# Trabajo 1

## Estructura de datos

Universidad de Cartagena



Carlos Esteban Lara Gil

Prof. John Arrieta

Cartagena de indias 9 de Marzo

Ingeniería de Software sede Piedra Bolívar

[**Trabajo 1**](#_pc6lzrlco4dk) **1**

[Estructura de datos](#_abhlkl7v2duf) 1

[Introducción.](#_bk6tr0enl2kn) 3

[Objetivos](#_1mrberf06u66) 3

[Justificación.](#_3kx2l7ioq93k) 3

[Desarrollo](#_16ig8v4h244m) 4

[1. Declarar y definir arreglos.](#_wl1c5iy0x6ej) 4

[2. Manipular los elementos dentro de arreglos:](#_t0fk35dxnw7) 7

[3. Buscar elementos en un arreglo:](#_va15tlcsdo9m) 9

[4. Ordenar arreglos](#_37sfxw7y0960) 11

[5. Trabajar con Matrices (tablas o arreglos multidimensionales)](#_90h0hzxzaw73) 14

## **Introducción.**

Con este primer trabajo se busca abordar lo principal de las estructuras de datos, lo cuál son arreglos y matrices que le dan sentido a nuestra estructura de datos usando el lenguaje de programación tipado fuerte más popular y estructurado que es Java ya que al ser un lenguaje cuyo paradigma es la programación orientada a objetos la hace más sencillo y legible la estructura de datos.

## **Objetivos**

En este trabajo se abordan: *Las funciones, las matrices y los arreglos*. Se busca que el estudiante alcance la plena comprensión de estos temas ya que es importante para en el futuro poder generar bases de datos estables y fuertes.

Aprender a documentar y llevar un proyecto en el semestre nos ayuda también a que estemos en un ambiente de programación más acercado a lo que es un trabajo en el mundo real con los retos que implica una organización empresarial.

El estudiante será capaz de implementar funciones y arreglos para poder arreglar un código o crearlo acorde a las necesidades del proyecto donde se encuentre a futuro. Por lo tanto, el estudiante será capaz de usar operadores del mundo real para representarlos en el mundo simbólico dando la posibilidad de aplicar el conocimiento adquirido para resolver problemas de programación.

## **Justificación**.

Las matrices y arreglos son fundamentales para la programación y dan la posibilidad de que el estudiante después del trabajo pueda crear aplicaciones robustas y más eficientes a la altura de la ingeniería que aprenden.

La práctica de la programación poniendo manos a la obra permite que por medio de la práctica ejercitar el talento del estudiante mediante el enfoque práctico la educación se vuelve más disciplinada, lo que permite la mejora de las habilidades y la confianza de trabajar en un proyecto grande

## Desarrollo

### Declarar y definir arreglos.

1. Declarar un arreglo por cada uno de los tipos de datos primitivos (byte, Byte, char, Character, short, Short, int, Integer, long, Long, float, Float, double, Double, Object, String.

// Arreglos de tipos primitivos

byte[] bytes;

Byte[] bytesObj;

char[] chars;

Character[] charsObj;

short[] shorts;

Short[] shortsObj;

int[] ints;

Integer[] intsObj;

long[] longs;

Long[] longsObj;

float[] floats;

Float[] floatsObj;

double[] doubles;

Double[] doublesObj;

// Arreglos de tipos Object y String

Object[] objects;

String[] strings;

// Arreglo de Strings con datos específicos

String[] dataStructs = {"Listas", "Colas", "Pilas", "Mapas", "Conjuntos"};

// Arreglo de Strings para características



1. Definir cada uno de los arreglos anteriores con tamaño de 5 elementos.

// Asignar tamaño de 5 a todos los arreglos

bytes = new byte[5];

bytesObj = new Byte[5];

chars = new char[5];

charsObj = new Character[5];

shorts = new short[5];

shortsObj = new Short[5];

ints = new int[5];

intsObj = new Integer[5];

longs = new long[5];

longsObj = new Long[5];

floats = new float[5];

floatsObj = new Float[5];

doubles = new double[5];

doublesObj = new Double[5];

objects = new Object[5];

strings = new String[5];



1. Asignar valores a cada arreglo usando la notación de posiciones o índices

// Asignar valores a algunos elementos de los arreglos

bytes[0] = 1;

bytesObj[1] = 2;

chars[2] = 'a';

charsObj[3] = 'b';

shorts[4] = 100;

shortsObj[0] = 200;

ints[1] = 1000;

intsObj[2] = 2000;

longs[3] = 1000000;

longsObj[4] = 2000000;

floats[0] = 3.14f;

floatsObj[1] = 6.28f;

doubles[2] = 3.1415926535;

doublesObj[3] = 6.2831853072;

objects[4] = new Object();

strings[0] = "Hola mundo";



1. Mostrar los valores de cada Arreglo

// Mostrar algunos valores de los arreglos

System.out.println("bytes[0]: " + bytes[0]);

System.out.println("bytesObj[1]: " + bytesObj[1]);

System.out.println("chars[2]: " + chars[2]);

System.out.println("charsObj[3]: " + charsObj[3]);

System.out.println("shorts[4]: " + shorts[4]);

System.out.println("shortsObj[0]: " + shortsObj[0]);

System.out.println("ints[1]: " + ints[1]);

System.out.println("intsObj[2]: " + intsObj[2]);

System.out.println("longs[3]: " + longs[3]);

System.out.println("longsObj[4]: " + longsObj[4]);

System.out.println("floats[0]: " + floats[0]);

System.out.println("floatsObj[1]: " + floatsObj[1]);

System.out.println("doubles[2]: " + doubles[2]);

System.out.println("doublesObj[3]: " + doublesObj[3]);

System.out.println("objects[4]: " + objects[4]);

System.out.println("strings[0]: " + strings[0]);



1. Declarar y definir otro arreglo de tipo String en una variable llamada dataStructs, e inicializarlo con los siguientes datos, usando la sintaxis de llaves.
   1. Lista
   2. Cola
   3. Pilas
   4. Mapas
   5. Conjuntos

// Arreglo ya definido e inicializado:

String[] dataStructs = {"Listas", "Colas", "Pilas", "Mapas", "Conjuntos"};



1. Declarar y definir otro arreglo de tipo String en una variable llamada características, con 5 elementos.

// Arreglo vacío para características:

String[] caracteristicas= new String[5];



1. Ingresar los siguientes datos por teclado:
   1. Simples, Circulares y Enlazadas
   2. Primero en entrar, Primero en Salir. Último en entrar, Primero en salir
   3. Primero en entrar, Último en salir
   4. Parejas de clave y valor
   5. Elementos no repetidos

// Importar la clase Scanner

import java.util.Scanner;

// Crear un objeto Scanner



### 2. Manipular los elementos dentro de arreglos:

1. Mostrar los elementos del arreglo *dataStructs* con su equivalente elemento del arreglo características.

// Mostrar elementos de dataStructs y características en paralelo

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

System.out.println(dataStructs[i] + ": " + características[i]);

}

/\*

\*Es requerido que el arreglo caracteristicas tenga la misma cantidad que dataStructs para que funcione

\*/



1. Declarar un arreglo de enteros en una variable llamada ventas

// Declarar un arreglo de enteros

int[] ventas;



1. Inicializar el arreglo ventas con una cantidad aleatoria de elementos.

// Importar la clase Random

import java.util.Random;

// Crear un objeto Random

Random random = new Random();

// Generar un número aleatorio para la cantidad de ventas

int numVentas = random.nextInt(100) + 1; // Entre 1 y 100 ventas

// Inicializar el arreglo ventas con el tamaño aleatorio

ventas = new int[numVentas];



1. Llenar el arreglo ventas con valores aleatorios entre 1 y 1000

// Llenar el arreglo ventas con valores aleatorios

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

ventas[i] = random.nextInt(1000) + 1; // Entre 1 y 1000

}



1. Mostrar la cantidad de ventas

// Mostrar la cantidad de ventas

System.out.println("Cantidad de ventas: " + ventas.length);



1. Mostrar el valor de cada una de las ventas

// Mostrar el valor de cada venta

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

System.out.println("Venta " + (i + 1) + ": " + ventas[i]);

}



1. Mostrar el total de ventas

// Calcular y mostrar el total de ventas

int totalVentas = 0;

for (int venta : ventas) {

totalVentas += venta;

}

System.out.println("Total de ventas: " + totalVentas);



1. Mostrar el promedio de las ventas.

// Calcular y mostrar el promedio de las ventas

float promedioVentas = (float) totalVentas / ventas.length;

System.out.println("Promedio de ventas: " + promedioVentas);



### 3. Buscar elementos en un arreglo:

1. Diseñar una función que permita buscar la venta más alta e invocar la función para probarla.

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n;

System.out.println("Ingrese el número de ventas: ");

n = sc.nextInt();

double[] ventas = new double[n];

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

System.out.println("Ingrese la venta " + (i + 1) + ": ");

ventas[i] = sc.nextDouble();

}

System.out.println("La venta más alta es: " + ventaMasAlta(ventas));

}

public static double ventaMasAlta(double[] ventas) {

double mayor = ventas[0];

for (int i = 1; i < ventas.length; i++) {

if (ventas[i] > mayor) {

mayor = ventas[i];

}

}

return mayor;

}

t



1. Diseñar una función que permita buscar la venta más baja e invocar la función para probarla.

public static int obtenerVentaMasBaja(int[] ventas) {

int ventaMasBaja = ventas[0];

for (int i = 1; i < ventas.length; i++) {

if (ventas[i] < ventaMasBaja) {

ventaMasBaja = ventas[i];

}

}

return ventaMasBaja;

}

// Probar la función

int ventaMasBaja = obtenerVentaMasBaja(ventas);

System.out.println("Venta más baja: " + ventaMasBaja);



1. Diseñar una función que permita obtener el total de las ventas e invocar la función para probarla.

public static int obtenerTotalVentas(int[] ventas) {

int totalVentas = 0;

for (int venta : ventas) {

totalVentas += venta;

}

return totalVentas;

}

// Probar la función

int totalVentas = obtenerTotalVentas(ventas);

System.out.println("Total de ventas: " + totalVentas);



1. Diseñar una función que permita obtener el promedio de las ventas e invocar la función para probarla.

public static float obtenerPromedioVentas(int[] ventas) {

int totalVentas = obtenerTotalVentas(ventas);

return (float) totalVentas / ventas.length;

}

// Probar la función

float promedioVentas = obtenerPromedioVentas(ventas);

System.out.println("Promedio de ventas: " + promedioVentas);



1. Diseñar una función que devuelva la venta cuyo valor se acerque más al valor que le sigue al promedio e invocar la función para probarla.

public static int obtenerVentaMasCercanaAlPromedio(int[] ventas) {

float promedioVentas = obtenerPromedioVentas(ventas);

int valorPosteriorPromedio = (int) Math.ceil(promedioVentas);

int ventaMasCercana = ventas[0];

int diferenciaMinima = Math.abs(ventaMasCercana - valorPosteriorPromedio);

for (int i = 1; i < ventas.length; i++) {

int diferenciaActual = Math.abs(ventas[i] - valorPosteriorPromedio);

if (diferenciaActual < diferenciaMinima) {

ventaMasCercana = ventas[i];

diferenciaMinima = diferenciaActual;

}

}

return ventaMasCercana;

}

// Probar la función

int ventaMasCercana = obtenerVentaMasCercanaAlPromedio(ventas);

System.out.println("Venta más cercana al valor posterior al promedio: " + ventaMasCercana);



### 4. Ordenar arreglos

1. Diseñar una función que permita ordenar las ventas de forma descendente e invocar la función para probarla.

public static void ordenarVentasDescendente(int[] ventas) {

for (int i = 0; i < ventas.length - 1; i++) {

for (int j = i + 1; j < ventas.length; j++) {

if (ventas[i] < ventas[j]) {

int temp = ventas[i];

ventas[i] = ventas[j];

ventas[j] = temp;

}

}

}

}

// Probar la función

ordenarVentasDescendente(ventas);

System.out.println("Ventas ordenadas descendentemente:");

for (int venta : ventas) {

System.out.println(venta);

}



1. Diseñar una función que permita ordenamiento que permita ordenar las ventas de forma ascendente e invocar la función para probarla.

public static void ordenarVentasAscendente(int[] ventas) {

for (int i = 0; i < ventas.length - 1; i++) {

for (int j = i + 1; j < ventas.length; j++) {

if (ventas[i] > ventas[j]) {

int temp = ventas[i];

ventas[i] = ventas[j];

ventas[j] = temp;

}

}

}

}

// Probar la función

ordenarVentasAscendente(ventas);

System.out.println("Ventas ordenadas ascendentemente:");

for (int venta : ventas) {

System.out.println(venta);

}



1. Diseñar una función que permite desordenar las ventas e invocar la función para probarla.

public static void desordenarVentas(int[] ventas) {

Random random = new Random();

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

int indiceAleatorio = random.nextInt(ventas.length);

int temp = ventas[i];

ventas[i] = ventas[indiceAleatorio];

ventas[indiceAleatorio] = temp;

}

}

// Probar la función

desordenarVentas(ventas);

System.out.println("Ventas desordenadas:");

for (int venta : ventas) {

System.out.println(venta);

}



1. Diseñar una función que permita ordenar las ventas primero las partes y luego en impares e invocar la función para probarla

public static void ordenarVentasParesImpares(int[] ventas) {

int[] ventasPares = new int[ventas.length / 2];

int[] ventasImpares = new int[ventas.length - ventasPares.length];

int indicePares = 0, indiceImpares = 0;

for (int venta : ventas) {

if (venta % 2 == 0) {

ventasPares[indicePares++] = venta;

} else {

ventasImpares[indiceImpares++] = venta;

}

}

// Ordenar pares e impares

ordenarVentasAscendente(ventasPares);

ordenarVentasAscendente(ventasImpares);

// Combinar ventas

indicePares = 0;

indiceImpares = 0;

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

if (i % 2 == 0) {

ventas[i] = ventasPares[indicePares++];

} else {

ventas[i] = ventasImpares[indiceImpares++];

}

}

}

// Probar la función

ordenarVentasParesImpares(ventas);

System.out.println("Ventas ordenadas pares e impares:");

for (int venta : ventas) {

System.out.println(venta);

}

### 5. Trabajar con Matrices (tablas o arreglos multidimensionales)

1. Declarar e inicializar una matriz de 5 x 8 de tipo Object en una variable llamada lenguajes

// Declarar e inicializar una matriz de 5 x 8 de tipo Object

Object[][] lenguajes = new Object[5][8];

// Asignar valores a la primera línea

lenguajes[0][0] = "NOMBRE";

lenguajes[0][1] = "AÑO";

lenguajes[0][2] = "AUTOR";

lenguajes[0][3] = "DETALLES";

lenguajes[0][4] = "FRAMEWORKS";

lenguajes[0][5] = "EJEMPLOS";

lenguajes[0][6] = "PARADIGMAS";

lenguajes[0][7] = "CATEGORÍA";



1. A la primera línea de la matriz lenguajes se debe asignar los siguientes valores en ese orden: [NOMBRE, AÑO, AUTOR, DETALLES, FRAMEWORKS]

public static void infoLenguajes(Object[][] lenguajes) {

Scanner scanner = new Scanner(System.in);

for (int i = 1; i < lenguajes.length; i++) {

System.out.println("\*\*Lenguaje " + i + "\*\*");

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

System.out.print(lenguajes[0][j] + ": ");

lenguajes[i][j] = scanner.nextLine();

}

}

}

// Probar la función

infoLenguajes(lenguajes);



1. Realizar una función llamada *infoLenguajes* que permita llenar la matriz lenguajes con datos por teclado a partir de la segunda fila.

public static boolean buscarLenguaje(Object[][] lenguajes, String nombre) {

for (int i = 1; i < lenguajes.length; i++) {

if (lenguajes[i][0].equals(nombre)) {

return true;

}

}

return false;

}

// Ejemplo de uso

String nombreLenguaje = "Python";

if (buscarLenguaje(lenguajes, nombreLenguaje)) {

System.out.println("El lenguaje " + nombreLenguaje + " se encuentra en la matriz");

} else {

System.out.println("El lenguaje " + nombreLenguaje + " no se encuentra en la matriz");

}



1. Realizar una función que reciba como parámetro el nombre de un lenguaje y retorna verdadero si este se encuentra dentro de la matriz lenguajes, de lo contrario retorna falso.

public static int[] obtenerCoordenadasLenguaje(Object[][] lenguajes, String nombre) {

for (int i = 1; i < lenguajes.length; i++) {

if (lenguajes[i][0].equals(nombre)) {

return new int[] {i, 0}; // Fila 0 = nombre

}

}

return null;

}

// Ejemplo de uso

String nombreLenguaje = "Java";

int[] coordenadas = obtenerCoordenadasLenguaje(lenguajes, nombreLenguaje);

if (coordenadas != null) {

System.out.println("El lenguaje " + nombreLenguaje + " se encuentra en la fila " + coordenadas[0]);

} else {

System.out.println("El lenguaje " + nombreLenguaje + " no se encuentra en la matriz");

}



1. Realizar una función que reciba como parámetro el nombre de un lenguaje y retorna las coordenadas (fila, columna) si este se encuentra encuentra dentro de la matriz lenguajes, de lo contrario retorna null.

public static Object[] obtenerRegistroFila(Object[][] lenguajes, int fila) throws Exception {

if (fila < 1 || fila >= lenguajes.length) {

throw new Exception("Fila " + fila + " no válida");

}

return lenguajes[fila];

}

// Ejemplo de uso

int fila = 2;

try {

Object[] registro = obtenerRegistroFila(lenguajes, fila);

for (Object dato : registro) {

System.out.print(dato + " ");

}

System.out.println();

} catch (Exception e) {

System.out.println("Error: " + e.getMessage());

}



1. Realizar una función que reciba como parámetro una fila y retorne el registro completo (datos en la fila) del lenguaje en esa fila. Si la fila está errada (no existe en la matriz lenguajes) entonces, debe retornar una excepción con un mensaje informando el problema

public static Object[] obtenerColumna(Object[][] lenguajes, int columna) throws Exception {

if (columna < 0 || columna >= lenguajes[0].length) {

throw new Exception("Columna " + columna + " no válida");

}

Object[] columnaDatos = new Object[lenguajes.length];

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

columna



1. Realizar una función que reciba como parámetro una columna y retorne un todos los datos en esa columna. Si la columna está errada (no existe en la matriz lenguajes) entonces, debe retornar una excepción con un mensaje informando el problema.

public static void main(String[] args) throws Exception {

// Leer datos de la consola

Scanner scanner = new Scanner(System.in);

System.out.println("Introduzca el número de filas:");

int filas = scanner.nextInt();

System.out.println("Introduzca el número de columnas:");

int columnas = scanner.nextInt();

// Crear matriz

Object[][] lenguajes = new Object[filas][columnas];

// Introducir datos en la matriz

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

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

System.out.println("Introduzca el dato para la fila " + (i + 1) + " columna " + (j + 1) + ":");

lenguajes[i][j] = scanner.next();

}

}

// Obtener columna

int columna = 2; // AÑO

Object[] datosColumna = obtenerColumna(lenguajes, columna);

// Imprimir datos de la columna

for (Object dato : datosColumna) {

System.out.print(dato + " ");

}

System.out.println();

}



1. Realizar una función que reciba como parámetro las coordenadas (fila, columna) y retorne el valor en esa coordinada y el registro (fila) completo con ojos datos de ese lenguaje, si las coordinadas son erradas (no existen en la matriz lenguajes) entonces, debe retornar una excepción con un mensaje informando el problema.

public static Object[] obtenerValorRegistro(Object[][] lenguajes, int fila, int columna) throws Exception {

if (fila < 0 || fila >= lenguajes.length) {

throw new Exception("Fila " + fila + " no válida");

}

if (columna < 0 || columna >= lenguajes[0].length) {

throw new Exception("Columna " + columna + " no válida");

}

Object[] registro = new Object[lenguajes[0].length];

for (int i = 0; i < lenguajes[0].length; i++) {

registro[i] = lenguajes[fila][i];

}

return new Object[]{registro[columna], registro};

}

// Ejemplo de uso

try {

int fila = 1; // C++

int columna = 2; // AÑO

Object[] valorRegistro = obtenerValorRegistro(lenguajes, fila, columna);

System.out.println("Valor en la coordenada (" + fila + ", " + columna + "): " + valorRegistro[0]);

System.out.println("Registro completo:");

for (Object dato : (Object[]) valorRegistro[1]) {

System.out.print(dato + " ");

}

System.out.println();

} catch (Exception e) {

System.out.println("Error: " + e.getMessage());

}



1. lenguaje y retorna las coordenadas (fila, columna) si este se encuentra dentro de la matriz lenguajes, de lo contrario retorna null.

public static int[] buscarLenguaje(Object[][] lenguajes, String lenguaje) {

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

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

if (lenguajes[i][0].equals(lenguaje)) {

return new int[]{i, j};

}

}

}

return null;

}

// Ejemplo de uso

String lenguaje = "Python";

int[] coordenadas = buscarLenguaje(lenguajes, lenguaje);

if (coordenadas != null) {

System.out.println("Lenguaje " + lenguaje + " encontrado en la coordenada (" + coordenadas[0] + ", " + coordenadas[1] + ")");

} else {

System.out.println("Lenguaje " + lenguaje + " no encontrado");

}



1. Realizar una función que permita mostrar los datos de la matriz lenguajes según la siguiente logia:
   1. La función de debe recibir como parámetro un arreglo con 5 elementos de tipo boolean
   2. Al invocar la función se debe mostrar solo la información de la matriz lenguajes de acuerdo al valor (false o true) que tenga el arreglo que recibe como parámetro. Por ejemplo:

boolean info[] = {true, false, true, true, false}; infoLenguajes(info)

Debe mostrar los nombres de los lenguajes, los autores, los frameworks

public static void mostrarInformacionLenguajes(boolean[] info) {

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

if (info[i]) {

switch (i) {

case 0:

System.out.println("Nombre:");

break;

case 1:

System.out.println("Año:");

break;

case 2:

System.out.println("Creador:");

break;

case 3:

System.out.println("Paradigma:");

break;

case 4:

System.out.println("Tipado:");

break;

}

for (int j = 0; j < lenguajes.length; j++) {

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

}

}

}

}

// Ejemplo de uso

boolean[] info = {true, false, true, true, false};

mostrarInformacionLenguajes(info);

