Arreglos

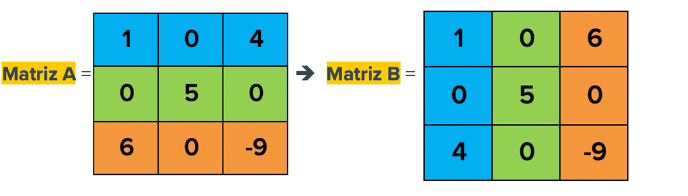
Ejercicios

# Ejercicios de aprendizaje

Continuamos con los ejercicios de aprendizaje y practicamos sobre los arreglos en Java.

|  | **VIDEOS: Te sugerimos ver los videos relacionados con este tema, antes de empezar los ejercicios, los podrás encontrar en tu aula virtual o en nuestro canal de YouTube.** |
| --- | --- |

1. Realizar un algoritmo que llene un vector con los 100 primeros números enteros y los muestre por pantalla en orden descendente.
2. 1.- Realizar un algoritmo que llene un vector con los 100 primeros números enteros y los muestre por pantalla en orden descendente.
3. \*/
4. package numerosenterosprogram;
5. // @author Horacio
6. public class NumerosEnterosProgram {
7. public static void main(String[] args) {
8. int[] numeros = new int[100];  // Creamos un vector de tipo entero con tamaño 100
10. // Llenamos el vector con los 100 primeros números enteros en orden ascendente
11. for (int i = 0; i < 100; i++) {
12. numeros[i] = i + 1;  // Asignamos el valor i + 1 a la posición i del vector
13. }
15. // Mostramos los números enteros en orden descendente
16. System.out.println("Números enteros en orden descendente:");
17. for (int i = numeros.length - 1; i >= 0; i--) {
18. System.out.println(numeros[i]);  // Mostramos el número entero en la posición i del vector
19. }
20. }
21. }
22. /\*
23. En este algoritmo, creamos un vector de tipo entero llamado 'numeros' con tamaño 100.
24. Luego, utilizamos un bucle for para llenar el vector con los 100 primeros números enteros en orden ascendente,
25. asignando el valor de 'i + 1' a la posición 'i' del vector en cada iteración.
26. Finalmente, utilizamos otro bucle for para mostrar los números enteros en orden descendente,
27. comenzando desde la última posición del vector ('numeros.length - 1') y retrocediendo hasta la primera posición ('0').
28. En cada iteración, mostramos el número entero en la posición 'i' del vector por pantalla.
29. \*/
30. Realizar un algoritmo que llene un vector de tamaño N con valores aleatorios y le pida al usuario un número a buscar en el vector. El programa mostrará dónde se encuentra el número y si se encuentra repetido.
31. /\*
32. \* 2-Realizar un algoritmo que llene un vector de tamaño N con valores aleatorios y le pida al usuario un número a buscar en el vector.
33. El programa mostrará dónde se encuentra el número y si se encuentra repetido.
34. \*/
35. package buscarnumeroenvector;
36. // @author Horacio
37. import java.util.Scanner; // Importamos la clase Scanner para leer la entrada del usuario
38. import java.util.Random; // Importamos la clase Random para generar números aleatorios
39. public class BuscarNumeroEnVector {
40. public static void main(String[] args) {
41. Scanner scanner = new Scanner(System.in); // Creamos una instancia de la clase Scanner para leer la entrada del usuario
42. System.out.print("Ingrese el tamaño del vector: ");
43. int n = scanner.nextInt(); // Leemos el tamaño del vector ingresado por el usuario
44. int[] vector = new int[n]; // Creamos un vector de tamaño n
45. Random random = new Random(); // Creamos una instancia de la clase Random para generar números aleatorios
46. System.out.println("El vector generado es: ");
47. for (int i = 0; i < n; i++) {
48. vector[i] = random.nextInt(101); // Generamos un número aleatorio entre 0 y 100 y lo asignamos al vector
49. System.out.print(vector[i] + " "); // Imprimimos el valor del vector en la posición i
50. }
51. System.out.println(); // Imprimimos un salto de línea para separar la salida
52. System.out.print("Ingrese un número a buscar en el vector: ");
53. int numeroBuscado = scanner.nextInt(); // Leemos el número a buscar ingresado por el usuario
54. boolean encontrado = false; // Variable booleana para indicar si se encontró el número buscado
55. int cantidadRepetido = 0; // Variable para contar la cantidad de veces que se repite el número buscado
56. for (int i = 0; i < n; i++) {
57. if (vector[i] == numeroBuscado) { // Comparamos si el número en la posición i del vector es igual al número buscado
58. encontrado = true; // Si se encuentra el número buscado, actualizamos la variable encontrado a true
59. cantidadRepetido++; // Incrementamos la cantidad de veces que se repite el número buscado
60. System.out.println("El número " + numeroBuscado + " se encuentra en la posición " + i + " del vector."); // Mostramos la posición del número encontrado
61. }
62. }
63. if (!encontrado) {
64. System.out.println("El número " + numeroBuscado + " no se encuentra en el vector."); // Si no se encuentra el número buscado, mostramos un mensaje
65. } else {
66. if (cantidadRepetido > 1) {
67. System.out.println("El número " + numeroBuscado + " se encuentra repetido " + cantidadRepetido + " veces en el vector."); // Si el número buscado se encuentra repetido, mostramos la cantidad de veces que se repite
68. } else {
69. System.out.println("El número " + numeroBuscado + " se encuentra repetido una vez en el vector."); // Si el número buscado se encuentra repetido una sola vez, mostramos un mensaje
70. }
71. }
72. scanner.close(); // Cerramos la instancia de la clase Scanner para liberar recursos
73. }
74. }
75. /\*
76. Importación de clases: Se importan dos clases necesarias para el funcionamiento del programa, Scanner y Random, que están en los paquetes java.util.
77. La clase Scanner se utiliza para leer la entrada del usuario, mientras que la clase Random se utiliza para generar números aleatorios.
78. Declaración de variables: Se declaran las variables necesarias para el programa.
79. Esto incluye una instancia de Scanner para leer la entrada del usuario, un arreglo de enteros vector para almacenar los números aleatorios
80. generados, una instancia de Random para generar números aleatorios, y variables para rastrear si se encuentra el número buscado
81. y cuántas veces se repite.
82. Lectura del tamaño del vector: Se solicita al usuario que ingrese el tamaño del vector mediante la clase Scanner
83. y se almacena en la variable n.
84. Generación del vector aleatorio: Se crea un ciclo for que itera n veces para generar números aleatorios entre 0 y 100 (inclusive)
85. utilizando la clase Random, y se asignan a las posiciones correspondientes del arreglo vector.
86. Los números generados se imprimen en la consola.
87. Lectura del número buscado: Se solicita al usuario que ingrese el número que desea buscar en el vector mediante la clase Scanner
88. y se almacena en la variable numeroBuscado.
89. Búsqueda del número buscado en el vector: Se crea otro ciclo for que itera a través del vector "vector" y compara cada elemento
90. con el número buscado utilizando una estructura condicional if.
91. Si se encuentra una coincidencia, se actualiza la variable encontrado a true y se incrementa la
92. variable cantidadRepetido en caso de que el número buscado se repita más de una vez en el vector.
93. Además, se muestra en la consola la posición del número buscado si se encuentra.
94. Resultados de la búsqueda: Se utiliza otra estructura condicional if para mostrar los resultados de la búsqueda.
95. Si el número buscado no se encuentra en el vector, se muestra un mensaje indicando esto.
96. Si se encuentra, se muestra la cantidad de veces que se repite el número buscado en el vector.
97. Cierre del scanner: Finalmente, se cierra la instancia de la clase Scanner utilizando el método close() para liberar recursos.
98. \*/
99. Recorrer un vector de N enteros contabilizando cuántos números son de 1 dígito, cuántos de 2 dígitos, etcétera (hasta 5 dígitos).
100. /\*
101. 3-Recorrer un vector de N enteros contabilizando cuántos números son de 1 dígito, cuántos de 2 dígitos, etcétera (hasta 5 dígitos).
102. \*/
103. package contadordigitos;
104. // @author Horacio
105. import java.util.Scanner;
106. public class ContadorDigitos {
107. public static void main(String[] args) {
108. // Creación de un arreglo de enteros
109. int[] vector;
111. // Declaración de variables
112. int n, num, digito;
113. int[] contador = new int[6]; // El índice 0 no se utiliza
114. try ( // Creación de una instancia de Scanner para leer la entrada del usuario
115. Scanner sc = new Scanner(System.in)) {
116. System.out.print("Ingrese el tamaño del vector: ");
117. n = sc.nextInt(); // Lectura del tamaño del vector
118. vector = new int[n]; // Creación del arreglo con el tamaño ingresado
119. // Lectura de los números enteros del usuario y almacenamiento en el arreglo
120. System.out.println("Ingrese los números enteros: ");
121. for (int i = 0; i < n; i++) {
122. vector[i] = sc.nextInt();
123. }   // Recorrido del vector y conteo de los números de dígitos
124. for (int i = 0; i < n; i++) {
125. num = vector[i];
126. digito = contarDigitos(num);
127. contador[digito]++; // Incremento del contador en la posición correspondiente
128. }   // Impresión de los resultados
129. System.out.println("Resultados:");
130. for (int i = 1; i <= 5; i++) {
131. System.out.println("Cantidad de números con " + i + " dígitos: " + contador[i]);
132. }
133. // Cierre del scanner
134. }
135. }
136. // Método para contar la cantidad de dígitos en un número entero
137. public static int contarDigitos(int num) {
138. int count = 0;
139. while (num != 0) {
140. num /= 10;
141. count++;
142. }
143. return count;
144. }
145. }
146. /\*
147. Se importa la clase Scanner del paquete java.util para leer la entrada del usuario.
148. Se declara un arreglo de enteros llamado vector para almacenar los números ingresados por el usuario.
149. Se declaran variables n para el tamaño del vector, num para almacenar los números del vector durante el recorrido,
150. digito para contar los dígitos de cada número y contador como un arreglo de enteros de tamaño 6 para contar la cantidad de números con 1,
151. 2, 3, 4 o 5 dígitos. El índice 0 del arreglo contador no se utiliza.
152. Se crea una instancia de Scanner llamada sc para leer la entrada del usuario.
153. Se solicita al usuario que ingrese el tamaño del vector.
154. Se crea el arreglo vector con el tamaño ingresado por el usuario.
155. Se solicita al usuario que ingrese los números enteros y se almacenan en el arreglo vector mediante un ciclo for.
156. Se recorre el arreglo vector y se cuenta la cantidad de dígitos de cada número utilizando el método `contarDigit
157. \*/
158. Realizar un programa que rellene un matriz de 4 x 4 de valores aleatorios y muestre la traspuesta de la matriz. La matriz traspuesta de una matriz A se denota por B y se obtiene cambiando sus filas por columnas (o viceversa).



 \*4.-Realizar un programa que rellene un matriz de 4 x 4 de valores aleatorios y muestre la traspuesta de la matriz.

La matriz traspuesta de una matriz A se denota por B y se obtiene cambiando sus filas por columnas (o viceversa).

                 [1] [0] [4]                                                                 [1] [0] [6]

       Matriz A= [0] [5] [0]  ->                                                   Matriz B= [0] [5] [0]

                 [6] [0] [-9]                                                                [4] [0] [-9]

package matriztraspuesta;

// @author Horacio

 import java.util.Random;

public class MatrizTraspuesta {

    public static void main(String[] args) {

        // Declaración y creación de una matriz de 4x4

        int[][] matriz = new int[4][4];

        // Llenado de la matriz con valores aleatorios

        llenarMatriz(matriz);

        // Impresión de la matriz original

        System.out.println("Matriz original:");

        imprimirMatriz(matriz);

        // Cálculo y obtención de la matriz traspuesta

        int[][] matrizTraspuesta = obtenerMatrizTraspuesta(matriz);

        // Impresión de la matriz traspuesta

        System.out.println("\nMatriz traspuesta:");

        imprimirMatriz(matrizTraspuesta);

    }

    // Método para llenar la matriz con valores aleatorios

    public static void llenarMatriz(int[][] matriz) {

        Random random = new Random();

        for (int i = 0; i < matriz.length; i++) {

            for (int j = 0; j < matriz[0].length; j++) {

                matriz[i][j] = random.nextInt(10); // Genera un número aleatorio entre 0 y 9

            }

        }

    }

    // Método para obtener la matriz traspuesta

    public static int[][] obtenerMatrizTraspuesta(int[][] matriz) {

        int filas = matriz.length;

        int columnas = matriz[0].length;

        int[][] matrizTraspuesta = new int[columnas][filas]; // Se intercambian filas por columnas

        for (int i = 0; i < filas; i++) {

            for (int j = 0; j < columnas; j++) {

                matrizTraspuesta[j][i] = matriz[i][j]; // Se asigna el valor de la matriz original a la traspuesta

            }

        }

        return matrizTraspuesta;

    }

    // Método para imprimir una matriz

    public static void imprimirMatriz(int[][] matriz) {

        for (int i = 0; i < matriz.length; i++) {

            for (int j = 0; j < matriz[0].length; j++) {

                System.out.print(matriz[i][j] + " "); // Se imprime cada elemento de la matriz

            }

            System.out.println(); // Salto de línea después de cada fila

        }

    }

}

/\*

Se importa la clase Random del paquete java.util para generar números aleatorios.

Se declara y crea una matriz de 4x4 llamada matriz que almacenará los valores aleatorios.

Se utiliza el método llenarMatriz() para llenar la matriz con valores aleatorios utilizando un ciclo for anidado.

Se imprime la matriz original utilizando el método imprimirMatriz() para mostrar su contenido en la consola.

Se utiliza el método obtenerMatrizTraspuesta() para obtener la matriz traspuesta de matriz.

Se crea una nueva matriz matrizTraspuesta con el tamaño intercambiado de filas por columnas,

es decir, con un tamaño de columnas x filas.

Luego, se utiliza otro par de ciclos for anidados para intercambiar las filas por las columnas de la matriz original

y asignar los valores correspondientes a la matriz traspuesta.

Se retorna la matriz traspuesta obtenida.

Se imprime la matriz traspuesta utilizando el método imprimirMatriz() para mostrar su contenido en la consola.

Los métodos llenarMatriz(), obtenerMatrizTraspuesta() y imprimirMatriz() son métodos auxiliares creados para modularizar el código

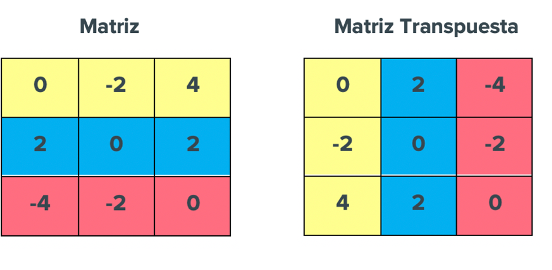
y realizar las tareas específicas de llenado de la matriz, obtención de la matriz traspuesta y la impresión de la matriz en la consola,

respectivamente.

El programa finaliza su ejecución.

\*/

1. Se importa la clase **Random** del paquete **java.util** para generar números aleatorios.
2. Se declara y crea una matriz de 4x4 llamada **matriz** que almacenará los valores aleatorios.
3. Se utiliza el método **llenarMatriz()** para llenar la matriz con valores aleatorios utilizando un ciclo **for** anidado.
4. Se imprime la matriz original utilizando el método **imprimirMatriz()** para mostrar su contenido en la consola.
5. Se utiliza el método **obtenerMatrizTraspuesta()** para obtener la matriz traspuesta de **matriz**. Se crea una nueva matriz **matrizTraspuesta** con el tamaño intercambiado de filas por columnas, es decir, con un tamaño de **columnas** x **filas**. Luego, se utiliza otro par de ciclos **for** anidados para intercambiar las filas por las columnas de la matriz original y asignar los valores correspondientes a la matriz traspuesta.
6. Se retorna la Matriztraspuesta obtenida.
7. Se imprime la matriz traspuesta utilizando el método **imprimirMatriz()** para mostrar su contenido en la consola.
8. Los métodos **llenarMatriz()**, **obtenerMatrizTraspuesta()** y **imprimirMatriz()** son métodos auxiliares creados para modularizar el código y realizar las tareas específicas de llenado de la matriz, obtención de la matriz traspuesta y la impresión de la matriz en la consola, respectivamente.
9. El programa finaliza su ejecución.
10. Realice un programa que compruebe si una matriz dada es antisimétrica. Se dice que una matriz A es antisimétrica cuando ésta es igual a su propia traspuesta, pero cambiada de signo. Es decir, A es antisimétrica si A = -AT. La matriz traspuesta de una matriz A se denota por AT y se obtiene cambiando sus filas por columnas (o viceversa).



**En este caso la matriz es antisimétrica.**

/\*

 \* 5.-Realice un programa que compruebe si una matriz dada es antisimétrica.

       Se dice que una matriz A es antisimétrica cuando ésta es igual a su propia traspuesta, pero cambiada de signo.

       Es decir, A es antisimétrica si A = -AT.

       La matriz traspuesta de una matriz A se denota por AT y se obtiene cambiando sus filas por columnas (o viceversa).

               Matriz                                                             Matriz Transpuesta

              [0] [-2] [4]                                                                 [0]  [2] [-4]

              [2]  [0]  [2]                                                                [-2] [0] [-2]

              [-4] [-2] [0]                                                                [4]  [2] [0]

 \*/

package matrizantisimetrica;

// @author Horacio

public class MatrizAntisimetrica {

    public static void main(String[] args) {

        // Definición de la matriz original

        int[][] matriz = {{0, -2, 4}, {2, 0, 2}, {-4, -2, 0}};

        // Llamada al método para comprobar si la matriz es antisimétrica

        boolean esAntisimetrica = esAntisimetrica(matriz);

        // Impresión del resultado en la consola

        if (esAntisimetrica) {

            System.out.println("La matriz es antisimétrica.");

        } else {

            System.out.println("La matriz NO es antisimétrica.");

        }

    }

    // Método para comprobar si una matriz es antisimétrica

    public static boolean esAntisimetrica(int[][] matriz) {

        // Obtención del número de filas y columnas de la matriz

        int filas = matriz.length;

        int columnas = matriz[0].length;

        // Comprobación de las dimensiones de la matriz (debe ser cuadrada)

        if (filas != columnas) {

            System.out.println("La matriz no es cuadrada, por lo tanto no puede ser antisimétrica.");

            return false;

        }

        // Comparación de la matriz original con su matriz traspuesta cambiada de signo

        for (int i = 0; i < filas; i++) {

            for (int j = 0; j < columnas; j++) {

                if (matriz[i][j] != -matriz[j][i]) {

                    // Si algún elemento no cumple con la condición, la matriz no es antisimétrica

                    return false;

                }

            }

        }

        // Si no se encontraron discrepancias, la matriz es antisimétrica

        return true;

    }

}

/\*

El programa utiliza un método llamado esAntisimetrica() que recibe como parámetro una matriz representada como un arreglo bidimensional

de enteros. A continuación, se realizan los siguientes pasos:

1-Se obtiene el número de filas y columnas de la matriz mediante los métodos length y matriz[0].length, respectivamente.

2-Se verifica si la matriz es cuadrada, es decir, si el número de filas es igual al número de columnas.

Si no lo es, se imprime un mensaje indicando que la matriz no puede ser antisimétrica y se retorna false.

3-Se comparan los elementos de la matriz original con los elementos de su matriz traspuesta cambiada de signo.

Esto se hace utilizando dos ciclos for anidados que recorren la matriz y su matriz traspuesta.

4-Si se encuentra al menos un par de elementos que no cumple con la condición de antisimetría

(es decir, que no son iguales pero con signo cambiado), se retorna false, indicando que la matriz no es antisimétrica.

5-Si no se encontraron discrepancias, es decir, si todos los elementos de la matriz original cumplen con la condición de antisimetría,

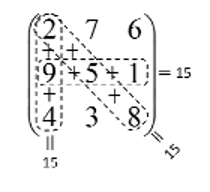
se retorna true, indicando que la matriz es antisimétrica.

\*/

Una matriz antisimétrica es una matriz cuadrada en la cual los elementos de la diagonal principal son todos cero, y los elementos en las posiciones opuestas a la diagonal principal son iguales pero con signo cambiado. Formalmente, una matriz A se considera antisimétrica si y solo si se cumple la siguiente condición: A = -AT, donde AT es la matriz traspuesta de A.

El programa en Java utiliza un enfoque directo para comprobar si una matriz es antisimétrica. Aquí se explica el flujo de trabajo del programa:

1. Se define una matriz original de ejemplo utilizando un arreglo bidimensional **int[][] matriz = {{0, -2, 4}, {2, 0, 2}, {-4, -2, 0}};** con los valores proporcionados en el enunciado.
2. Se llama al método **esAntisimetrica(matriz)** para comprobar si la matriz es antisimétrica.
3. El método **esAntisimetrica()** recibe la matriz como parámetro y realiza los siguientes pasos:
   * Verifica si la matriz es cuadrada, es decir, si el número de filas es igual al número de columnas. Si no lo es, se imprime un mensaje indicando que la matriz no puede ser antisimétrica y se retorna **false**.
   * Compara los elementos de la matriz original con los elementos de su matriz traspuesta cambiada de signo, utilizando dos ciclos **for** anidados para recorrer la matriz y su matriz traspuesta.
   * Si encuentra al menos un par de elementos que no cumple con la condición de antisimetría, es decir, que no son iguales pero con signo cambiado, retorna **false**, indicando que la matriz no es antisimétrica.
   * Si no se encuentran discrepancias, es decir, si todos los elementos de la matriz original cumplen con la condición de antisimetría, retorna **true**, indicando que la matriz es antisimétrica.
4. Finalmente, el resultado de la verificación se imprime en la consola, indicando si la matriz es antisimétrica o no.
5. Un cuadrado mágico 3 x 3 es una matriz 3 x 3 formada por números del 1 al 9 donde la suma de sus filas, sus columnas y sus diagonales son idénticas. Crear un programa que permita introducir un cuadrado por teclado y determine si este cuadrado es mágico o no. El programa deberá comprobar que los números introducidos son correctos, es decir, están entre el 1 y el 9.

****

 6.- Un cuadrado mágico 3 x 3 es una matriz 3 x 3 formada por números del 1 al 9 donde la suma de sus filas,

sus columnas y sus diagonales son idénticas.

Crear un programa que permita introducir un cuadrado por teclado y determine si este cuadrado es mágico o no.

El programa deberá comprobar que los números introducidos son correctos, es decir, están entre el 1 y el 9.

 \*/

package cuadradomagico;

// @author Horacio

import java.util.Scanner;

public class CuadradoMagico {

  public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);

    int[][] cuadrado = new int[3][3]; // Matriz para almacenar el cuadrado mágico

    // Pedir al usuario que introduzca los números del cuadrado mágico

    System.out.println("Introduzca los números del cuadrado mágico (1-9):");

    for (int i = 0; i < 3; i++) { // Iterar por cada fila del cuadrado mágico

      for (int j = 0; j < 3; j++) { // Iterar por cada columna del cuadrado mágico

        cuadrado[i][j] = sc.nextInt(); // Leer el número introducido por el usuario

      }

    }

    // Comprobar que los números introducidos son correctos

    boolean numerosCorrectos = true;

    for (int i = 0; i < 3; i++) {

      for (int j = 0; j < 3; j++) {

        if (cuadrado[i][j] < 1 || cuadrado[i][j] > 9) { // Si un número está fuera del rango válido

          numerosCorrectos = false; // Los números introducidos no son correctos

          break;

        }

      }

    }

    // Si los números son correctos, comprobar si es un cuadrado mágico

    if (numerosCorrectos) {

      int sumaFila1 = cuadrado[0][0] + cuadrado[0][1] + cuadrado[0][2];

      int sumaFila2 = cuadrado[1][0] + cuadrado[1][1] + cuadrado[1][2];

      int sumaFila3 = cuadrado[2][0] + cuadrado[2][1] + cuadrado[2][2];

      int sumaColumna1 = cuadrado[0][0] + cuadrado[1][0] + cuadrado[2][0];

      int sumaColumna2 = cuadrado[0][1] + cuadrado[1][1] + cuadrado[2][1];

      int sumaColumna3 = cuadrado[0][2] + cuadrado[1][2] + cuadrado[2][2];

      int sumaDiagonal1 = cuadrado[0][0] + cuadrado[1][1] + cuadrado[2][2];

      int sumaDiagonal2 = cuadrado[0][2] + cuadrado[1][1] + cuadrado[2][0];

      // Comprobar si las sumas son iguales

      if (sumaFila1 == sumaFila2 && sumaFila1 == sumaFila3 && sumaFila1 == sumaColumna1 && sumaFila1 == sumaColumna2 && sumaFila1 == sumaColumna3 && sumaFila1 == sumaDiagonal1 && sumaFila1 == sumaDiagonal2) {

        System.out.println("¡El cuadrado es mágico!");

      } else {

        System.out.println("El cuadrado no es mágico.");

      }

    } else {

      System.out.println("Los números introducidos no son correctos.");

    }

  }

}

/\*

La primera línea importa la clase Scanner, que permite leer la entrada del usuario desde la consola.

En el método main, se crea un nuevo objeto Scanner y una matriz para almacenar el cuadrado mágico.

Luego se pide al usuario que introduzca los números del cuadrado mágico, y se utiliza un par de bucles for anidados para iterar por cada fila

y columna del cuadrado y leer los números introducidos por el usuario.

Después se comprueba que los números introducidos estén dentro del rango válido (1-9).

Si un número está fuera de este rango, se establece la variable booleana numerosCorrectos en false.

Si los números son correctos, se calculan las sumas de las filas, las columnas y las diagonales del cuadrado mágico y se comprueba

si son iguales. Si las sumas son iguales, el cuadrado es mágico. De lo contrario, no lo es.

Finalmente, se imprime un mensaje indicando si el cuadrado es mágico o no, o si los números introducidos no son correctos.

\*/

La primera línea importa la clase Scanner, que permite leer la entrada del usuario desde la consola. En el método main, se crea un nuevo objeto Scanner y una matriz para almacenar el cuadrado mágico.

Luego se pide al usuario que introduzca los números del cuadrado mágico, y se utiliza un par de bucles for anidados para iterar por cada fila y columna del cuadrado y leer los números introducidos por el usuario.

Después se comprueba que los números introducidos estén dentro del rango válido (1-9). Si un número está fuera de este rango, se establece la variable booleana numerosCorrectos en false.

Si los números son correctos, se calculan las sumas de las filas, las columnas y las diagonales del cuadrado mágico y se comprueba si son iguales. Si las sumas son iguales, el cuadrado es mágico. De lo contrario, no lo es.

Finalmente, se imprime un mensaje indicando si el cuadrado es mágico o no, o si los números introducidos no son correctos.

\*Este ejercicio es un ejemplo de la Guia5 y el enlace lo podes ver en: https://youtu.be/CKSW8ws8fAU

 \* Realizar la multiplicación de un vector por una matriz.

 \*Dado un vector 1xN y una matriz NxM, el resultado del producto es 1xM

 [Recuerda que los vectores se representan por 1fila y N columnas]

Ej vector 1x2 \* matriz 2x3 = producto 1x3

v=|3,5| \* |4,8,6| = |3\*4\*5\*2,3\*8\*5\*1\*7| = |22,29,53|

              |2,1,7|

 \*/

package vectormatrixmultiplication;

// @author Horacio

 import java.util.Scanner;

public class VectorMatrixMultiplication {

    //Scanner leer = new Scanner(System.in);

    public static void main(String[] args) {

//declaración y creación de una vector

//tipo[] nombreVector = new tipo [Tamaño];

//declarar un arreglo de una dimensión

int [] vector;

//Crear en memoria (construir/dimensionar) con el valor por defecto (cero para los enteros)

vector = new int[2];

//los dos pasos en una linea

int [] producto = new int [3];

//Declaración y creación de una matriz.

//tipo [] [] nombreMatriz = new tipo[filas] [columnas]

//Declaración e inicialización con valores determinados

int [] [] matriz = {{4, 8, 6}, {2, 1, 7}};

//Declaración y creación de arreglos genéricos

//tipo [] []......[] nombreArreglo = new tipo[cardinalidad1] [cardinalidad2] [cardinalidad N];

        System.out.println("Ingrese los valores del vector de tamaño " +vector.length+":");//función length devuelve el largo o la cardinalidad que posee el vector. En el caso de la matriz la cardinalidad decada una de las dimensiones.

        Scanner leer = new Scanner(System.in);

        //int = 0, porque los subíndices de los arreglos en Java inician en cero.

        for (int i = 0; i< vector.length; i++) {

            System.out.println("v["+i+"]=");

            //acceder al valor i del vector

            vector [i] = leer.nextInt();

        }

           //Multiplica vector por matriz

           int sum;

           //....para cada columna de la matriz...

           for (int j = 0; j < matriz [0].length; j++) {

                    sum = 0;

                    //..recorro el vector y multiplico

                    for (int i = 0; i < vector.length; i++) {

                          sum += vector [i] \* matriz [i] [j];

                    }

                    producto[j] = sum;

           }

           String aux = "";

           //Mostrar vector

           System.out.println("\* Vector:");

           //bucle for "mejorado" (enhanced)

           //for (tipo elemento : arreglo)

           for (int elemento: vector) {

                 aux = aux + "  " + elemento;

           }

           System.out.println(aux);

           //mostramos matriz

           System.out.println("\n\* Matriz:");

           //para cada fila de la matriz

           for (int [] fila : matriz) {

               aux = "";

               //para cada elemento de la fila

               for (int elemento: fila) {

                         aux += "  " + elemento;

               }

               System.out.println(aux);

           }

           //Mostrar resultado

           aux = "";

           System.out.println("\n\* Vector x Matriz:");

           for (int elemento: producto) {

               aux += "  " + elemento;

           }

           System.out.println(aux);

}

    }

Los Arreglos: Son estructuras que nos permiten guardar en memoria múltiples valores del mismo tipo. Son estructuras potencialmente multidimensionales

Cuando poseen 1 sola Dimensión lo denominamos Vectores

2 Dimensiones lo denominamos Matrices

Pueden tener un montón de otras dimensiones, pero eso se complica representar gráficamente.