Unidad 4

Aplicación de las estructuras de almacenamiento

Contenidos

Parte 1: Estructuras

En programación, las **estructuras de almacenamiento** permiten organizar, gestionar y manipular datos de forma eficiente. Estas estructuras son esenciales porque:

1. Facilitan el manejo de grandes volúmenes de datos.

2. Optimizan el acceso, búsqueda y actualización de datos.

3. Proporcionan soluciones eficientes a problemas complejos.

1. Tipos de Estructuras de Almacenamiento

En Java, existen diversos tipos de estructuras de almacenamiento que pueden clasificarse en dos grandes grupos:

1.1. Estructuras de Datos Lineales

Estas estructuras almacenan datos de forma secuencial (lineal). Ejemplos:

- **Arrays**: Almacenan elementos en posiciones contiguas de memoria.
- **Listas**: Estructuras dinámicas que permiten la inserción y eliminación flexible.
- **Colas y Pilas**: Permiten un acceso controlado (FIFO y LIFO, respectivamente).

1.2. Estructuras de Datos No Lineales

Estas estructuras almacenan los datos de manera jerárquica o relacional:

- **Árboles**: Organizan datos en nodos jerárquicos.
- **Grafos**: Almacenan datos como conjuntos de nodos y relaciones entre ellos.

En este bloque, nos centraremos exclusivamente en **Arrays** y **Cadenas de Caracteres**, ya que constituyen la base de estructuras más avanzadas.

2. Importancia de los Arrays y Cadenas de Caracteres

1. Arrays:

- Son una **estructura fundamental** para organizar y almacenar datos del mismo tipo.
- Permiten acceder a los elementos de manera rápida mediante **índices**.
- Son ideales para operaciones de búsqueda, ordenación y manipulación de datos en bloque.

2. Cadenas de Caracteres (Strings):

- Representan **textos** en Java y son esenciales en el desarrollo de aplicaciones.
- Facilitan operaciones como concatenación, búsqueda, comparación y conversión.

Parte 2: Arrays unidimensionales y multidimensionales

Un **array** en Java es una estructura de almacenamiento estática que almacena un conjunto de elementos del **mismo tipo de dato** en posiciones contiguas de memoria.

Los arrays se dividen en:

1. **Arrays unidimensionales: Almacenan elementos en una sola fila.**
2. **Arrays multidimensionales: Almacenan elementos en dos o más dimensiones, como matrices o tablas.**

1. Declaración de Arrays

1.1. Declaración de Arrays Unidimensionales

La declaración de un array unidimensional especifica el tipo de datos y el nombre del array.

Sintaxis:

```
tipo[] nombreArray;   // Recomendado
tipo nombreArray[];   // Alternativa
```

Ejemplo: Declaración de Arrays

```
int[] numeros;  // Declaración de un array de enteros
double[] precios;   // Declaración de un array de números decimales
String[] nombres;   // Declaración de un array de cadenas
```

1.2. Declaración de Arrays Multidimensionales

Los arrays multidimensionales se declaran con varios pares de corchetes ([ ]).

Sintaxis:

```
tipo[][] nombreArray;   // Array bidimensional (matriz)
tipo[][][] nombreArray; // Array tridimensional
```

Ejemplo: Declaración de Arrays Multidimensionales

```
int[][] matriz;        // Declaración de un array bidimensional
String[][] tabla;    // Declaración de una tabla de cadenas
double[][][] cubo;  // Declaración de un array tridimensional
```

2. Creación de Arrays

Una vez declarado un array, debe ser **creado** asignándole un tamaño fijo.

2.1. Creación de Arrays Unidimensionales

La creación de un array asigna memoria para los elementos del tipo especificado.

Sintaxis:

```
nombreArray = new tipo[tamaño];
```

Ejemplo: Creación de Arrays

```
int[] numeros = new int[5];  // Array con espacio para 5 enteros
String[] nombres = new String[3];  // Array para 3 cadenas
```

2.2. Creación de Arrays Multidimensionales

Para crear arrays multidimensionales, se deben definir el número de filas y columnas.

Sintaxis:

```
nombreArray = new tipo[filas][columnas];
```

Ejemplo: Creación de Arrays Multidimensionales

```
int[][] matriz = new int[3][3];   // Matriz de 3x3
String[][] tabla = new String[2][4];  // Tabla de 2 filas y 4 columnas
```

3. Inicialización de Arrays

La inicialización asigna **valores** a los elementos del array.

3.1. Inicialización de Arrays Unidimensionales

1. Inicialización Directa:

```
int[] numeros = {1, 2, 3, 4, 5};
```

2. Asignación Individual:

```
int[] numeros = new int[3];
numeros[0] = 10;
numeros[1] = 20;
numeros[2] = 30;
```

3.2. Inicialización de Arrays Multidimensionales

1. Inicialización Directa:

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

2. Asignación Individual:

```
int[][] matriz = new int[2][2];
matriz[0][0] = 1;
matriz[0][1] = 2;
matriz[1][0] = 3;
matriz[1][1] = 4;
```

4. Acceso a los Elementos de un Array

Los elementos de un array se acceden mediante su **índice**, que empieza en `0`.

4.1. Acceso en Arrays Unidimensionales

```
int[] numeros = {10, 20, 30};
System.out.println("Primer elemento: " + numeros[0]);  // Accede al índice 0
```

4.2. Acceso en Arrays Multidimensionales

```
java
Copiar código
int[][] matriz = {
    {1, 2},
    {3, 4}
};
System.out.println("Elemento [1][1]: " + matriz[1][1]);  // Accede al valor 4
```

5. Recorridos en Arrays

5.1. Recorrer Arrays Unidimensionales

Con Bucle `for`:

```
int[] numeros = {10, 20, 30, 40};
for (int i = 0; i < numeros.length; i++) {
    System.out.println("Elemento " + i + ": " + numeros[i]);
}
```

Con Bucle `for-each`:

```
for (int num : numeros) {
    System.out.println(num);
}
```

5.2. Recorrer Arrays Multidimensionales

```
int[][] matriz = {
    {1, 2},
    {3, 4}
};
for (int i = 0; i < matriz.length; i++) {
    for (int j = 0; j < matriz[i].length; j++) {
        System.out.println("Elemento [" + i + "][" + j + "]: " + matriz[i]
[j]);
    }
}
```

6. Búsquedas en Arrays

Para buscar un elemento en un array, se recorre elemento por elemento comparando con el valor buscado.

Ejemplo de Búsqueda

```
int[] numeros = {10, 20, 30, 40};
int buscado = 30;
for (int i = 0; i < numeros.length; i++) {
    if (numeros[i] == buscado) {
          System.out.println("Elemento encontrado en la posición: " + i);
          break;
    }
}
```

7. Ordenación de Arrays

Para ordenar un array, usamos el método `Arrays.sort` de la clase `java.util.Arrays`.

Ejemplo: Ordenar un Array

```
import java.util.Arrays;
int[] numeros = {50, 10, 30, 20, 40};
Arrays.sort(numeros);
for (int num : numeros) {
    System.out.println(num);
}
```

Salida:

```
10
20
30
40
50
```

8. Ejercicio Guiado

Problema

Crea un programa que:

1. **Declare un array unidimensional de tamaño** `5` **e inicialice los valores manualmente.**
2. **Encuentre el número mayor y menor del array.**
3. **Imprima todos los elementos ordenados de menor a mayor.**

Parte 3: Cadenas de caracteres

Las **cadenas de caracteres** en Java se representan mediante la clase `String`. A diferencia de los **arrays**, que almacenan elementos del mismo tipo, las cadenas permiten manipular secuencias de **caracteres**. La clase `String` en Java es **inmutable**, es decir, su contenido no puede modificarse después de la creación.

1. Declaración de Cadenas de Caracteres

Las cadenas en Java pueden declararse de dos formas:

1. **Declaración Literal: La forma más común y directa.**
2. **Declaración con el Constructor `new`: Utiliza la clase `String` de forma explícita.**

Sintaxis

Declaración Literal

```
String mensaje = “Hola, Mundo”;
```

Declaración con `new`

```
String mensaje = new String(“Hola, Mundo”);
```

Ejemplo Básico

```
public class EjemploString {
    public static void main(String[] args) {
        // Declaración literal
        String saludo1 = “Hola”;
        // Declaración con new
        String saludo2 = new String(“Hola”);
        System.out.println(saludo1);
        System.out.println(saludo2);
    }
}
```

Salida:

```
Hola
Hola
```

2. Creación e Inicialización

Las cadenas se crean al asignarles un valor o usando el constructor de la clase `String`.

Ejemplo de Inicialización

```
java
Copiar código
String nombre = “Juan”;  // Inicialización directa
String ciudad = new String(“Madrid”); // Creación con el constructor
System.out.println(“Nombre: “ + nombre);
System.out.println(“Ciudad: “ + ciudad);
```

Salida:

```
makefile
Copiar código
Nombre: Juan
Ciudad: Madrid
```

3. Operaciones Básicas con Cadenas

La clase `String` proporciona múltiples métodos para manipular y trabajar con cadenas de caracteres.

3.1. Concatenación

La concatenación permite unir dos o más cadenas.

Uso del Operador `+`

```
String saludo = “Hola”;
String nombre = “Mundo”;
String mensaje = saludo + “, “ + nombre + “!”;
System.out.println(mensaje);
```

Salida:

```
Hola, Mundo!
```

Uso del Método `concat`

```
String saludo = "Hola";
String nombre = "Mundo";
String mensaje = saludo.concat(", ").concat(nombre).concat("!");
System.out.println(mensaje);
```

Salida:

```
Hola, Mundo!
```

3.2. Longitud de una Cadena

El método `length()` devuelve la cantidad de caracteres en la cadena.

```
String mensaje = "Hola, Mundo";
System.out.println("Longitud: " + mensaje.length());
```

Salida:

```
Longitud: 11
```

3.3. Acceso a Caracteres

El método `charAt()` permite acceder a un carácter específico mediante su índice.

```
String mensaje = "Hola";
char primerCaracter = mensaje.charAt(0);
System.out.println("Primer carácter: " + primerCaracter);
```

Salida:

```
Primer carácter: H
```

3.4. Subcadenas

El método `substring()` permite extraer una parte de la cadena.

Ejemplo de Subcadena

```
String mensaje = "Bienvenido a Java";
String subcadena = mensaje.substring(11); // Extrae desde el índice 11
System.out.println("Subcadena: " + subcadena);
```

Salida:

```
Subcadena: Java
```

3.5. Comparación de Cadenas

Existen dos formas principales de comparar cadenas:

1. `equals()`: **Compara el contenido de dos cadenas.**
2. `==`: **Compara las referencias de memoria.**

Ejemplo: Comparación Correcta con `equals()`

```
String cadena1 = "Hola";
String cadena2 = new String("Hola");
System.out.println(cadena1.equals(cadena2)); // true
System.out.println(cadena1 == cadena2);     // false
```

Salida:

```
true
false
```

4. Conversión de Cadenas

La clase `String` permite convertir cadenas a otros tipos de datos.

4.1. Convertir Cadenas a Enteros

```
String numeroStr = "123";
int numero = Integer.parseInt(numeroStr);
System.out.println("Número convertido: " + numero);
```

Salida:

```
Número convertido: 123
```

4.2. Convertir Enteros a Cadenas

```
int numero = 456;
String numeroStr = String.valueOf(numero);
System.out.println("Número como cadena: " + numeroStr);
```

Salida:

```
Número como cadena: 456
```

5. Ejercicios Guiados

Ejercicio 1: Concatenar Cadenas

Escribe un programa que:

1. **Solicite al usuario su nombre y su apellido.**
2. **Concatenar ambos valores en una sola cadena y mostrarlo en consola.**

Ejercicio 2: Contar Vocales en una Cadena

Crea un programa que:

1. **Solicite al usuario que introduzca una cadena de texto.**
2. **Cuente el número de vocales (`a, e, i, o, u`) en la cadena.**

6. Preguntas de Evaluación

1. ¿Cómo se declara una cadena en Java?

a) `String cadena = "texto";`
b) `char cadena[] = "texto";`
c) `int cadena = "texto";`
d) `new cadena = String();`

2. ¿Qué método devuelve la longitud de una cadena?

a) `size()`
b) `length()`
c) `len()`
d) `count()`

3. ¿Qué método se utiliza para concatenar cadenas?

a) `append()`
b) `join()`
c) `concat()`
d) `merge()`

4. ¿Cuál es la diferencia entre `equals()` y `==` en cadenas?

a) `equals()` compara referencias y `==` el contenido.
b) `equals()` compara el contenido y `==` las referencias.
c) Ambas comparan contenido.
d) Ambas comparan referencias.

Conclusión del Bloque: Aplicación de las estructuras de almacenamiento

1. Fundamentos de Estructuras de Almacenamiento

Las estructuras de almacenamiento en Java son herramientas fundamentales para organizar y manipular datos de manera eficiente.

Tipos Principales

- Arrays (unidimensionales y multidimensionales)
- Cadenas de caracteres (String)

2. Arrays

Características Fundamentales

- Estructura estática para elementos del mismo tipo
- Tamaño fijo definido en la creación
- Indexación desde 0

Operaciones Básicas

1. Declaración:

```
tipo[] nombreArray;
```

2. Creación:

```
nombreArray = new tipo[tamaño];
```

3. Acceso:

```
nombreArray[indice];
```

3. Arrays Multidimensionales

Permiten crear estructuras tipo matriz o tabla.

```
int[][] matriz = {
    {1, 2, 3},
    {4, 5, 6}
};
```

4. Cadenas de Caracteres (String)

Métodos Principales

- length(): obtiene longitud
- charAt(): accede a caracteres
- substring(): extrae subcadenas
- equals(): compara contenido

Conversiones Comunes

- String a int: Integer.parseInt()
- int a String: String.valueOf()

5. Aplicaciones Prácticas

- Gestión de datos tabulares
- Procesamiento de texto
- Cálculos matemáticos
- Almacenamiento de registros

6. Ejercicio Práctico

Desarrolla un programa de gestión de calificaciones que incluya:

- Array unidimensional para 5 calificaciones de estudiantes
- Cálculo de:
  - Media
  - Valor máximo
  - Valor mínimo
- Mostrar calificaciones ordenadas de menor a mayor
- Mensaje final con resultados usando String

7. Preguntas de Autoevaluación

1. ¿Cuál es la forma correcta de acceder a elementos en un array?

a) array.get(1)
b) array[0]
c) array(0)
d) array.elemento(1)

2. Para comparar el contenido de dos Strings, ¿qué método debemos usar?

a) ==
b) compare()
c) equals()
d) compareTo()

3. ¿Qué método se utiliza para ordenar un array en Java?

a) array.sort()
b) Arrays.sort()
c) Collections.sort()
d) sort.array()

4. Para convertir un String a número entero, usamos:

a) String.toInteger()
b) Integer.parseInt()
c) Convert.toInt()
d) Int.parse()