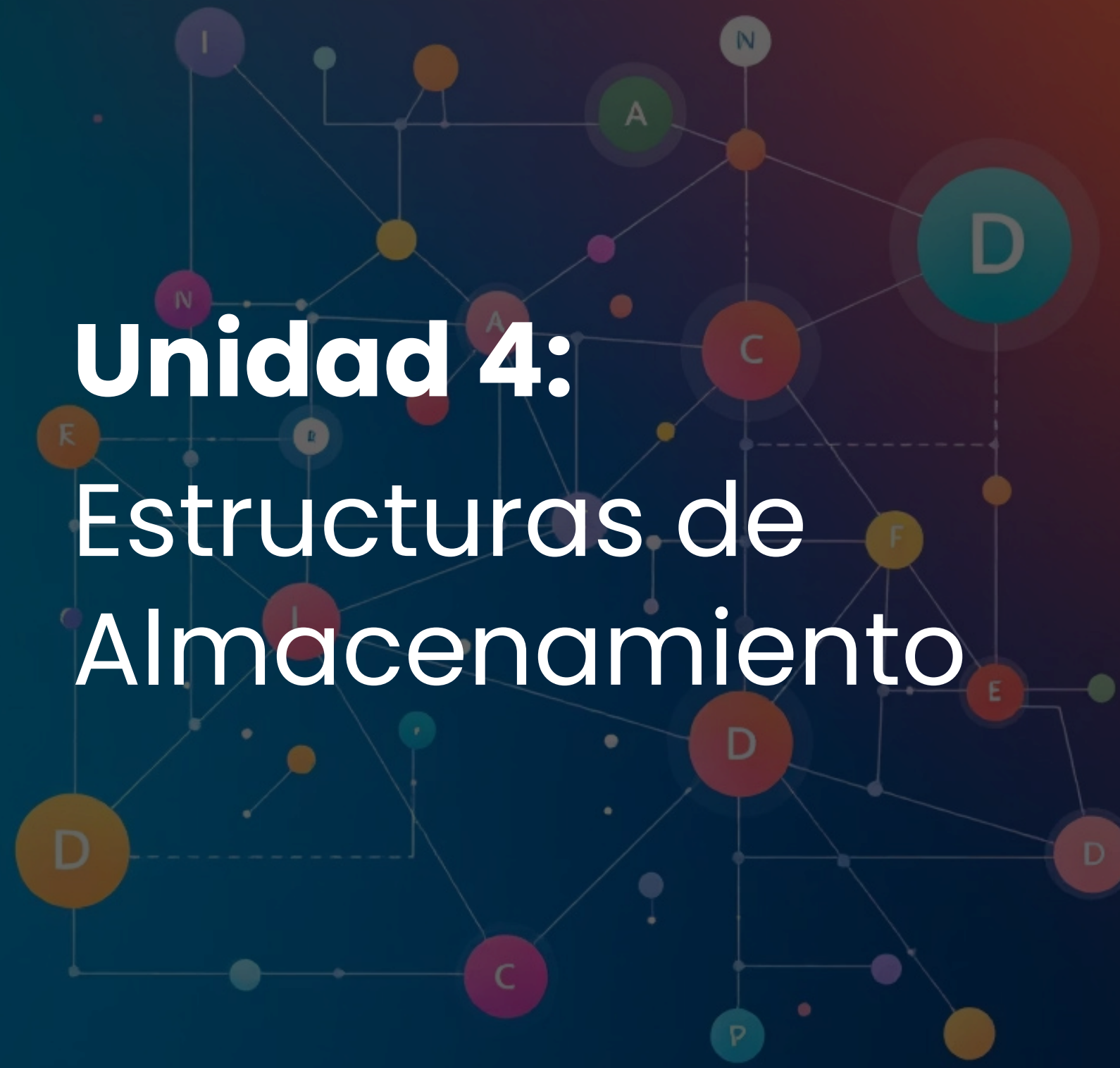
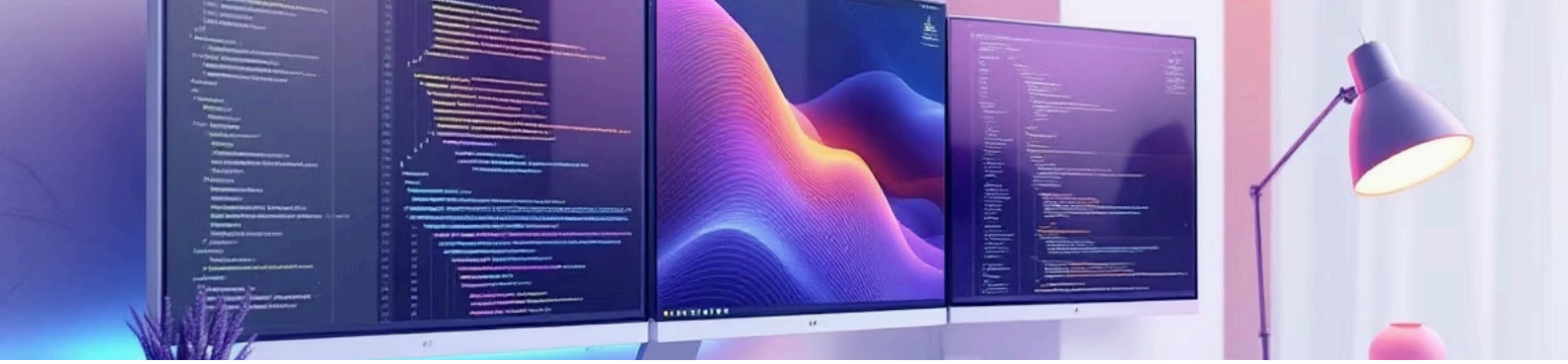
PROMETEO



Unidad 4:
Estructuras de Almacenamiento

Programación

Técnico Superior de DAM / DAW



Sesión 9 – Arrays unidimensionales y multidimensionales

La base del almacenamiento estructurado en programación

Los arrays son una de las estructuras de datos más fundamentales y utilizadas en la programación moderna. En esencia, un array permite almacenar múltiples elementos del mismo tipo en una secuencia ordenada y contigua de memoria, lo que facilita el acceso rápido y eficiente a la información. Imagina una hilera de casillas, cada una numerada y conteniendo un valor: eso es un array unidimensional. Si multiplicas esa hilera por varias filas, obtienes un array multidimensional, perfecto para representar tablas, grillas o matrices de datos.

Array Unidimensional

Un array unidimensional actúa como una lista ordenada. Por ejemplo:

```
int[] numeros = new int[5];
```

Aquí se crea un array capaz de guardar cinco números enteros. Cada posición (índice) se numera desde 0 hasta 4, es decir:

- numeros[0] representa el primer elemento.
- numeros[4] representa el último.

Arrays Multidimensionales

Por su parte, los arrays multidimensionales son literalmente "arrays de arrays". Un ejemplo típico:

```
int[][] matriz = new int[3][4];
```

Esta estructura puede interpretarse como una tabla de 3 filas y 4 columnas. Para acceder a un elemento concreto:

```
matriz[1][2];
```

Esto devuelve el valor ubicado en la segunda fila (índice 1) y la tercera columna (índice 2).

Los índices son cruciales: acceder a una posición fuera del rango (numeros[5], por ejemplo) generará un error de ejecución (ArrayIndexOutOfBoundsException). Por eso, en programación profesional, validar los índices antes de acceder a un elemento es una práctica esencial.

Homogeneidad

Todos los elementos son del mismo tipo de dato.

Tamaño fijo

Una vez declarado, no se puede ampliar ni reducir sin crear un nuevo array.

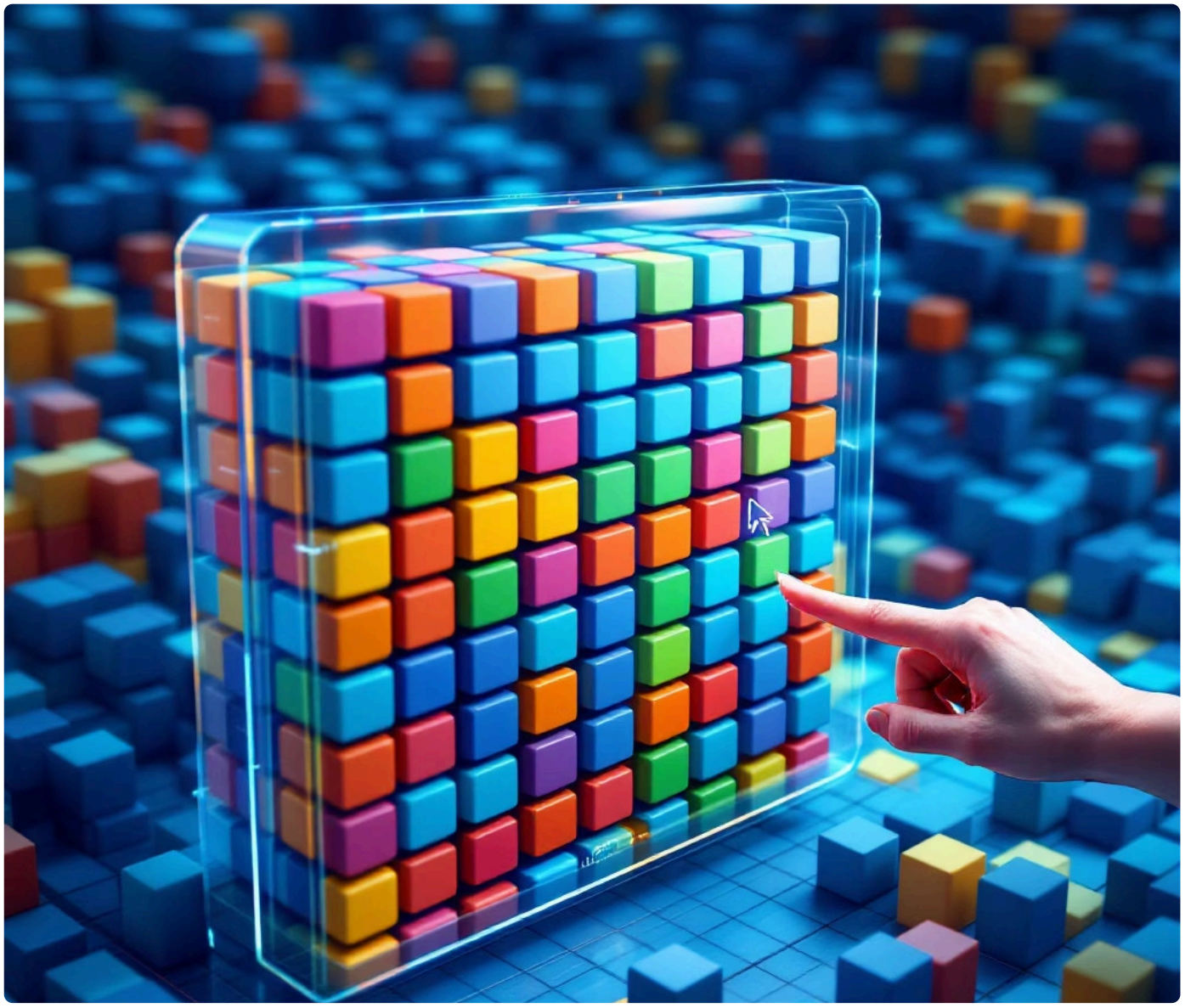
Acceso directo

Cualquier elemento se puede leer o modificar en tiempo constante (O(1)) gracias a su almacenamiento contiguo en memoria.

Propiedad .length

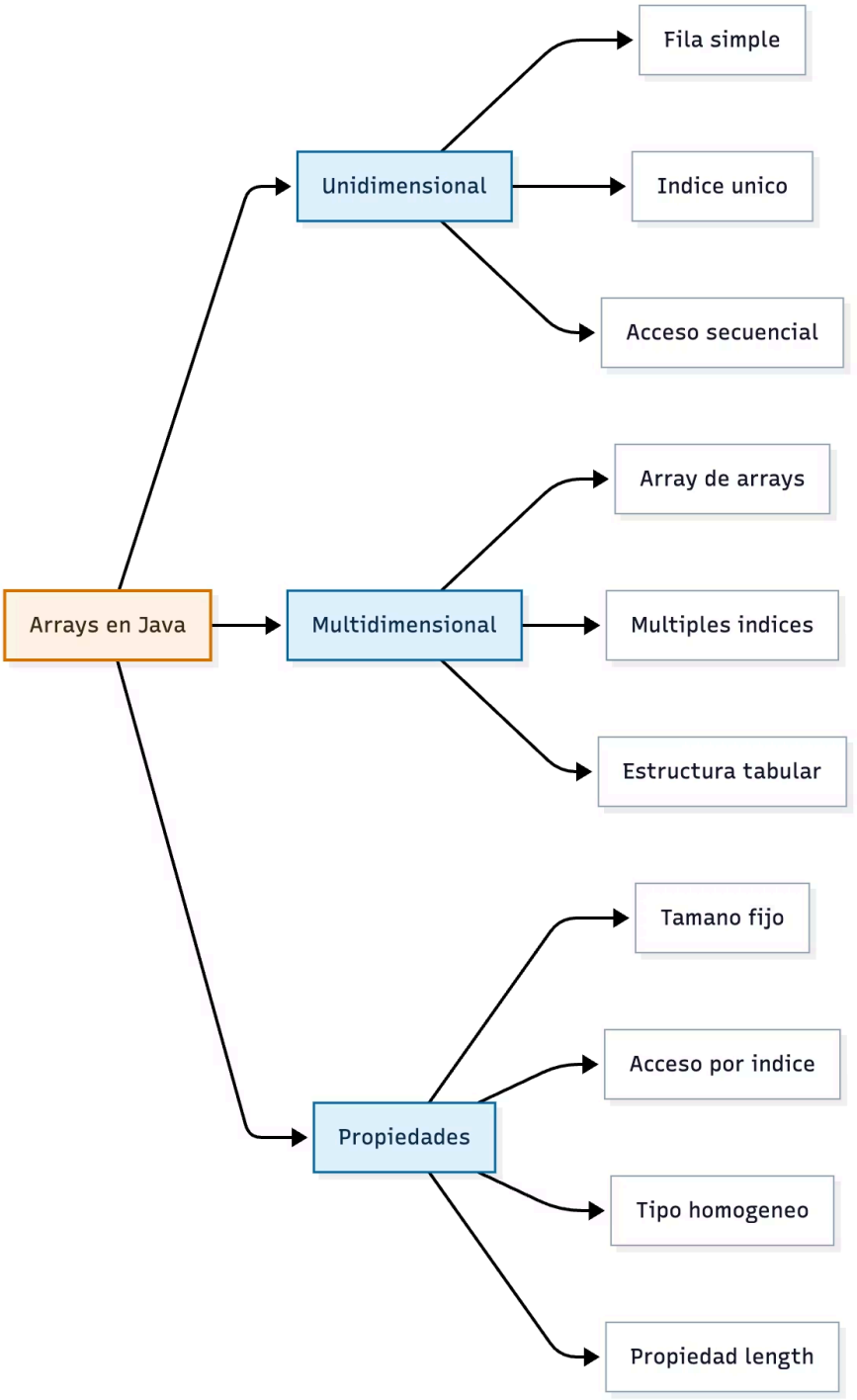
Permite conocer el tamaño total del array (en multidimensionales, cada subarray tiene su propio .length).

En la práctica profesional, los arrays son la base de estructuras más avanzadas como listas dinámicas, matrices numéricas o buffers de datos. Aunque en lenguajes modernos existen colecciones más flexibles (como ArrayList en Java), el conocimiento sólido de los arrays sigue siendo imprescindible para optimizar rendimiento y comprender la gestión de memoria.



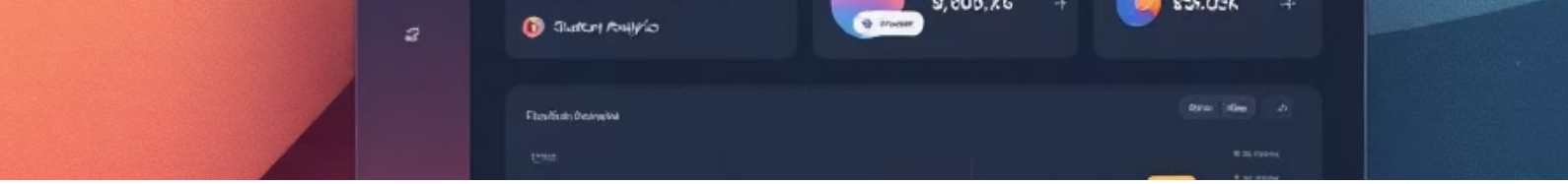
2. Esquema Visual: Concepto de Arrays en Java

El siguiente esquema describe visualmente los tipos y características principales de los arrays.



Interpretación del esquema:

- El nodo **Unidimensional** representa la versión más simple, una sola fila lineal de elementos.
- El nodo **Multidimensional** ilustra que cada elemento puede contener otro array, formando estructuras bidimensionales o incluso tridimensionales.
- Finalmente, las **Propiedades** definen las reglas esenciales: los arrays tienen tamaño fijo, acceden a los datos por índice y almacenan valores del mismo tipo.



Caso de Estudio: Sistema de Notas de Canvas LMS

Contexto: Canvas LMS es una de las plataformas de gestión educativa más utilizadas a nivel global, con millones de usuarios en universidades y centros de formación. Su sistema de calificaciones se basa en el uso intensivo de estructuras de datos optimizadas, entre ellas, arrays.

Estrategia: Cada estudiante tiene asociada una lista de calificaciones, que se modela como un array unidimensional:

```
double[] notasJuan = {8.5, 7.2, 9.1, 6.8};
```

Cada posición del array representa una asignatura diferente. Para obtener la nota de la tercera asignatura:

```
notasJuan[2];
```

Si Canvas necesita gestionar las calificaciones de toda la clase, el sistema emplea un array bidimensional:

```
double[][] notasClase = new double[30][4];
```

- Las filas representan estudiantes (30 en total).
- Las columnas, las asignaturas (4 en este caso).

De este modo, notasClase[5][2] corresponde a la nota del estudiante número 6 (índice 5) en la tercera asignatura (índice 2).

Resultado: Esta estructura permite operaciones rápidas y simultáneas, como:

- Cálculo del promedio por estudiante (for sobre columnas).
- Cálculo del promedio por asignatura (for sobre filas).
- Identificación de la nota más alta o más baja.

Canvas procesa matrices con millones de registros para generar reportes analíticos y visualizaciones en tiempo real. Gracias al uso de arrays, las operaciones de lectura y cálculo mantienen un rendimiento excelente incluso con grandes volúmenes de datos.

Herramientas y Consejos para tu Futuro Profesional

1

Inicialización rápida y legible

Si ya conoces los valores, puedes inicializar directamente:

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

2

Impresión de arrays

- Usa Arrays.toString(array) para arrays unidimensionales.
- Usa Arrays.deepToString(array) para multidimensionales.

Estas funciones, de la clase java.util.Arrays, son ideales para depuración y salida por consola.

3

Validación de índices

Evita errores comunes con una simple condición:

```
if (i >= 0 && i < array.length) {
    System.out.println(array[i]);
}
```

Así evitas el temido ArrayIndexOutOfBoundsException.

4

Redimensionamiento de arrays

Los arrays son de tamaño fijo. Para ampliarlos:

- Crea un nuevo array más grande.
- Usa System.arraycopy() o Arrays.copyOf() para copiar los datos.

En proyectos reales, para estructuras dinámicas se emplea ArrayList, pero entender la base te permite optimizar recursos.

5

Herramientas de visualización y análisis

Usa VisualVM (incluida en el JDK) para monitorear el consumo de memoria y rendimiento de estructuras de datos durante la ejecución.

5. Mitos y Realidades

❌ Mito: "Los arrays siempre comienzan en el índice 1."

✅ **FALSO.** En Java (como en la mayoría de los lenguajes modernos), los arrays comienzan en el índice 0. Por tanto, el primer elemento se accede con array[0]. Este comportamiento proviene del modo en que se gestiona la memoria a bajo nivel.

❌ Mito: "Los arrays multidimensionales ocupan mucha más memoria."

✅ **FALSO.** Un array bidimensional no ocupa memoria de forma exponencial. En Java, los arrays multidimensionales son realmente arrays de arrays, es decir, estructuras de referencias. Solo ocupan la memoria necesaria para sus elementos más los punteros de referencia. Esto los hace más flexibles y eficientes que una estructura rígida tradicional.

Resumen Final para el Examen

- Arrays: estructuras homogéneas de tamaño fijo.
- Unidimensional: fila simple, índice único, acceso directo.
- Multidimensional: arrays de arrays, múltiples índices, estructura tabular.
- Índices: comienzan siempre en 0.
- Canvas LMS: ejemplo de uso real con matrices de calificaciones.
- Consejo: valida índices y usa Arrays.toString() para depuración.

Sesión 10 – Cadenas de caracteres: métodos útiles de String

La manipulación del texto en la era digital

En cualquier aplicación moderna —desde un formulario web hasta una base de datos empresarial— el manejo de texto es esencial. En Java, las cadenas de caracteres o Strings representan precisamente eso: secuencias ordenadas e inmutables de caracteres. Su uso es tan frecuente que la clase String forma parte del núcleo del lenguaje, en el paquete java.lang.

Cuando decimos que un String es inmutable, significa que una vez creado su contenido no puede modificarse. Cada operación que parece "cambiar" una cadena en realidad crea un nuevo objeto en memoria. Por ejemplo:

```
String saludo = "Hola";
saludo = saludo + " mundo";
```

Aquí no se modifica la cadena "Hola". En su lugar, se crea una nueva cadena "Hola mundo" y la variable saludo apunta a ese nuevo objeto. Esta característica garantiza la seguridad y estabilidad de los datos, especialmente cuando se trabaja con múltiples hilos (multithreading), pero también tiene implicaciones de rendimiento en operaciones que implican muchas concatenaciones.

Para manipular Strings de forma eficiente, Java ofrece una amplia gama de métodos integrados. A continuación, se dividen por categoría:

1. Acceso y análisis

length()

Devuelve el número de caracteres de la cadena.

charAt(int index)

Devuelve el carácter en la posición indicada (recordando que el primer índice es 0).

indexOf(String texto)

Devuelve la posición de la primera aparición del texto indicado.

substring(int inicio, int fin)

Extrae una parte de la cadena desde la posición inicio hasta fin - 1.

Ejemplo práctico:

```
String frase = "Programar es divertido";
int longitud = frase.length();       // 21
char letra = frase.charAt(0);        // 'P'
String sub = frase.substring(0, 10);  // "Programar "
```

2. Comparación y búsqueda

En Java, comparar cadenas no es lo mismo que comparar números o referencias. El operador == compara si dos variables apuntan al mismo objeto en memoria, mientras que el método equals() compara el contenido.

```
String a = "Hola";
String b = new String("Hola");

System.out.println(a == b);       // false (referencias distintas)
System.out.println(a.equals(b));   // true (contenido igual)
```

Para comparar ignorando mayúsculas y minúsculas se usa equalsIgnoreCase():

```
"JAVA".equalsIgnoreCase("java"); // true
```

3. Transformación y limpieza

Los Strings se pueden manipular para transformar su contenido:

toLowerCase() y toUpperCase()

Convierten todo a minúsculas o mayúsculas.

trim()

Elimina los espacios al inicio y al final.

replace(CharSequence a, CharSequence b)

Reemplaza texto.

split(String regex)

Divide una cadena en partes según un delimitador.

Ejemplo:

```
String texto = "  Java, Python, C++  ";
String limpio = texto.trim();          // "Java, Python, C++"
String[] lenguajes = limpio.split(", ");
```

4. Eficiencia en concatenaciones

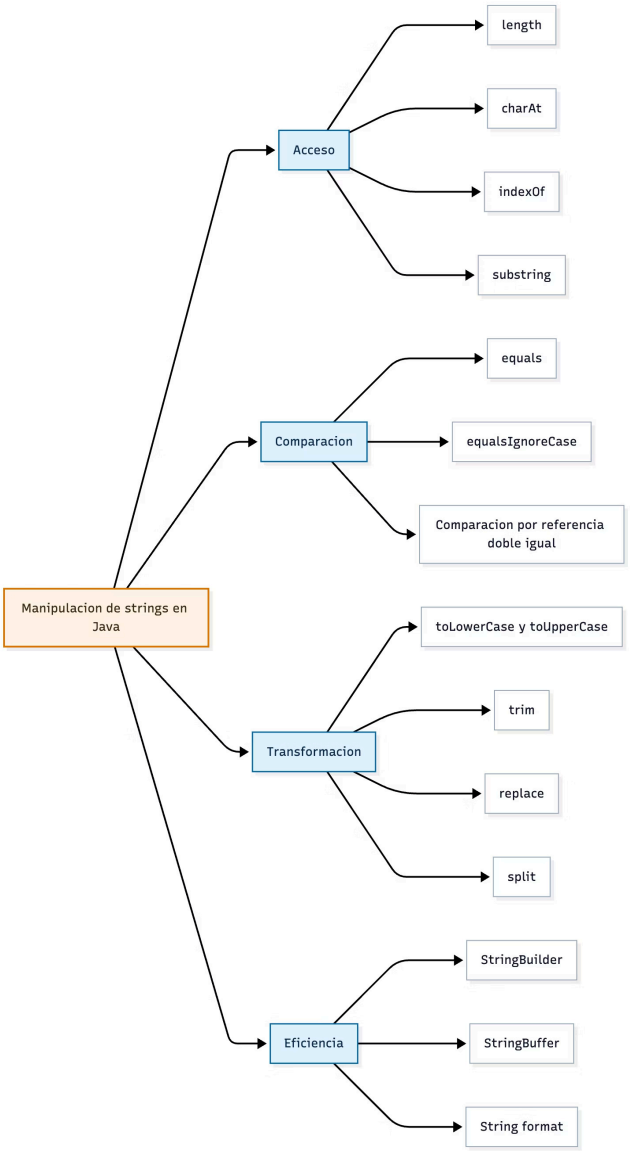
Cuando se necesita construir texto dinámico (por ejemplo, al generar reportes o mensajes personalizados en un bucle), concatenar repetidamente Strings puede volverse costoso. Para evitarlo, Java proporciona la clase StringBuilder, que permite crear cadenas modificables en memoria y luego convertirlas en String mediante .toString().

```
StringBuilder sb = new StringBuilder();
sb.append("Hola");
sb.append(" mundo");
String resultado = sb.toString(); // "Hola mundo"
```

El StringBuilder es mucho más eficiente en operaciones iterativas, y su uso es una práctica estándar en el desarrollo profesional.

Esquema Visual: Métodos fundamentales de String

El siguiente esquema conceptual resume los principales grupos de métodos de la clase String en Java.



Interpretación del diagrama:

- **Acceso:** métodos para leer información de la cadena (tamaño, posiciones, fragmentos).
- **Comparación:** funciones para verificar igualdad o diferencia entre textos.
- **Transformación:** operaciones que cambian el formato o el contenido textual.
- **Eficiencia:** herramientas para construir cadenas grandes o concatenar múltiples textos sin penalizar el rendimiento.

Caso de Estudio: Validación de Contraseñas en Instagram

Contexto:

Instagram gestiona más de **500 millones de intentos de inicio de sesión diarios**. Cada vez que un usuario introduce su contraseña, el sistema debe validar múltiples criterios de seguridad: longitud, presencia de caracteres especiales, mezcla de mayúsculas y minúsculas, y ausencia de patrones comunes.

Estrategia técnica:

El backend de Instagram (basado en Python y Java en distintas capas) utiliza estructuras de texto y métodos equivalentes a los de la clase String en Java para validar entradas de los usuarios antes de enviarlas a la base de datos cifrada.

Un pseudocódigo similar en Java sería:

```
public static boolean validarPassword(String pass) {
    if (pass == null || pass.trim().isEmpty()) return false;
    boolean longitud = pass.length() >= 8;
    boolean mayus = !pass.equals(pass.toLowerCase());
    boolean minus = !pass.equals(pass.toUpperCase());
    boolean numero = pass.matches(".*[0-9].*");
    boolean especial = pass.matches(".*[!@#$%^&*].*");
    return longitud && mayus && minus && numero && especial;
}
```

El sistema combina diferentes métodos (length(), equals(), toLowerCase(), matches()) para evaluar cada condición. Además, se usan objetos como **StringBuilder** para construir mensajes personalizados de error:

```
StringBuilder errores = new StringBuilder();
if (!longitud) errores.append("Debe tener al menos 8 caracteres.\n");
if (!mayus) errores.append("Debe incluir una mayúscula.\n");
if (!especial) errores.append("Debe incluir un carácter especial.\n");
```

Resultado

Este tipo de validación, basada en operaciones con Strings, permite detectar inconsistencias sin exponer información sensible y sin consumir grandes recursos del servidor. La eficiencia es vital: **procesar medio billón de validaciones diarias** requiere un uso óptimo de memoria, concatenación controlada y operaciones atómicas seguras.

En términos de ingeniería, el caso de Instagram demuestra cómo el dominio de las cadenas de caracteres trasciende la teoría: son el corazón de los sistemas de autenticación, mensajería, búsquedas y analítica de texto.

Herramientas y Consejos para tu Futuro Profesional

Usa StringBuilder o StringBuffer para concatenaciones frecuentes

- StringBuilder es más rápido, pero no es seguro en entornos multihilo.
- StringBuffer es thread-safe, recomendado cuando varias tareas pueden acceder a la misma cadena.

```
StringBuilder mensaje = new StringBuilder();
for (int i = 1; i <= 5; i++) {
   mensaje.append("Elemento ").append(i).append(" ");
}
System.out.println(mensaje.toString());
```

Valida cadenas antes de operar

Evita errores comunes comprobando si la cadena no es null ni vacía:

```
if (texto != null && !texto.isEmpty()) { ... }
```

Formatea texto profesionalmente

Usa `String.format()` o `System.out.printf()` para crear mensajes con variables:

```
String nombre = "María";
int edad = 25;
System.out.println(String.format("%s tiene %d años", nombre, edad));
```

Analiza y depura con herramientas de Regex

Plataformas como **RegexPlanet** o **Regex101** te ayudan a construir y probar expresiones regulares para validar o dividir texto (por ejemplo, emails o contraseñas).

Monitorea rendimiento

Usa **VisualVM** o el profiler de tu IDE (como IntelliJ o Eclipse) para identificar cuellos de botella al manipular grandes volúmenes de texto.

Mitos y Realidades

❌ Mito: “Concatenar Strings siempre es ineficiente.” ✅ FALSO. Para pocas concatenaciones, el compilador de Java optimiza automáticamente el código usando StringBuilder internamente. La ineficiencia solo aparece cuando se concatenan miles de veces dentro de bucles, donde sí debe usarse un StringBuilder explícito.

❌ Mito: “equals() y == hacen lo mismo al comparar cadenas.” ✅ FALSO. El operador == compara referencias en memoria, no contenido. Dos Strings pueden tener el mismo texto pero ocupar diferentes ubicaciones de memoria, y == los considerará distintos. El método correcto para comparar el contenido es equals() o equalsIgnoreCase() si se ignoran mayúsculas/minúsculas.

Resumen Final para el Examen

- String: secuencia de caracteres inmutable.
- Métodos clave: length(), charAt(), substring(), indexOf(), equals(), trim(), replace(), split().
- Comparación: equals() compara contenido; == compara referencias.
- Eficiencia: usa StringBuilder para concatenaciones repetitivas.
- Caso real: Instagram valida contraseñas mediante métodos de String combinados con expresiones regulares.
- Consejo: valida siempre entradas nulas o vacías antes de operar.