7, 5, 7, 2, 6, 7, 3, 5, 8, entonces el programa deberá mostrar:

```
Notas por categoría:
1: 1 notas
```

```
2: 1 notas
3: 1 notas
4: 0 notas
5: 2 notas
6: 1 notas
7: 3 notas
8: 1 notas
9: 0 notas
10: 0 notas
```

2.3.2. Manipulación de cadenas

Para trabajar con cadenas en Java, utilizamos la clase `String` que proporciona bastantes métodos muy útiles que podemos usar (convertir de mayúsculas a minúsculas, obtener una subcadena, encontrar un texto...). Puedes consultar la lista completa de métodos disponibles de String en la documentación oficial. Veamos algunos de ellos:

2.3.2.1. Crear y explorar cadenas

Se puede crear una cadena de diferentes maneras: con un valor constante, preguntando al usuario...

```
String text = "Hello world";
String name = scanner.nextLine();
```

También se pueden concatenar cadenas con el operador `+` o en algunos casos con `+=` (si queremos añadir una cadena al final de otra).

```
text = text + ", how are you?";
```

No se puede tratar una cadena como un array de char (como en C++ o C#) y obtener cada caracter mediante un índice entre corchetes. Para obtener el carácter de una determinada posición necesitaremos usar el método `charAt`. También podemos saber la longitud de una cadena con el método `length`.

```
for (int i = 0; i < text.length(); i++)
    System.out.println(text.charAt(i));
```

2.3.2.2. Comparar cadenas

Se pueden comparar dos cadenas de difirentes maneras:

- Si se quiere saber cual es mayor o menor podemos usar el método `compareTo`. Devuelve un número negativo si la cadena de la izquierda es menor, 0 si son iguales y un número positivo si la cadena de la derecha es mayor.

```

```

```
if (text1.compareTo(text2) < 0)
        System.out.println("Second text is greater");
```

- Si se quiere saber si dos cadenas son iguales, usaremos el método `equals` (recordad, NO se puede usar el comparardor `==` para esto). También podemos usar el método `equalsIgnoreCase` para no tener en cuenta las minúsculas y las mayúsculas en la comparación.

```
if (text1.equals(text2))
        System.out.println("Texts are equal");

if ("hello".equalsIgnoreCase("HELLO"))
    System.out.println("Text are equal ignoring cases");
```

2.3.2.3. Encontrar textos en cadenas

Se pueden encontrar textos en una cadena de diferentes maneras:

- Si solo queremos saber si una cadena contiene un texto concreto, se puede usar el método `contains`, que devolverá un booleano:

```
if (text.contains("hello"))
        System.out.println("There is a 'hello' in the text");
```

- Si se quiere saber en qué posición se encuentra el texto en caso de aparecer, se puede usar `indexOf` (devuelve la primera aparición del texto en la cadena o -1 en caso de no estar contenido) o `lastIndexOf` (devuelve la última aparición del texto buscado o -1 en caso de no aparecer).

```
int pos = text.indexOf("hello");
if (pos > 0)
        System.out.println("There is a 'hello' at position" + pos);
```

- Si se quiere saber si un texto comienza con un determinado prefijo o termina con un sufijo concreto, se pueden usar los métodos `startsWith` or `endsWith` que devuelven un booleano:

```
if (text.startsWith("Hello"))
        System.out.println("Text starts with 'Hello'");
```

2.3.2.4. Conversión de cadenas

Se puede convertir una cadena completa a mayúsculas o minúsculas con los métodos `toUpperCase` y `toLowerCase`:

```

```

```
String text = "Hello world";
String textUpper = text.toUpperCase(); // "HELLO WORLD"
```

Se puede obtener una subcadena de una cadena dada con el método `substring`. Tiene 2 parámetros: el índice desde el que comienza la subcadena (comenzando por 0) y el índice de final de la subcadena (este último carácter no incluido). Si no se pone el segundo parámetro devolverá la subcadena desde el índice inicial hasta el final de la cadena. Por ejemplo: `"Welcome".substring(3, 5)` devuelve "co" (índices 3 y 4 de la cadena).

Se puede reemplazar una subcadena con otra con el método `replace`. Tiene dos parámetros: el antiguo texto y el nuevo texto. Devolverá la cadena resultante.

```
String result = text.replace("Hello", "Good morning");
```

- Hay otras maneras de conseguir esto, tales como el método `replaceAll`, con el que se pueden usar expresiones regulares que coincidan con el texto a reemplazar o `replaceFirst` que solo reemplaza la primera aparición del texto a ser reemplazado.

Se puede dividir una cadena usando un delimitador con el método `split`. Devolverá un array de cadenas con las partes resultantes.

```
String text = "Hello world";
String[] parts = text.split(" "); // Two parts, "Hello" and "world"
```

Ejercicios propuestos:

2.3.2.1. Crea un programa llamado *SortJoin* que pida al usuario que introduzca una lista de nombres separados por espacios en blanco. El programa deberá separar la cadena, ordenar los nombres alfabéticamente y los mostrará separados por comas. Por ejemplo, si el usuario introduce esta lista de nombres: `Susan Kailey William John`, se mostrará `John, Kailey, Susan, William`.

2.3.2.2. Crea un programa llamado *CheckMessages* que pida 10 cadenas al usuario y las guarde en un array. Después de esto, se deberá reemplazar cada aparición de la palabra "Eclipse" por "NetBeans". El programa deberá mostrar las cadenas almacenadas en el array ya actualizadas.

2.3.2.3 Crea un programa llamado *LispChecker*. LISP es un lenguaje de programación donde cada instrucción está dentro de paréntesis. Esta podría ser una instrucción en LISP:

```
(let ((new (x-point a y))))
```

Se debe implementar un programa que coja una cadena con instrucciones LISP (una sola cadena) y comprobar que los paréntesis son correctos (o sea, que el número de paréntesis que se abren sea el mismo que los que se cierran)

2.3.3. Algunos retos más

Ahora que ya sabemos un poco más sobre arrays y strings, intentemos algunos retos más de *Acepta el reto*.

> **Ejercicios propuestos**:
>
> **2.2.3.3.** Intenta resolver estos retos de *Acepta el reto*:
>
> - Los bocadillos de la hormiga reina
> - Nana al bebé de mamá y papá
> - Reloj a través del espejo

2.3.3.1. Retos con entrada infinita

En algunos retos, no hay condición de finalización de petición de datos (como en el Reto 417). En estos casos, debes leer con el objeto `Scanner` mientras haya algo que leer:

```
Scanner sc = new Scanner(System.in);
...
while(sc.hasNextLine())
{
    String text = sc.nextLine();
    ...
}
```