Material descargable

egg

Teoría **JAVA V**

¡Hola nuevamente! 👋

En esta oportunidad, nos sumergiremos en un emocionante tema de programación: las **matrices** en Java.

Además de los conceptos de arrays y bucles que ya hemos explorado**, nos enfocaremos en cómo declarar matrices, acceder a sus elementos y recorrerlas para realizar operaciones en cada uno de ellos**.

Dominar el uso de matrices nos brindará habilidades prácticas para trabajar con estructuras fundamentales en el desarrollo de aplicaciones. Al comprender cómo utilizarlas eficientemente, podremos resolver problemas más complejos y optimizar nuestros programas.

¡Prepárate para llevar tu dominio de Java al siguiente nivel! 🚀

Matrices

Las matrices son arrays bidimensionales. En contraste con los arrays unidimensionales, que representan una fila de elementos, las matrices son arrays compuestos por otros arrays.

```
public static void main(String[] args) {
    int[] array1 = {1,5,2,3};
}
```

Al declarar una matriz, utilizamos la notación de *"array de arrays"* para indicar que sus elementos son otros arrays.

```
public static void main(String[] args) {
    int[][] matriz1 = { {1,4} , {2,3} , {8,5} };
    int[][] matriz2 = new int[2][3];
}
```

Si nos fijamos detenidamente, **la notación para declarar matrices se interpreta como *"un array que contiene arrays del tipo int"*.** Para crear un array de un tipo específico, como int, agregamos corchetes al tipo original, *"int[]"*. Si deseamos crear un array de ese tipo, le sumamos más corchetes, *"int[][]"*.

En el ejemplo mencionado, la matriz *"matriz1"* es un array de tamaño 3 que contiene arrays de tamaño 2. En el caso de *"matriz2"*, puede resultar menos evidente determinar el tamaño del arreglo que contiene a los demás. La regla a seguir es que el primer corchete hace referencia al array contenedor. Por lo tanto, la forma de *"matriz2"* sería la siguiente:

```
public static void main(String[] args) {
    int[][] matriz2 = { {5,3,2} , {8,1,5} };
}
```

En este caso, "matriz2" es un array de tamaño 2 que contiene arrays de tamaño 3.

💡*Es fundamental comprender esta estructura de las matrices, ya que nos permitirá trabajar con conjuntos de datos más complejos y realizar operaciones eficientes.*

Acceso a los elementos

Para acceder a los elementos de una matriz, seguimos el mismo principio que con los arrays. Veamos un ejemplo:

```
public static void main(String[] args) {
    int[][] matriz = {{5,3,2},{8,1,5}};
    //Accedemos al primer elemento (que es otro array):
    System.out.println(Arrays.toString(matriz[0])); //[5, 3, 2]
    //Accedemos al segundo elemento (que es otro array):
    System.out.println(Arrays.toString(matriz[1])); //[8, 1, 5]
    //Accedemos al segundo elemento del primer array:
    System.out.println(matriz[0][1]); //3
    //Accedemos al primer elemento del segundo array:
    System.out.println(matriz[1][0]); //8
}
```

En el ejemplo, accedemos a los elementos de la matriz utilizando la notación de corchetes. Al especificar matriz[0], estamos accediendo al primer elemento del array contenedor, que, a su vez, es otro array. Al imprimirlo utilizando Arrays.toString(), obtenemos una representación legible de dicho array.

Recorrer una matriz

El recorrido de una matriz sigue una lógica similar al recorrido de un array. Veamos un ejemplo:

```
public static void main(String[] args) {
    int[][] matriz = {{5,3,2},{8,1,5}};
    for (int i = 0; i < matriz.length; i++) {
        System.out.println(matriz[i]);
    }
}
```

En el siguiente ejemplo, al utilizar "matriz[i]" estamos accediendo al primer elemento del array contenedor, que en este caso es un array interno. Sin embargo, al imprimir directamente el array, sólo se mostraría su dirección en memoria. Solucionemos esta situación:

```
public static void main(String[] args) {
    int[][] matriz = {{5,3,2},{8,1,5}};
    for (int i = 0; i < matriz.length; i++) {
        System.out.println(Arrays.toString(matriz));
    }
}
```

Ahora sí, en cada iteración, imprimimos uno de los arrays internos.

Por otro lado, si nuestro objetivo es modificar los valores de los arrays internos, necesitaremos recorrerlos. Para lograrlo, agregaremos otro bucle "for" a nuestro código:

```java
public static void main(String[] args) {
    int[][] matriz = {{5,3,2},{8,1,5}};
    for (int i = 0; i < matriz.length; i++) {
        for (int j = 0; j < matriz[i].length; j++) {
            matriz[i][j] = (int) (Math.random()*(6)+1);
        }
    }
}
```

Observemos cómo el segundo bucle "for" utiliza el atributo "length" de cada array interno, haciendo referencia a "matriz[i]". Es importante detenerse a analizar el funcionamiento de los bucles anidados y notar que el valor de "i" se mantiene constante mientras se ejecutan las iteraciones del bucle "for" interno.

Comprender cómo se realizan las iteraciones en estos bucles será fundamental para los ejercicios futuros.

break y continue

break y continue son dos palabras claves que se pueden usar dentro de los bucles para salir de ellos o saltar a la siguiente iteración.

break: Termina con el bucle sin importar si se cumple con la condición (no ejecuta el resto del código del bucle).

continue: Pasa automáticamente a la siguiente iteración(no ejecuta el resto del código del bucle).

```java
public static void main(String[] args) {
    int[][] matriz = {
        {1, 2, 3},
        {4, 5, 6},
        {0, 8, 9},
        {1, 0, 2},
        {4, 6, 7}
    };
//Programa que imprime sólo los números impares de cada fila hasta encontrar el primer 0.
    for (int i = 0; i < matriz.length; i++) {
        for (int j = 0; j < matriz[i].length; j++) {
            int valor = matriz[i][j];

            // Ejemplo de uso de continue
            if (valor != 0 && valor % 2 == 0) {
                continue; // Salta a la siguiente iteración sin ejecutar el código restante dentro del bucle interno
            }

            // Ejemplo de uso de break
            if (valor == 0) {
                System.out.println("Se encontró el número 0 se detiene el bucle que recorre la fila i="+i);
                break; // Sale del bucle interno y continúa con la siguiente iteración del bucle externo
            }
            System.out.println("posición: ["+i+","+j+"] Valor: "+valor);
        }
    }
}
```

```
CONSOLA:
posición: [0,0] Valor: 1
posición: [0,2] Valor: 3
posición: [1,1] Valor: 5
Se encontró el número 0 se detiene el bucle que recorre la fila i=2
posición: [3,0] Valor: 1
Se encontró el número 0 se detiene el bucle que recorre la fila i=3
posición: [4,2] Valor: 7
```