INGENIERÍA MECATRÓNICA

DI_CERO

DIEGO CERVANTES RODRÍGUEZ

JAVA 8

APACHE NETBEANS IDE 12.5

Arreglos/Arrays

Contenido

Arrays, Matrices o Arreglos 2

Clase Predefinida Array 4

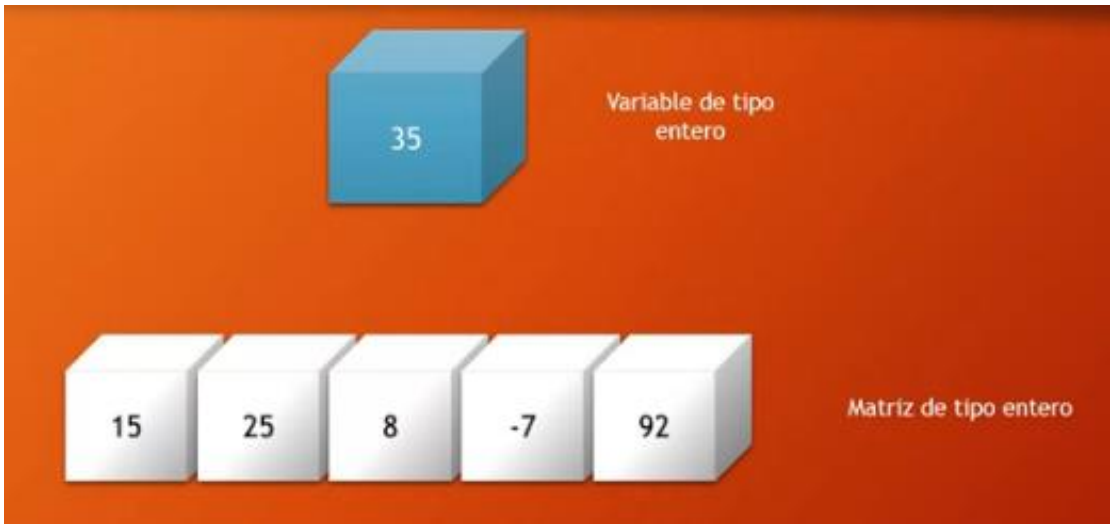
Recorrer un Array 6

Arrays Multidimensionales 7

Referencias: 9

Arrays, Matrices o Arreglos

A los Arrays también se les puede decir matrices, vectores o arreglos y al igual que las variables son un espacio en la memoria de la computadora, pero en ellos se podrán almacenar varios valores del mismo tipo y que tengan alguna relación entre sí en vez de solo uno, estos datos podrán variar durante la ejecución del programa y en ellos se podrán almacenar datos numéricos, booleanos, dates (o fechas), Strings (o cadenas de texto), etc.



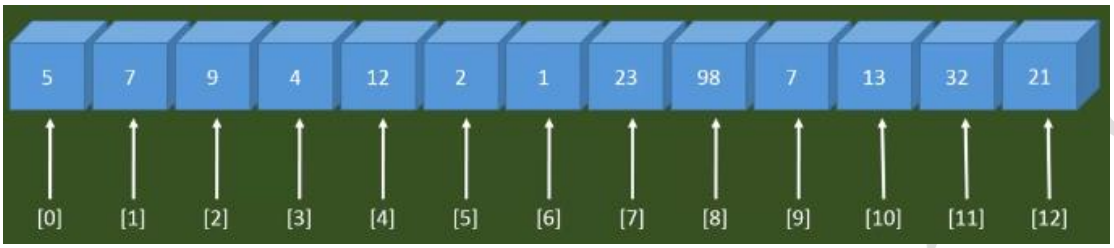
En Java los arrays no son un tipo primitivo, sino que más bien son un objeto de la clase predefinida Array (así como las palabras que son un objeto de la clase predefinida String).

Cada objeto de la clase Array debe ser declarado primero que nada indicando el tipo primitivo de los datos que almacenará en su interior, seguido de unos corchetes que le indicarán al programa que estamos tratando con un arreglo (estos corchetes pueden estar antes o después del nombre del array), luego debo poner su nombre, el signo de igual, la palabra reservada new para que pueda crear un nuevo objeto de la clase predefinida Array y finalmente indicar el tamaño del vector.

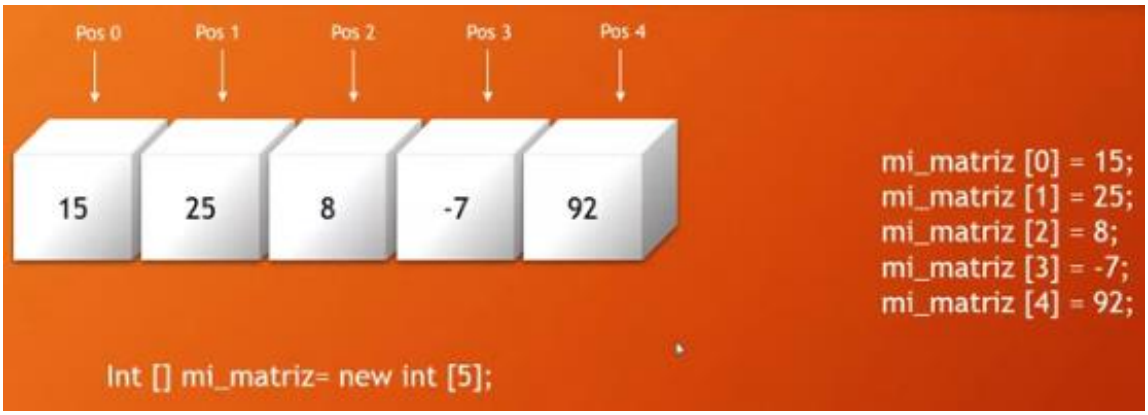
```
tipo_primitivo[] nombre_del_array = new tipo_primitivo[#de_valores_que_almacenará];

tipo_primitivo nombre_del_array[] = new tipo_primitivo[#de_valores_que_almacenará];
```

Al número de valores que almacenará el array se le llama tamaño de la matriz y se cuenta desde el 1, pero cuando queramos acceder a cada una de las posiciones en mi arreglo, deberé contar desde cero el índice de la primera posición.



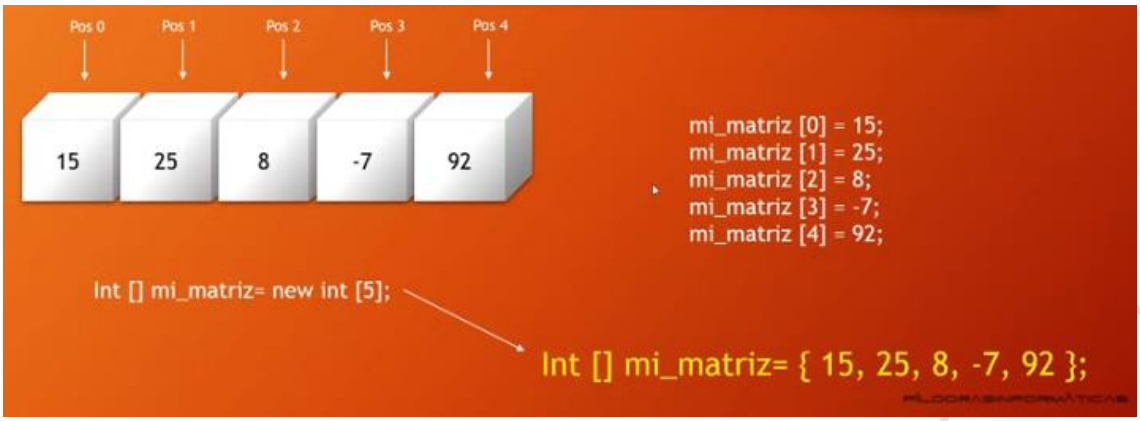
Lo que hicimos anteriormente solo fue declarar el arreglo, para poder darle valor a cada una de estas posiciones debo inicializar cada posición, esto se puede hacer de manera independiente como se ve en la siguiente figura o hacerlo todo de jalón, siempre tomando en cuenta que los índices o posiciones de un array se cuentan desde cero, mientras que el tamaño del array se cuenta desde 1.



Si no quiero declarar primero y luego inicializar cada valor, la forma en la que se puede declarar e inicializar un array en la misma línea es usando llaves de apertura y cierre { } en vez de usar corchetes cuadrados [ ] de la siguiente manera:

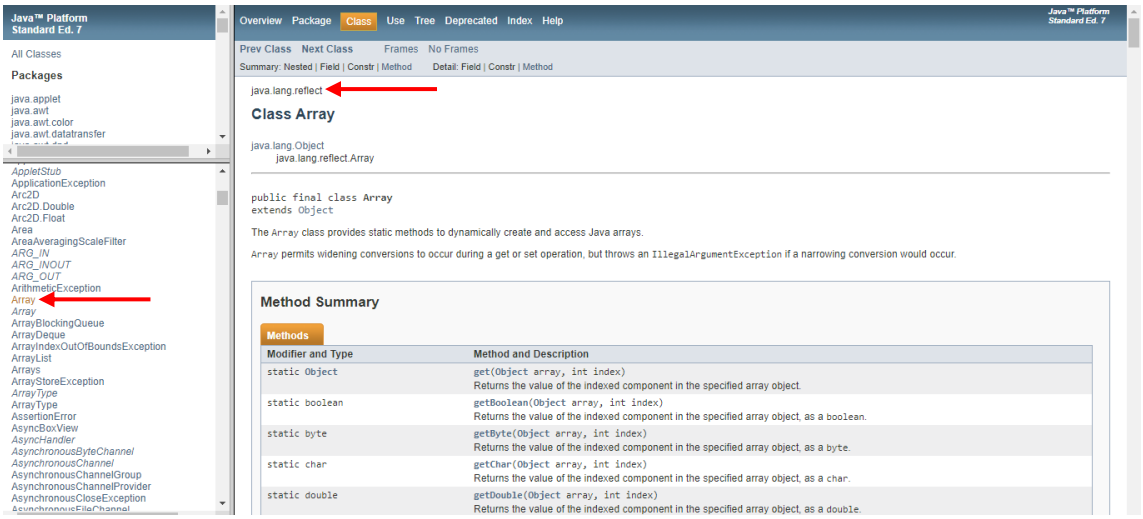
tipo_primitivo[] nombre_del_array = {valor1, valor2, …, valor n};

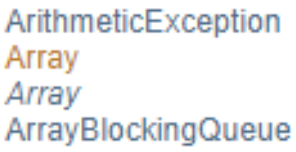
tipo_primitivo nombre_del_array[] = {valor1, valor2, …, valor n};



Clase Predefinida Array

En Java los Arrays no son un tipo de dato, sino un objeto o instancia de la clase predefinida Array, esto se maneja así para que con los distintos métodos de esta clase predefinida pueda manipular o analizar cada array que cree. **Esta clase pertenece al paquete principal java.lang, por lo que no debe ser importado.**

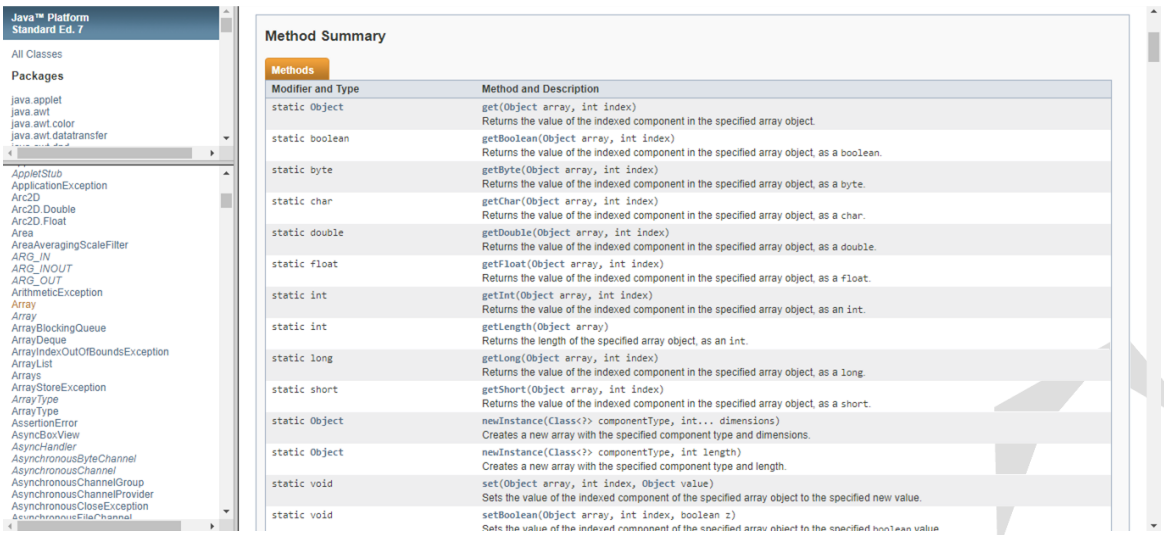




Podemos ver que hay dos elementos con el mismo nombre en la API, esto pasa porque una de ellas no es una clase es una **interfaz (concepto que explicaremos después)** y **podemos distinguir una de otra porque las interfaces siempre están escritas en letra *itálica*.**

CONSTRUCTOR: La clase Array no tiene ningún método constructor.

MÉTODOS: Los métodos son los que puedo usar para la manipulación de arrays dentro del código. Los métodos están enlistados en orden alfabético dentro de la API.



- **.length()**: Con este método obtenemos el número de letras o caracteres que conforman a un objeto String, osea una palabra.

*En la documentación de la API podemos ver que el método no recibe como parámetro nada, pero se debe aplicar a un objeto de la clase String por medio de un punto y regresa un número de tipo int. Es un método **NO** estático.*

La mayoría de los métodos de la clase String NO son estáticos, esto implica que para usarlos debo crear un objeto de la clase String y por medio de este aplicar el método.

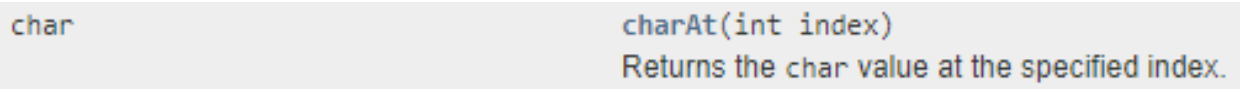
La sintaxis para crear un objeto de la clase String es muy particular ya que no es muy diferente a la forma en la que se declaran variables, se debe poner la palabra reservada String, el nombre de la cadena de caracteres y entre comillas dobles la palabra, se realiza de la siguiente manera:

String nombre_objeto_String = "cadena de caracteres";

Aunque la sintaxis es parecida a la de la declaración e iniciación de variables, a esto se le dice que es un objeto o instancia de la clase String no una variable y se usa para aplicar un método NO estático, esto solo se cumple para la clase String, no para las demás.

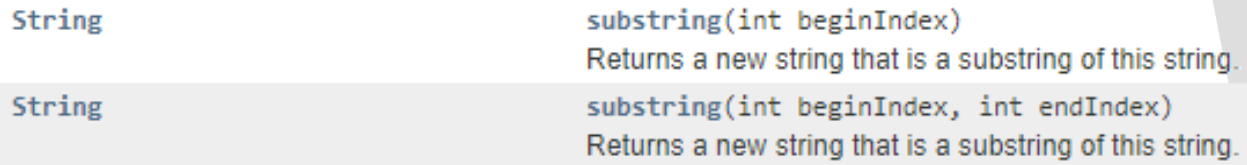
nombre_objeto_String.metodo_no_estático;

- **.charAt(posición)**: Retorna el carácter que se encuentre en la posición indicada, las posiciones en la palabra se empiezan a contar desde cero.



En la documentación de la API podemos ver que el método recibe como parámetro un números de tipo int que le indique al método la ubicación de la letra que quiero extraer del objeto String y regresa un tipo de dato char (caracter o letra), al igual que en el caso anterior, el método se aplica por medio de poner un punto después del nombre del objeto String al que lo quiera aplicar. Es un método NO estático.

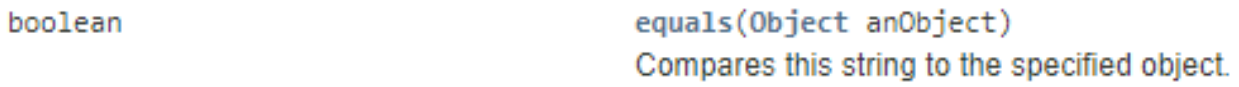
- **.subString(posición1, posición2)**: Se usa para extraer solo una parte de la palabra (objeto String) a la que se le esté aplicando el método, existen dos métodos con el mismo nombre y se aplicará uno u otro dependiendo del número de parámetros que le introduzca. Las posiciones se empiezan a contar desde cero.



En la documentación de la API podemos ver que el método recibe como parámetro la posición de inicio o fin desde donde quiero extraer una parte de esa palabra, ambas posiciones son de tipo int. Si se usa una

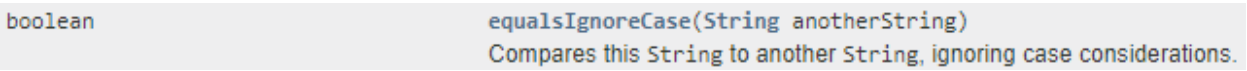
*sola posición de la palabra se extraerá desde ahí hasta el final de la palabra y si se pone el inicio y fin se extraerá solo de ese rango. Ambos son métodos **NO** estáticos.*

- **.equals(objeto_String):** Este método se usa para ver si dos palabras (osea objetos String) son iguales tomando en cuenta mayúsculas y minúsculas.



En la documentación podemos ver que el método recibe como parámetro un objeto de la clase String y regresa un tipo de dato booleano true o false. Es un método NO estático.

- **.equalsIgnoreCase(objeto_String):** Este método se usa para ver si dos palabras (osea objetos String) son iguales sin tomar en cuenta mayúsculas o minúsculas.



En la documentación podemos ver que el método recibe como parámetro un objeto de la clase String y regresa un tipo de dato booleano true o false. Es un método NO estático.

Recorrer un Array

Para recorrer las posiciones de un array y de esta manera poder leerlo o manipularlo usualmente se usa el bucle **for** o un bucle especial hecho para esto llamado bucle **for each**.

- **Bucle for: Este bucle tiene 3 elementos dentro de su paréntesis**: **el número de iniciación del bucle**, **la condición** (que dicta el número de veces que se ejecutará el código) **y el incremento o decremento** (que son operadores matemáticos que dictan de cuánto en cuánto aumentará el número de iniciación hasta llegar al límite dictado por la condición), **todo separado por un punto y coma**.

```
for (iniciación bucle; condición; incremento/decremento) {

    Código a ejecutar mientras el resultado de la condición del for sea true

}
```

Cada vez que el bucle acabe y la condición se siga cumpliendo, el programa llega a la llave de cierre del bucle y después regresa al inicio para aplicarle el incremento o decremento a la variable de iniciación y repetir el ciclo hasta que la condición del bucle se deje de cumplir, osea que el resultado de la operación lógica de la condición sea igual a **false**.

Usualmente la estructura más utilizada para que se ejecute n veces un código es la siguiente:

```
for (i=0; i<n; i++) {

    Código a ejecutar mientras el resultado de la condición del for sea true

}
```

- **Bucle for each: Este bucle está creado específicamente para manejar Arrays,** tiene 2 elementos dentro de su paréntesis y son los siguientes: **una variable de iniciación que sea del mismo tipo que el array que queremos recorrer** y **el nombre de la matriz que quiero recorrer**, **todo separado por dos puntos**.

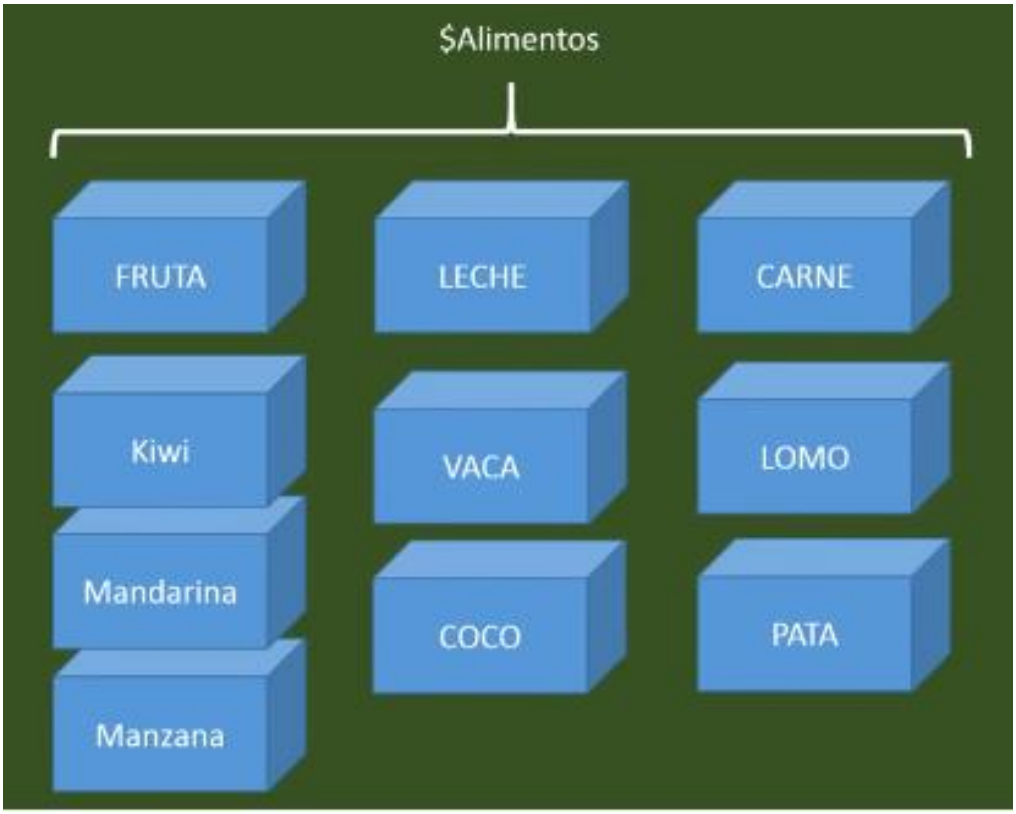
```
for (tipo_primitivo        nombre_variable_de_iniciacion     :     nombre_array) {
    Lo que quiero que le haga a la matriz que recorra true
}
```

Usualmente la estructura más utilizada para imprimir en consola los elementos de un array que tenga almacenadas palabras (osea objetos String) es la siguiente:

```
for (String      nombre_variable     :     nombre_array) {
    Esto imprimirá en consola todos los elementos del array nombre_array de tipo String
    System.out.println(nombre_variable);
}
```

Arrays Multidimensionales

Un Array multidimensional es aquel que en cada posición tiene creado otro array, creando así dos o más dimensiones en cada posición (dependiendo de cuántos arrays existan dentro de cada una), nosotros hasta ahorita hemos hecho arrays de una sola dimensión que almacenan un valor en cada índice. Pero si en vez de haber un valor dentro de cada espacio, hay más espacios que a su vez almacenarán valores cada uno, ahí es donde ya tenemos un array multidimensional.



Cada posición o índice puede tener sus propios valores o sus propios arrays internos, de esta manera se irán aumentando las dimensiones de array, como si fueran una especie de cajones que tienen cajones dentro o como si cada espacio del array fuera una muñeca rusa.

Para declarar un array multidimensional se deben poner más de un par de corchetes cuadrados, de esta manera es como defino la dimensión de la matriz creada y se hace de la siguiente manera donde el número de corchetes dicta el número de dimensiones y en cada uno se debe poner el tamaño de cada dimensión:

```
tipo_primitivo [] [] nombre_del_array = new tipo_primitivo[tamaño_1era_dimensión] [tamaño_2da_dimensión];

tipo_primitivo nombre_del_array [] [] = new tipo_primitivo[tamaño_1era_dimensión] [tamaño_2da_dimensión];
```

Si quiero inicializar individualmente cada valor, tendré que poner el nombre del array, luego indicar su posición dentro del array multidimensional y finalmente darle un valor, a continuación, se muestra un ejemplo:

```
int array2D[][] = new int[4][5];

array2D[0][0] = 15;
array2D[0][1] = 21;
array2D[0][2] = 18;
array2D[0][3] = 9;
array2D[0][4] = 15;

array2D[1][0] = 10;
array2D[1][1] = 52;
array2D[1][2] = 17;
array2D[1][3] = 19;
array2D[1][4] = 7;

array2D[2][0] = 19;
array2D[2][1] = 2;
array2D[2][2] = 19;
array2D[2][3] = 17;
array2D[2][4] = 7;

array2D[3][0] = 92;
array2D[3][1] = 13;
array2D[3][2] = 13;
array2D[3][3] = 32;
array2D[3][4] = 41;
```

Si queremos recorrer una matriz de este tipo tenemos que anidar el mismo número de bucles **for** que el de la dimensión del array, en este caso tendríamos que anidar 2 bucles para que el primer bucle recorra los primeros 4 índices de la 1era dimensión y el segundo recorra los 5 índices de la segunda dimensión, en consola los valores del array se imprimirán en el mismo orden en el que los declaramos en nuestro ejemplo:

```
for (i=0; i<n; i++) {

    for (j=0; j<n; j ++) {

        Con esta forma de declarar los 2 bucles for anidados, se entrará primero a la coordenada de la primera dimensión para recorrer todas las coordenadas de la 2da dimensión y al terminar seguirá con las demás coordenadas de la 1era dimensión para repetir el ciclo

        System.out.println(nombre_array_multidimensional_2D [ i ] [ j ]);

    }

}
```

Referencias:

Pildoras Informáticas, “Curso Java”, 2023 [Online], Available: https://www.youtube.com/playlist?list=PLU8oAlHdN5BktAXdEVCLUYzvDyqRQJ2lk