Arrays en Java

Declarar un Array

Para declarar un Array se escribe:

```
clase_o_tipo_de_dato[] nombre_del_Array;
```

Por ejemplo, si quisiera crear un Array unidimensional para guardar los nombres de los días de la semana, haríamos:

```
String [] nombresDiasDeLaSemana;
```

Cada par de corchetes **[]** indica una *dimensión* o *eje* de la estructura. Por eso, para agregar una nueva dimensión lo único que debemos hacer es agregar un nuevo par de corchetes. Generalmente, cada *eje* hace referencia a un *valor agrupador* o *identificador*. Por ejemplo, si quisiera guardar el acumulado de milímetros de lluvia caída durante cada uno de los meses de los últimos tres años, necesitaríamos dos ejes que se referencien a los años y a los meses, por lo que tendríamos dos pares de corchetes:

```
float[][] mmPrecipitaciones;
```

No obstante, en ambos casos sólo tenemos **una única *referencia*** a cada una de las estructuras, pues **en Java los Arrays son objetos**. Java **no crea toda la estructura** en este momento pues *no sabe* cuál será el tamaño que queremos darle. Para eso, debemos ***definirlos***.

Definir un Array

Al definir una Array le damos el tamaño que deseamos. Este será un **tamaño inmodificable** pues, como en muchos otros lenguajes, **en Java los Arrays son *estructuras estáticas***.

Existen dos formas de definir un Array:

- Mediante su creación a través del operador ***new***.
- Mediante la declaración directa de sus valores.

Para definir un Array a través del operador ***new*** debemos escribir:

```
clase_o_tipo [] variable = new clase_o_tipo[tamaño_dimensión];
```

El tamaño de la dimensión siempre es un número igual o mayor que 2. Este valor no tiene por qué ser un valor constante; simplemente debe ser un valor conocido al momento de definir el tamaño, y puede provenir de variables, operaciones matemáticas o cualquier

otra instrucción que genere un número, siempre y cuando ese número sea un valor entero positivo.

Por ejemplo, si quisiéramos crear un Array de diez enteros, hacemos:

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

Si la estructura tiene más de una dimensión, cada par de corchetes debe contener su tamaño. Por ejemplo, para el Array de dos dimensiones que definimos antes para acumular las precipitaciones, hacemos:

```
float[][] mmPrecipitaciones = new float[3][12];
```

***TIP**: declarar las dimensiones directamente con el valor numérico no es una buena práctica. Para estos casos aconsejamos usar constantes, que son mucho más claras:*

```
static final int ANIOS = 3;
static final int MESES = 12;
…
…
float[][] mmPrecipitaciones = new float[ANIOS][MESES];
```

Pero hay una forma más de definir un Array, y es a partir de sus propios valores. En este caso, los mismos datos determinan el tamaño de cada dimensión. Para el caso anterior de los días de la semana podemos hacer o siguiente:

```
String [] nombresDiasDeLaSemana = { "Lunes", "Martes", "Miércoles",
"Jueves", "Viernes", "Sábado", "Domingo" };
```

Una estructura bidimensional (por ejemplo un "cuadrado" de 3 por 3 celdas de números) puede definirse así:

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

Para el caso de estructuras de más de una dimensión, no es necesario que las dimensiones "internas" *midan* lo mismo. La siguiente estructura, más que como una *matriz*, puede verse como un *Array de Arrays*:

```
char [][] datos = { { 'A', 'B', 'C', 'D' }, { 'E', 'F' }, { 'G',
'H', 'I', 'J', 'K', 'L' }, { 'M', 'N' } };
```

En este caso, los "grupos" de caracteres que pertenecen a la segunda dimensión miden 4, 2, 6 y 2 "celdas", respectivamente.

Calcular el tamaño de un Array

En Java los Arrays son objetos, y como tales tienen una propiedad llamada *length*, la cual indica el largo de esa dimensión. Así, para las estructuras definidas anteriormente, tenemos:

```
nombresDiasDeLaSemana.length -> 7 (de la posición 0 a la 6)
```

En este caso es simple, porque es una única dimensión.

Para el caso de la estructura definida para acumular las precipitaciones, el tamaño de la dimensión "externa" se obtiene de la misma forma que en el caso del Array unidimensional:

```
mmPrecipitaciones.length -> 3 (de la 0 a la 2)
```

Pero para la primera dimensión debemos recurrir a mirar el "Array interno":

```
mmPrecipitaciones[0].length -> 12 (0 a 11)
```

Ahora, ¿cómo hacemos para averiguar el tamaño de cada *ocurrencia* de la segunda dimensión en un *Array de Arrays*?

En el caso de la estructura de caracteres *datos* definida anteriormente:

```
datos[0].length -> 4 (0 a 3)
datos[1].length -> 2 (0 a 1)
datos[2].length -> 6 (0 a 5)
datos[3].length -> 2 (0 a 1)
```

Valores iniciales de los elementos de un Array

Siendo variables donde guardamos valores, el contenido inicial de cada elemento de un Array coincide con el valor inicial que encontramos al declarar una variable común ("no Array") sin inicializar. Para el caso de los Arrays de tipos de datos nativos, tendrá el valor *por defecto* para ese tipo de dato: 0 para los números, por ejemplo. En el caso de los Arrays de objetos, el valor inicial será ***null***.

Recorrer completo un Array

Para recorrer completo un Array podemos usar el ciclo ***for***. Será necesario un ***for para cada una de las dimensiones***, y sus valores nunca deben superar el *índice* máximo de la dimensión, que es **tamaño - 1**. En caso de querer alcanzar una posición menor a cero ó un valor igual o mayor que el tamaño de la dimensión se dará **un error de ejecución**.

TIP: para *navegar* un Array siempre apoyate en las constantes usadas para crearlo o en el *length* de la dimensión. Así nunca vas a te vas a caer de la estructura.

Veamos, por ejemplo, cómo recorrer el Array de una dimensión de los días de la semana:

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

Al Array bidimensional de precipitaciones lo recorremos con dos for *anidados*. Para la segunda dimensión, y sabiendo que la estructura es "pareja" en cuanto a la cantidad de *celdas* de la segunda dimensión:

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

Debés asegurarte de que los índices de cada ciclo son acordes en rango al tamaño de la dimensión. En el caso anterior estamos recorriendo todos los meses de cada año ordenados por mes y año. Pero para recorrerlo "cruzado" (todos los eneros, todos los febreros, etc.) **invertiremos el orden de los ciclos pero no el de los índices**:

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

Por la misma razón (para no "caernos" de la estructura), para recorrer el Array bidimensional de caracteres que tiene en su segunda dimensión *elementos* de distinto tamaño debemos fijarnos en cada una de ellos para saber hasta cuánto incrementar en cada ciclo el índice secundario:

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

Recorriendo Arrays de objetos

Teniendo en cuenta que el valor inicial de un objeto es un ***null***, y que **no es necesario llenar el Array en orden**, antes de usar un elemento de un Array de objetos debemos chequear que su valor no sea ***null***.

Veamos el siguiente ejemplo: Un consultorio médico da 10 turnos por día, uno cada media hora. No es normal que los turnos se reserven *en orden*, por lo que es posible que

la estructura contenga algo así (cada par valor encerrado entre corchetes es uno de los elementos del Array:

```
[null][Turno][Turno][null][Turno][Turno][Turno][null][Turno][null]
```

Como algunas posiciones (incluso la primera) tienen ***null*** lo correremos chequeando primero el contenido de la celda:

```
for (int i=0; i < turnos.length; i++) {
    if (turnos[i] != null) {
        System.out.println(turnos[i].getDatos());
    }
}
```

Recorridos incompletos y búsquedas

No siempre necesitaremos recorrer la estructura por completo. Muchas veces lo que queremos es encontrar algún elemento en particular o un lugar libre. Para eso usaremos un *while* que controlará dos cosas:

- Si aún podemos seguir buscando (es decir, si no llegamos al final de la estructura).
- Si aún necesitamos hacerlo (si aún no encontramos lo que buscamos).

```
int i = 0;
while (i < estructura.length && estructura[i] != valor_buscado) {
    i++;
}
if (i != estructura.length) {
    System.out.writeln("El valor buscado está en la posición " + i);
} else {
    System.out.writeln("El valor buscado no se encuentra en la
estructura");
}
```

TIP: Si recorrés la estructura al revés (desde la última posición hacia la primera), en caso de que no encuentres el valor buscado el índice siempre quedará en -1.

```
int i = estructura.length - 1;
while (i <= 0 && estructura[i] != valor_buscado) {
    i++;
}
if (i != -1) {
    System.out.writeln("El valor buscado está en la posición " + i);
} else {
    System.out.writeln("El valor buscado no se encuentra en la
estructura");
}
```