Junta de Andalucía
Consejería de Educación y Deporte

UT 3
Estructuras de datos
–
Módulo de Programación
1º DAW

Autor: Fran Gómez
2024/2025

¿Qué se va a evaluar?

RA6 Escribe programas que manipulen información seleccionando y utilizando tipos avanzados de datos.

a) Se han escrito programas que utilicen arrays

b) Se han reconocido las librerías de clases relacionadas con tipos de datos avanzados.

c) Se han utilizado listas para almacenar y procesar información.

d) Se han utilizado iteradores para recorrer los elementos de las listas.

e) Se han reconocido las características y ventajas de cada una de la colecciones de datos disponibles.

f) Se han creado clases y métodos genéricos.

g) Se han utilizado expresiones regulares en la búsqueda de patrones en cadenas de texto.

h) Se han identificado las clases relacionadas con el tratamiento de documentos XML.

i) Se han realizado programas que realicen manipulaciones sobre documentos XML.

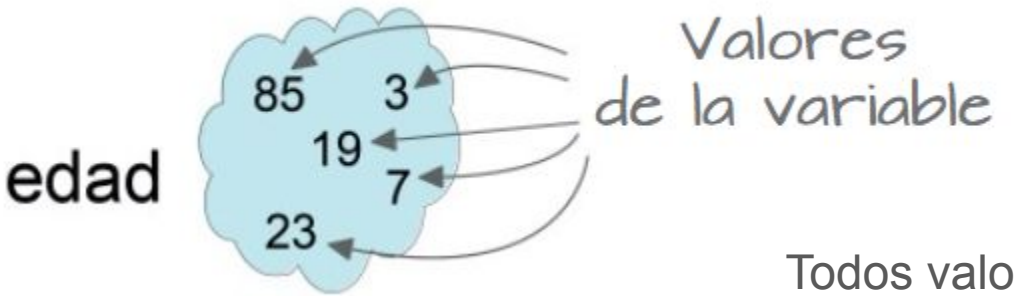
Índice de contenidos

Tablas o Arrays: Introducción

¿Cuántos valores puede almacenar simultáneamente una variable?

```
edad = 6;
edad = 23;
```

Variables escalares



Tablas

Todos valores del mismo tipo

Tablas o Arrays: Concepto

Definición: estructura de datos (de tipo objeto) que almacena un conjunto de elementos de un mismo tipo, accedidos mediante un índice.

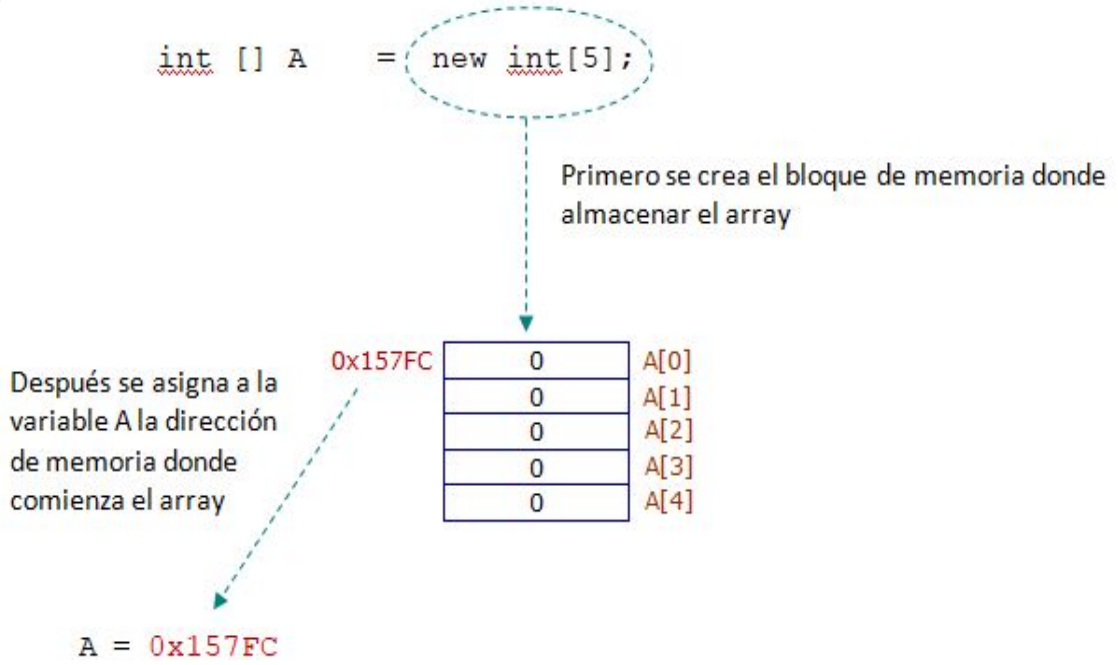
Índice de un array: número natural, comenzando en 0, que identifica las sucesivas posiciones de un array. El último índice sería N-1, donde N es el número de elementos del array.

Tablas o Arrays: Creación

Sintaxis: *tipo [ ] nombre = new tipo[longitud];*

- Declarar variable
  - Tipo
  - Nombre

- Inicializar o creación:
  - Tamaño (estático)
  - Todas la celdas se inicializan con el valor por defecto del tipo



Fuente: Programación Java

```
tipo nombreVariable[];
tipo[] nombreVariable;
```

Tablas o Arrays: Acceso

Operador de acceso → [ ]

Sintaxis: nombreVariable [índiceArray];

Tipos de acceso:

- Lectura → int valor = a[3];
- Escritura → a[3] = valor;

	0	1	2	3	4
a	18	21	19	18	20

Tablas o Arrays: Inicialización

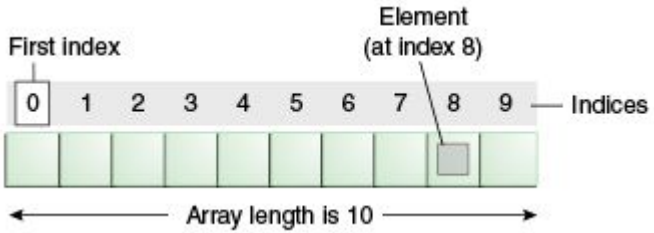
Damos valores a cada celda

```
int datos[]; //declaramos la variable
datos = new int[4]; //creamos la tabla
datos[0] = 2; //asignamos valores
datos[1] = -3;
datos[2] = 0;
datos[3] = 7;
```

En una sóla línea:

```
int datos[] = {2, -3, 0, 7}; //tabla de longitud 4
```

Tablas o Arrays: Rango



Fuente: Oracle Java documentation

Sintaxis: *tipo [ ] nombre = inicialización;*

```
int [] enteros = {1,3,5,7,9};

int [] enteros = new int[5];
enteros[0] = 1;
enteros[1] = 3;
enteros[2] = 5;
enteros[3] = 7;
enteros[4] = 9;
```

Si nos salimos del rango → Error en tiempo de ejecución.

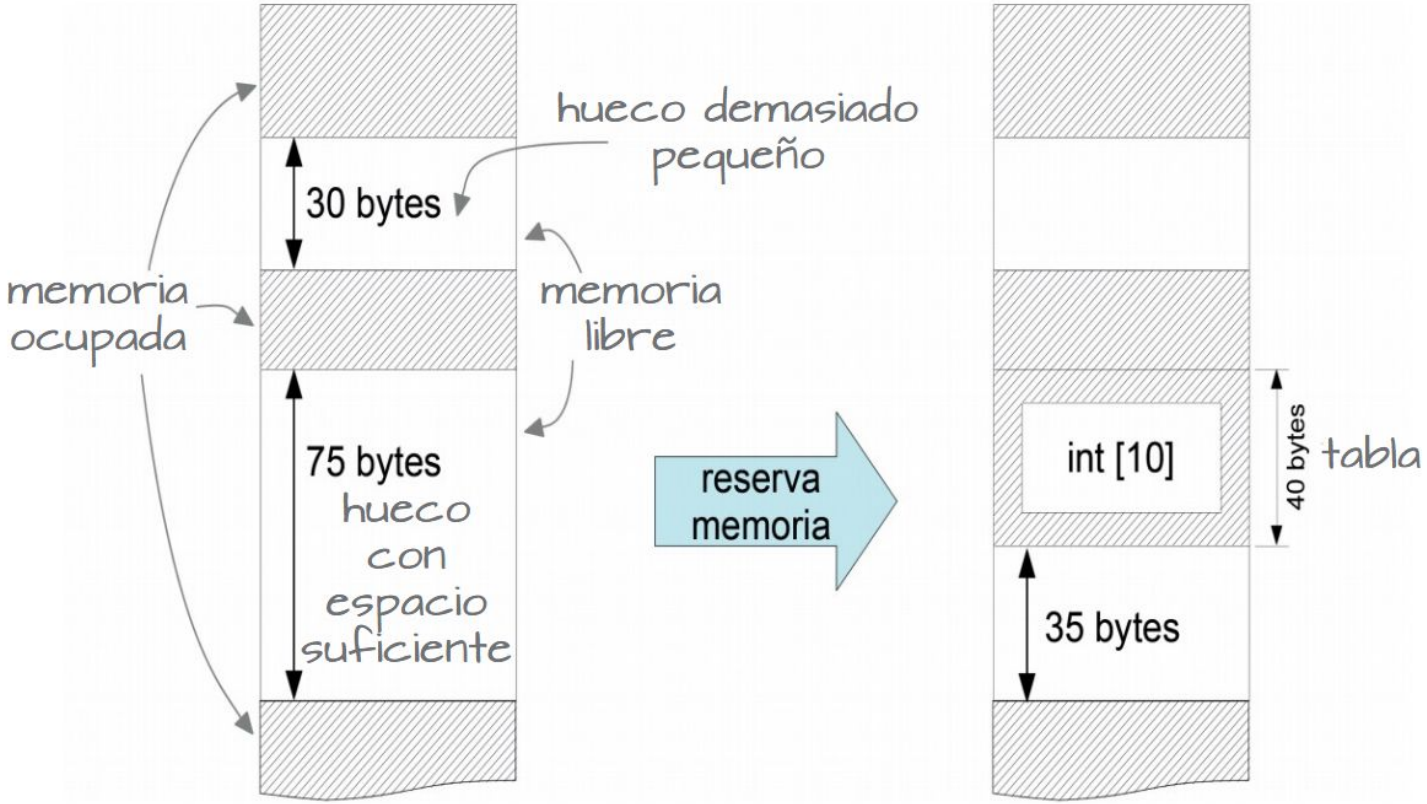
int quinto = enteros[5]

Length: propiedad (no método) de los arrays que indica la longitud del array.

```
int [] enteros = {1,3,5,7,9};
System.out.println(enteros.length); // 5
```

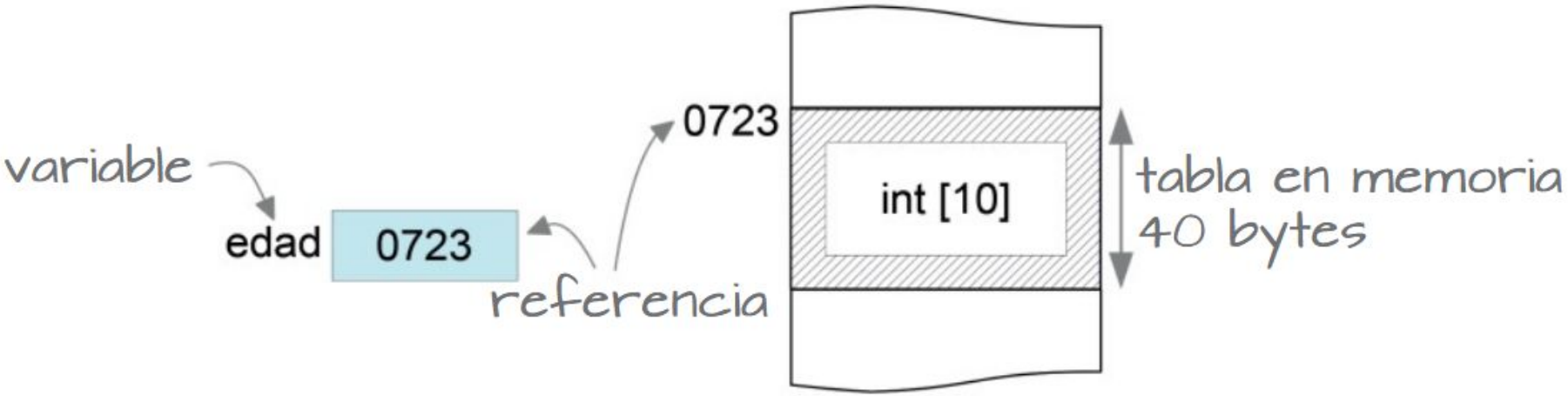
Tablas o Arrays: Referencias

```
edad = new int[10];
```



Tablas o Arrays: Referencias

```
edad = new int[10];
```




```
int t[] = new int[10];
System.out.println(t);
```

Tablas o Arrays: Referencias


```
edad = new int[10];
```

Tablas o Arrays: Referencias

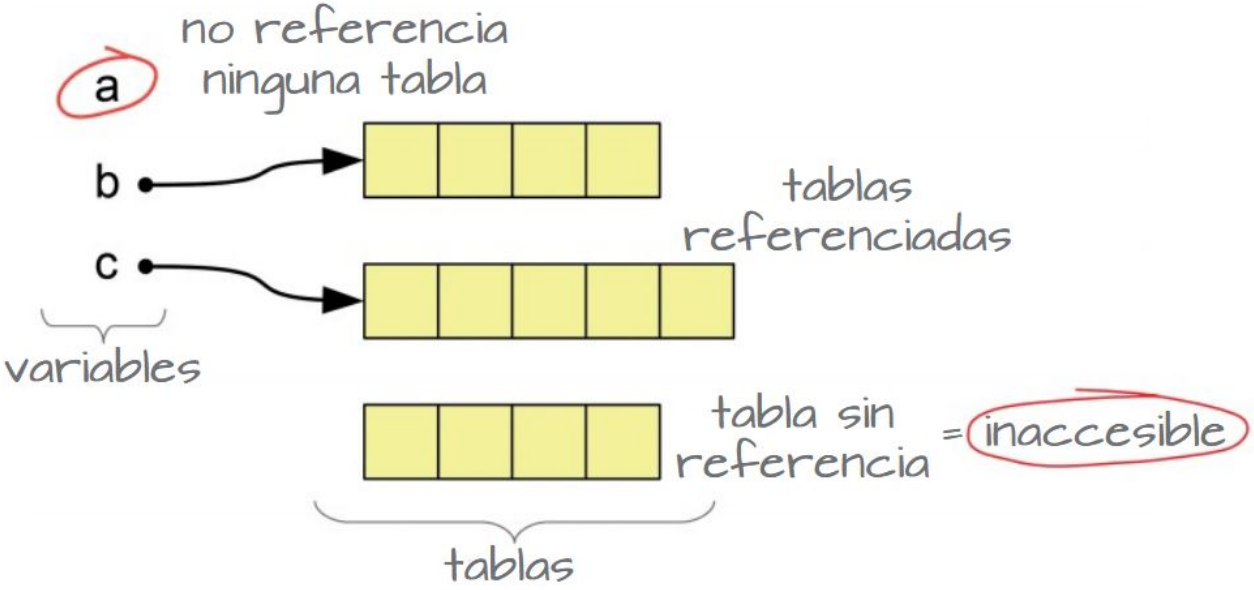
Ejercicio 1

Crea tres tablas de cinco elementos: la primera de enteros, la segunda de double y la tercera de booleanos.

Muestra las referencias en las que se encuentra cada una de las tablas anteriores.

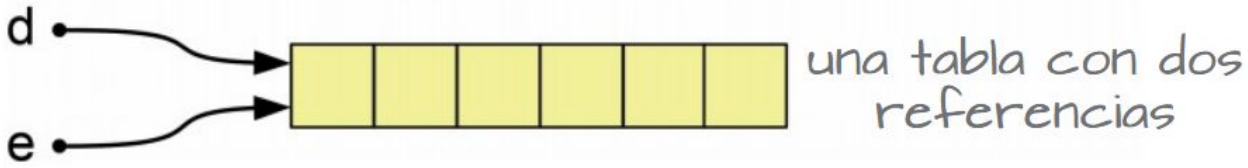
Tablas o Arrays: Referencias

```
int a[], b[], c[]; //variables
b = new int[4]; //tabla de cuatro enteros accesible mediante la variable b
c = new int[5]; //tabla de cinco enteros accesible mediante la variable c
new int[3]; //creamos una tabla cuya referencia no se asigna a ninguna variable
```



Tablas o Arrays: Referencias

```
int d[], e[]; //variables
d = new int[6]; //construimos una tabla referenciada por d
e = d; //ahora la variable e referencia la misma tabla que d. Ambas guardan
       //la misma dirección de memoria
```



Tablas o Arrays: Referencias

Ejercicio 2

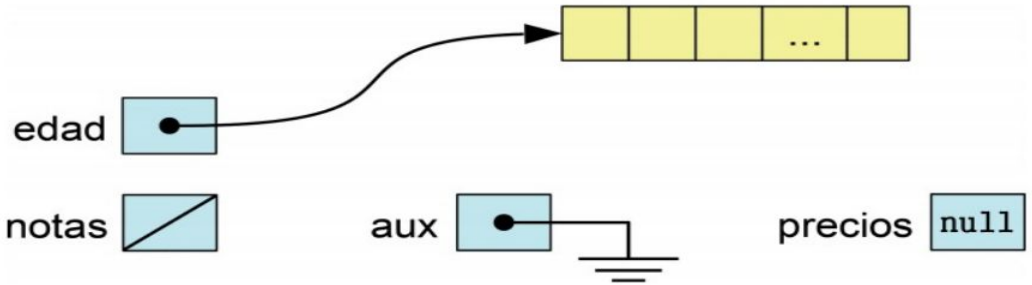
Construye una tabla de 10 elementos del tipo que desees. Declara diferentes variables de tabla que referenciarán la tabla creada.

Comprueba, imprimiendo por pantalla, que todas las variables contienen la misma referencia.

Recolector de basura

Veamos un ejemplo de cómo dejar sin referencia a una tabla:

```
int t1[], t2[]; //variables de tipo tabla entera
t1 = new int[100]; //t1 referencia una tabla de 100 elementos
t2 = t1; //ahora t2 también referencia la misma tabla
t1 = null; //anulamos t1: no referencia nada
          //la tabla sigue siendo accesible desde t2
t2 = null; //anulamos t2: tampoco hace referencia a nada
//la tabla es inaccesible: el recolector de basura se encargará de ella
```



Paso de parámetros

Cualquier método puede recibir y devolver parámetros de cualquier tipo, incluidos los de tipo array.

```
    int x = 1;
    int y = 2;
    String z = "hola";
    Double[] arrayDeEntrada = {1d,2d,3d};
    int[][] datosRetorno = metodo(x, y, z, arrayDeEntrada);
}

static int[][] metodo (int x, int y, String z, Double[] arrayDeEntrada) {
    return new int[2][2];
}
```

Pero para los arrays **¿se pasa la referencia o una copia del array?**

Paso de parámetros

Paso por Valor:

- Se envía una **copia del valor** de la variable al método.
- Los cambios realizados dentro del método **NO afectan** a la variable original.
- Ejemplo: Variables primitivas en Java (int, double, char, etc.).

Paso por Referencia:

- Se envía una **referencia (dirección de memoria)** de la variable al método.
- Los cambios realizados dentro del método **SÍ AFECTAN** a la variable original.
- Ejemplo: Objetos y arrays en Java.

En **Java**, el paso de parámetros es siempre por **valor** → Pero la gestión que hace Java de la memoria provoca el mismo efecto que el paso por **referencia**

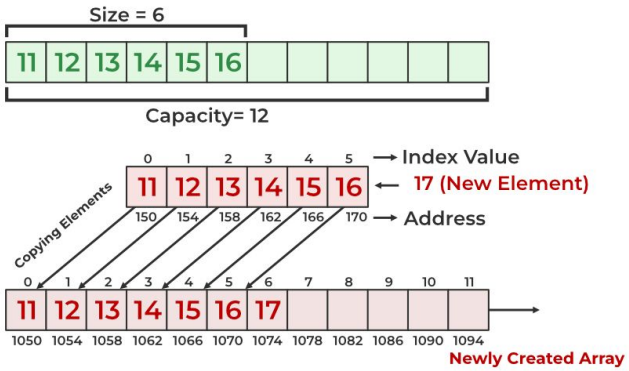
Paso de parámetros


```java
public class Ejemplo {
    public static void main(String[] args) {
        int valor = 10;
        cambiarValor(valor);
        System.out.println("Después del método: " + valor); // Imprime 10

        int[] array = {10};
        cambiarArray(array);
        System.out.println("Después del método: " + array[0]); // Imprime 20
    }

    public static void cambiarValor(int numero) {
        numero = 20; // No afecta a la variable original
    }

    public static void cambiarArray(int[] array) {
        array[0] = 20; // Afecta al objeto original
    }
}
```

Tipos de Arrays

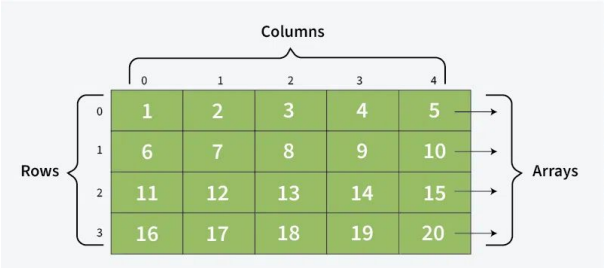
Según cómo sea la reserva de memoria:

- Estáticos
- Dinámicos



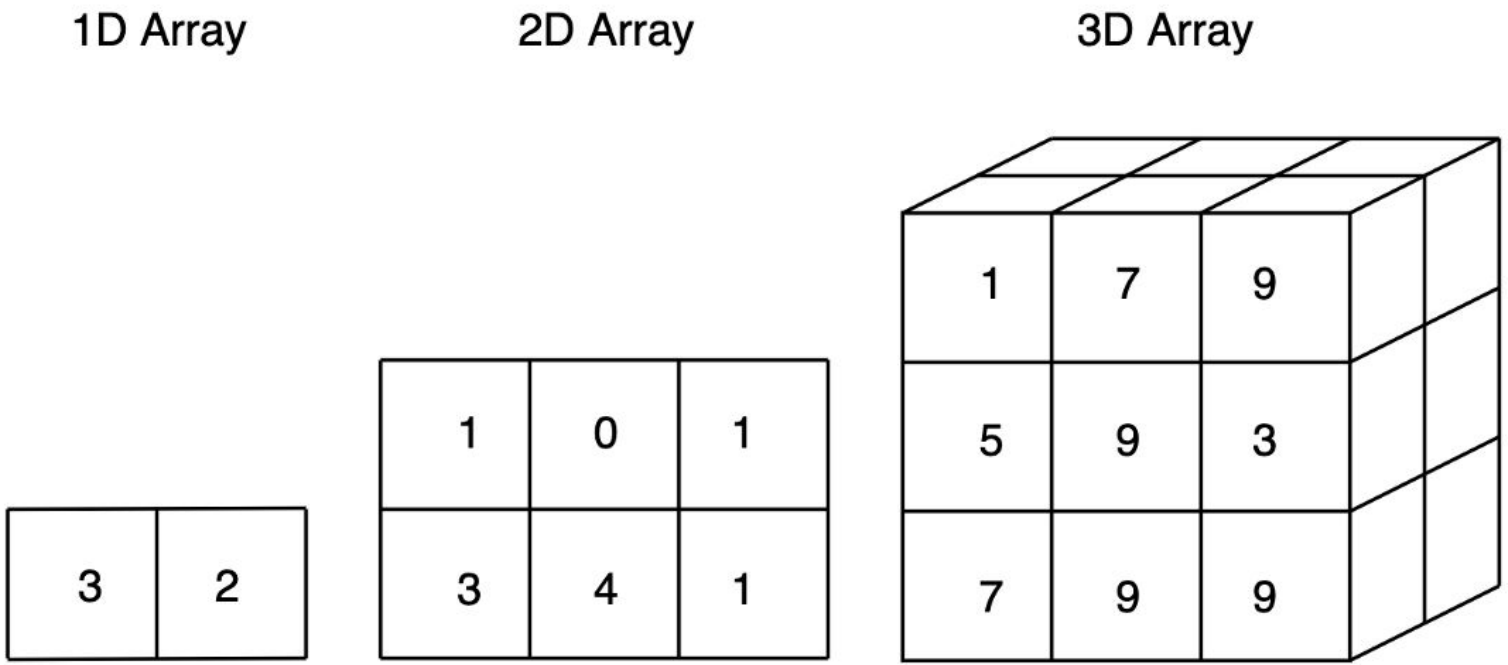
Según las dimensiones:

- Unidimensional
- Multidimensional
  - Bidimensional (matriz)




```
int [][] enteros = new int[5][2];
enteros[0][0] = 1;
enteros[0][1] = 2;

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

Dimensiones de los arrays



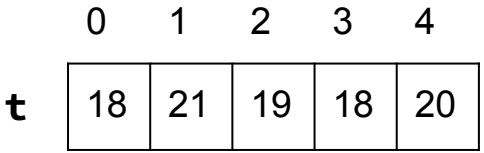
Operaciones con arrays

Podemos realizar múltiples operaciones sobre cualquier tipo de array

- Recorrido
- Impresión
- Inicialización masiva
- Búsqueda
- Ordenación
- Comparación
- Copia
  - Superficial
  - Profunda
- Inserción
- Eliminación
- Intercambio

Vamos a ir explorándolas mientras introducimos más conceptos…

Recorrer un array



¿Como podemos ir recorriendo todos los elementos del array?

```
for (int i = desde; i <= hasta; i++) {
    //procesado de t[i]
    ...
}
```

for-each

```
for (declaración variable: tabla) {
    ...
}
```

Simplifica la acción a obtener los elementos secuencialmente

```
int naturales[] = {1,2,3,4,5};
for (int x : naturales) {
    System.out.println(x);
}
```

Recorrer un array

```
for (int i = desde; i <= hasta; i++) {
   //procesado de t[i]
   ...
}
```

Ejercicio 3

- Crea una tabla como la de la imagen que representa los sueldos de los empleados de una empresa.
- Auméntales el sueldo un 10% a cada empleado de forma que el array de sueldos quede actualizado

	0	1	2	3	4
sueldos	1800	1200	2000	1200	900

Ejercicio 4

Crear una tabla de longitud 10 que se inicializará con números aleatorios comprendidos entre 1 y 100.

Mostrar la suma de todos los números aleatorios que se guardan en la tabla.

Imprimir un array

int [] a = {2,4,6,8,10}

System.out.println(a); ⇒ ¿Qué imprimirá por pantalla?

Imprimir el contenido y no la referencia ⇒ Hay que acceder al contenido

Ejercicio: imprime la referencia y el contenido del array anterior.

- for
- foreach

Imprimir un array

Ejercicio 5

Crea un método que reciba un array de enteros y devuelva un entero con la suma de todos los números del array. Invoca este método desde un main e imprime el resultado.

Nota: tendrás que inicializar el array en el main antes de invocar al método.

Ejercicios

Ejercicio 6

Diseñar un programa que solicite al usuario que introduzca por teclado 5 números decimales. A continuación, mostrar los números en el mismo orden que se han introducido.

Ejercicio 7

Escribir una aplicación que solicite al usuario cuántos números desea introducir. A continuación, introducir por teclado esa cantidad de números enteros, y por último, mostrar en el orden inverso al introducido.

Ejercicio 8

Diseñar la función: int maximo (int t[]), que devuelva el máximo valor contenido en la tabla t.

Ejercicio 9

Recorrer una matriz (array bidimensional)

A	B	C
D	E	F
G	H	I

Inicializar la matriz e imprimirla para que muestre: A B C D E F G H I

Clase java.util.Arrays

https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html

Ver la forma alternativa de imprimir, inicializar, etc…

java.util.Arrays.toString

java.util.Arrays.fill

…

Imprimir con java.util.Arrays.toString


```java
import java.util.Arrays;

public class ArrayToString {
    public static void main(String[] args) {

        // create an integer array
        int[] n = {1, 2, 3, 4,};

        // print the array using
        // Arrays.toString()
        System.out.println(Arrays.toString(n));
    }
}
```

Output

```
[1, 2, 3, 4]
```

Inicialización masiva

java.util.Arrays.fill

Syntax of Arrays.fill() method

public static void fill(int[] a, int val)

public static void fill(int[] a, int fromIndex, int toIndex, int val)

Parameters:

- `a`: Array to be filled.
- `val`: Value to assign to each element of the array.
- `fromIndex`: Starting index (inclusive) for filling.
- `toIndex`: Ending index (exclusive) for filling.

Return Type: It does not return any value, but modifies the array directly.

Inicialización masiva

Ejercicio 10

Crea un programa que:

- Declare un array de números enteros de tamaño 10.
- Use `Arrays.fill` para inicializar el array con un mismo valor.
- Modifique los valores de una parte del array (por ejemplo, las posiciones 5 a 8) con un nuevo valor usando otra variante de `Arrays.fill`.
- Imprima el array después de cada modificación para observar los cambios. Usa para ello la clase Arrays.

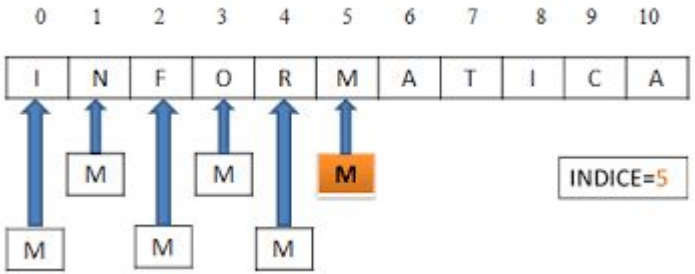
Consulta la API como referencia para saber el funcionamiento

Buscar un elemento en un array

Algoritmos de búsqueda

1. Búsqueda secuencial → aplica a arrays desordenados
2. Búsqueda binaria → aplica a arrays ordenados

Búsqueda secuencial o lineal



Recorro todo el array en busca del elemento

complejidad O(n)

```java
public static int busquedaLineal(int[] array, int clave) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == clave) {
            return i; // Retorna la posición del elemento.
        }
    }
    return -1; // No encontrado.
}
```

Buscar un elemento en un array

Ejercicio 11:

1. Crea un método que reciba un número y un array y devuelva la posición donde ha encontrado el número o -1 si no ha encontrado el número.

2. Crea un array con 5 números aleatorios y busca dentro de él un número introducido por el usuario, invocando el método anterior.

3. Compara el resultado de tu método con el del método java.util.Arrays.binarySearch

Ordenación de arrays

Algoritmos de ordenación

1. Ordenación burbuja
2. Ordenación por selección
3. Ordenación por inserción

Ordenación de arrays

Ejercicio 12:

En el ejercicio anterior, invoca un nuevo método que reciba un array y lo ordene: java.util.Arrays.sort

Invoca a este método desde el main pasándole el array de números aleatorios. Haz esta invocación antes de realizar la llamada al método de búsqueda.

Vuelve a comparar resultados entre la búsqueda binaria y la secuencial.

Comparar arrays

Dados dos arrays, ¿cómo comparo si son iguales o distintos?

```
int [] a = {1,2,3};

int [] b = {1,2,3};
```

```
int x = 5;
int y = 5;
x == y → true
```

- Comparar referencias ⇒ ==
  - a == b → false
- Comparar contenido ⇒ java.util.Arrays.equals
  - Arrays.equals(a, b) → true

Comparar arrays

Ejercicio 13

Crea un algoritmo dentro de un método con la siguiente cabecera:

public static boolean compararArrays(int[] array1, int[] array2)

Deberá devolver *true* si ambos arrays tienen el mismo contenido y *false* en caso contrario.

No uses métodos auxiliares de ninguna librería.

Realiza pruebas con los siguientes arrays desde un método main:

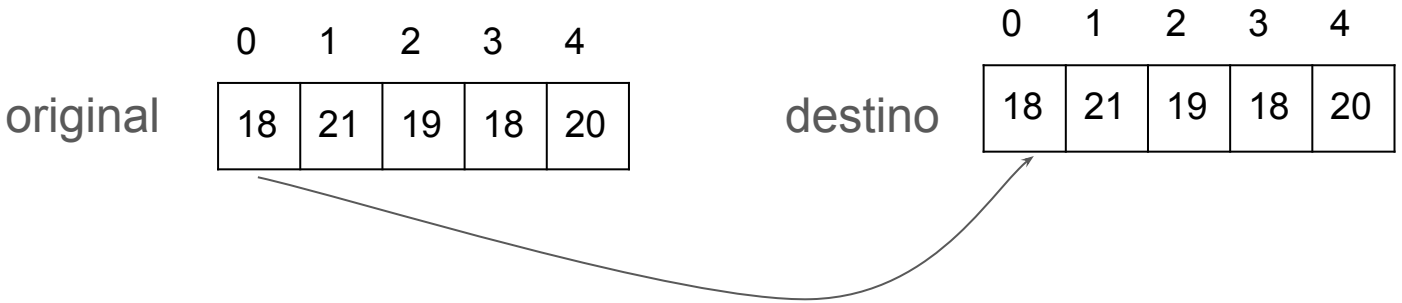
```
int[] array1 = {1, 2, 3};
int[] array2 = {1, 2, 3};
int[] array3 = {3, 2, 1};
int[] array4 = {1,2,3,4}
```

Copiar arrays

El procedimiento para realizar manualmente la copia exacta de una tabla consiste en:

1. Crear una nueva tabla, que llamaremos *destino* o *copia*, del mismo tipo y longitud que la tabla original.

2. Recorrer la tabla original, copiando el valor de cada elemento en su lugar correspondiente en la tabla destino.



Copiar arrays: copyOf

- `static tipo[] copyOf(tipo origen[], int longitud)`: construye y devuelve una copia de `origen` con la longitud especificada. Si la longitud de la nueva tabla es menor que la de la original, solo se copian los elementos que caben. En caso contrario, los elementos extras se inicializan por defecto. Este método, como la mayoría de los métodos de `Arrays`, está sobrecargado para poder trabajar con todos los tipos.

Veamos un ejemplo:

```
int t[] = {1, 2, 1, 6, 23}; //tabla origen
int a[], b[]; // tablas destino
a = Arrays.copyOf(t, 3); //a = [1, 2, 1]
b = Arrays.copyOf(t, 10); //b = [1, 2, 1, 6, 23, 0, 0, 0, 0, 0]
```

Copiar arrays: copyOfRange

- `static tipo[] copyOfRange(tipo origen[], int desde, int hasta)`: crea y devuelve una tabla donde se han copiado los elementos de `origen` comprendidos entre los índices `desde` y `hasta`, sin incluir este último.

Un ejemplo:

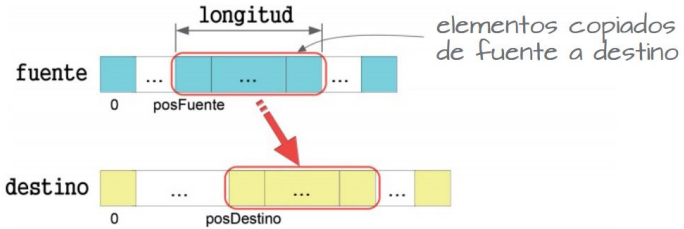
```
int t[] = {7, 5, 3, 1, 0, -2};
int a[] = Arrays.copyOfRange(t, 1, 4); //a = [5, 3, 1]
```

que realiza una copia desde los índices 1 al 3 (el anterior al 4).

Copiar arrays: arraycopy

Otro método disponible es `arraycopy()` de la clase `System`, que copia elementos consecutivos entre dos tablas. La diferencia entre `arraycopy()` y `copyOfRange()` es que el primero no crea ninguna tabla, ambas tablas deben estar creadas previamente. Su sintaxis es:

- `void arraycopy(Object tablaOrigen, int posOrigen, Object tablaDestino, int posDestino, int longitud`: copia en la `tablaDestino`, a partir del índice `posDestino`, los datos de la `tablaOrigen`, comenzando en el índice `posOrigen`. El parámetro `longitud` especifica el número de elementos que se copiarán entre ambas tablas. Hay que tener precaución, ya que los valores de los elementos afectados por la copia de la tabla destino se perderán. Véase la Figura 5.13.



Copia de arrays: ejercicio

Ejercicio 14

Crea un programa donde declares e inicialices un array de enteros a tu elección.

Seguidamente, crea 4 copias, cada una usando uno de los métodos anteriores.

Utiliza el método del ejercicio 13, compararArrays para comprobar que están bien copiados.

Opcional: invierte el orden de los elementos del array original.

Inserción de elementos en un array

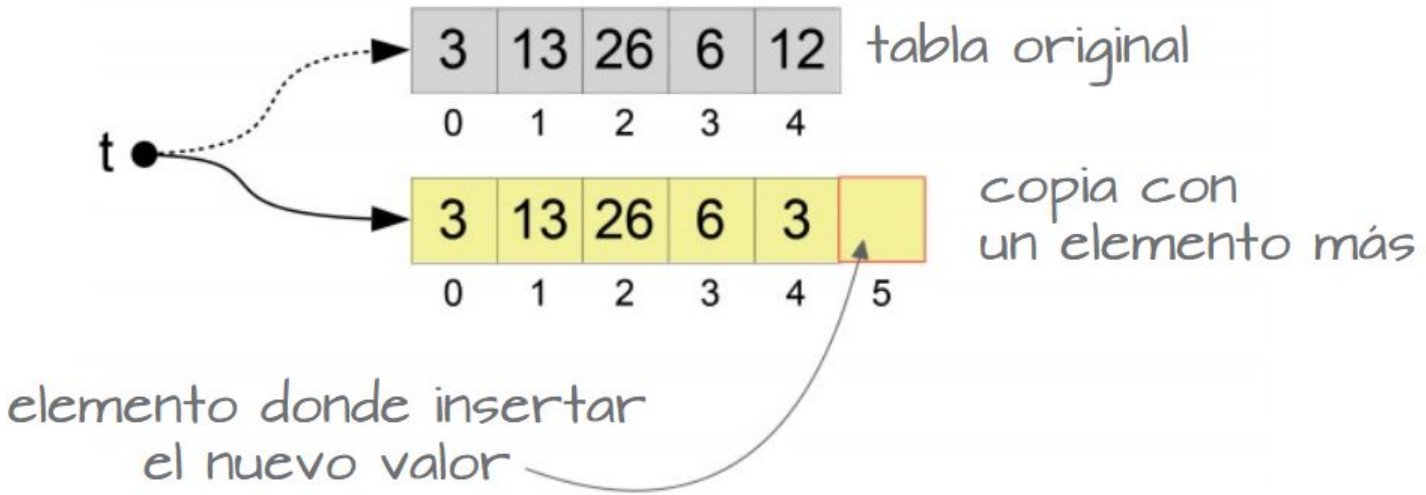
Depende de si la tabla está o no ordenada:

- **No ordenada**: podemos insertarlo al final
- **Ordenada**: hay que insertarlo en la posición que le corresponda

En ambos casos habrá que crear un nuevo array con una posición más que sustituya al anterior y copiar el resto de elementos.

Inserción en array no ordenado

```
t = Arrays.copyOf(t, t.length + 1); //la copia incrementa la longitud
t[t.length-1] = nuevo;
```

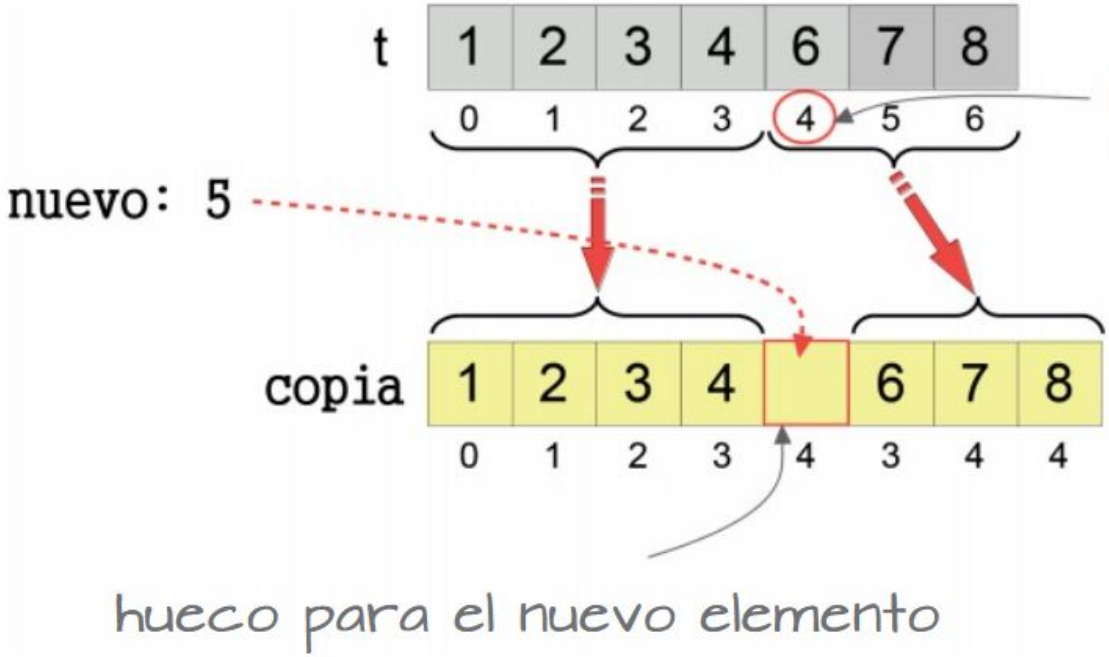


Inserción en array no ordenado

Ejercicio 15

Leer y almacenar n números enteros en una tabla, a partir de la que se construirán otras dos tablas con los elementos con valores pares e impares de la primera, respectivamen te. Las tablas pares e impares deben mostrarse ordenadas.

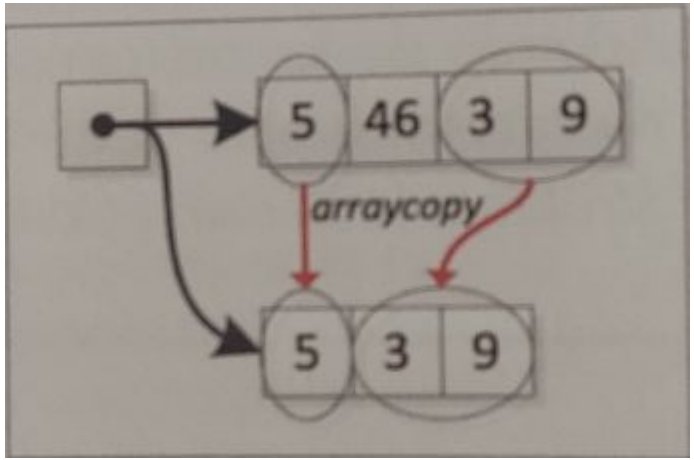
Inserción en array ordenado




```
int t[] = {1, 2, 3, 4, 6, 7, 8};
int nuevo = 5;
int indiceInsercion = Arrays.binarySearch(t, nuevo);
//si indiceInsercion >= 0, el nuevo elemento (que está repetido) se inserta en
//el lugar en que ya estaba, desplazando al original. Si por el contrario:
if (indiceInsersion < 0) { //si no lo encuentra
   //calcula donde debería estar
   indiceInsercion = -indiceInsercion - 1;
}
int copia[] = new int[t.length + 1]; //nueva tabla con longitud+1
//copiamos los elementos antes del "hueco"
System.arraycopy(t, 0, copia, 0, indiceInsercion);
//copiamos desplazados los elementos tras el "hueco"
System.arraycopy(t, indiceInsercion,
copia, indieInsercion+1, t.length - indiceInsercion);
copia[indiceInsercion] = nuevo; //asignamos el nuevo elemento
t = copia; //t referencia la nueva tabla
System.out.println(Arrays.toString(t)); //mostramos
```

Eliminación de elementos en un array

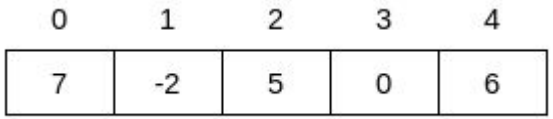
- Manualmente.
- Usando algún método predefinido como arraycopy.



Fuente: Programación Ed. Síntesis.

Intercambio de elementos en el array

a




```
int aux = a [2];
a [2] = a [3];
a [3] = aux;
```

Ejercicio: desplaza los elementos del array anterior una posición a la derecha, el último elemento pasaría a la posición 0.

Cadenas de caracteres: String

Cadenas de caracteres: String

- String no es un tipo primitivo sino una referencia (una clase de la API de Java)

- Puede verse como una clase envoltorio de un array de chars: char[]
  - Recuerda que un char sí es un tipo primitivo y va entre comillas simples ‘’

- La clase String es especial, sus instancias van una zona distinta de la memoria. Esto implica que no hace falta usar el new. Podemos usar comillas dobles “”
  - String cadena = “Hola”;
  - String cadena = new String(“Hola”);

- La clase String es inmutable: cualquier operación sobre un String crea un nuevo String.
  - String cadena = “Hola” + “mundo!\n”

Secuencias de escape

Carácter	Nombre
`\b`	Borrado a la izquierda
`\n`	Nueva línea
`\r`	Retorno de carro
`\t`	Tabulador
`\f`	Nueva página
`\'`	Comilla simple
`\"`	Comilla doble
`\\`	Barra invertida

Cadenas de caracteres: String

Ejemplo:

Crear una cadena de caracteres que contenga un símbolo Unicode e imprimirla por consola.

```
"Un corazón: \u2661"
```

Unicode

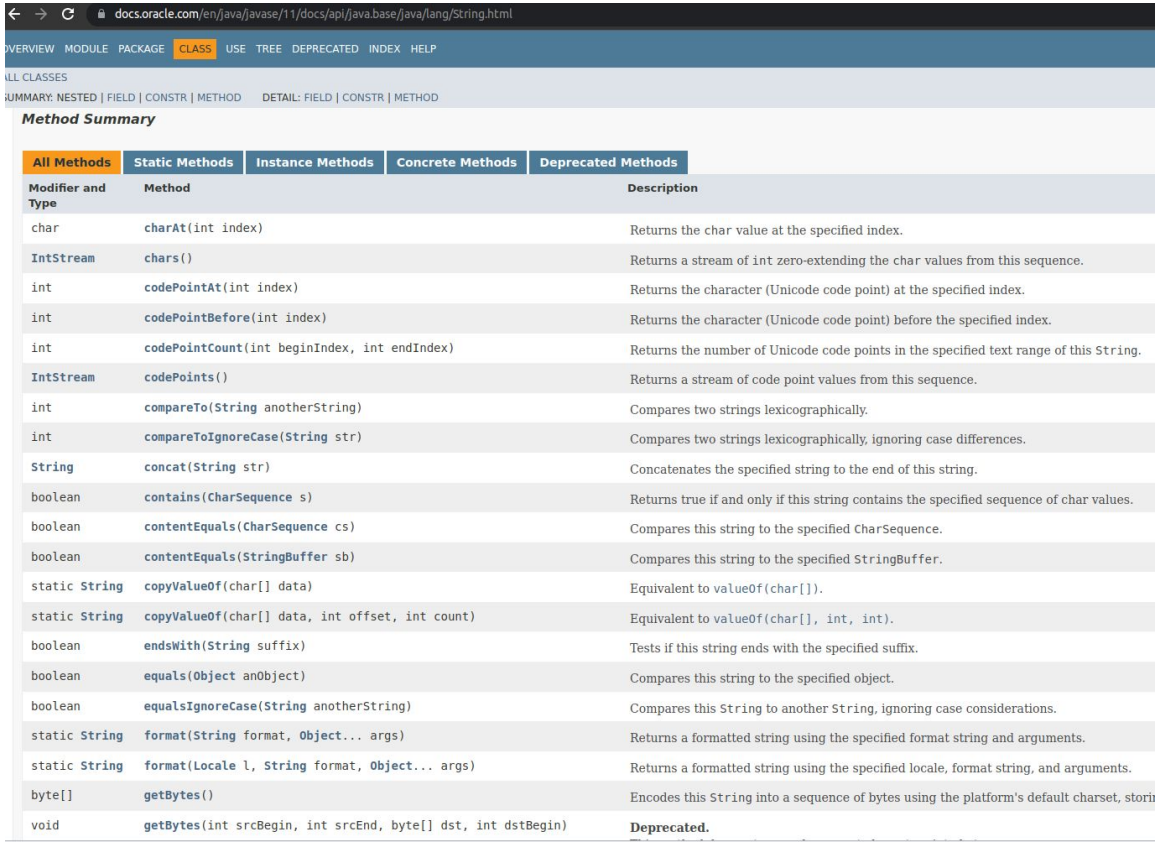
- Java usa el estándar Unicode que asocia a cada símobolo (carácter) un code point (número): char ←→ int

- Se pueden realizar conversiones entre char (2bytes) e int (4bytes).

Ejercicio

Escribir un programa que muestre todos los caracteres Unicode junto a su code point, cuyo valor esté comprendido entre \u0000 y \uFFFF.

String: operaciones



docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

LL CLASSES

UMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods | Deprecated Methods

Modifier and Type	Method	Description
char	charAt(int index)	Returns the char value at the specified index.
IntStream	chars()	Returns a stream of int zero-extending the char values from this sequence.
int	codePointAt(int index)	Returns the character (Unicode code point) at the specified index.
int	codePointBefore(int index)	Returns the character (Unicode code point) before the specified index.
int	codePointCount(int beginIndex, int endIndex)	Returns the number of Unicode code points in the specified text range of this String.
IntStream	codePoints()	Returns a stream of code point values from this sequence.
int	compareTo(String anotherString)	Compares two strings lexicographically.
int	compareToIgnoreCase(String str)	Compares two strings lexicographically, ignoring case differences.
String	concat(String str)	Concatenates the specified string to the end of this string.
boolean	contains(CharSequence s)	Returns true if and only if this string contains the specified sequence of char values.
boolean	contentEquals(CharSequence cs)	Compares this string to the specified CharSequence.
boolean	contentEquals(StringBuffer sb)	Compares this string to the specified StringBuffer.
static String	copyValueOf(char[] data)	Equivalent to valueOf(char[]).
static String	copyValueOf(char[] data, int offset, int count)	Equivalent to valueOf(char[], int, int).
boolean	endsWith(String suffix)	Tests if this string ends with the specified suffix.
boolean	equals(Object anObject)	Compares this string to the specified object.
boolean	equalsIgnoreCase(String anotherString)	Compares this String to another String, ignoring case considerations.
static String	format(String format, Object... args)	Returns a formatted string using the specified format string and arguments.
static String	format(Locale l, String format, Object... args)	Returns a formatted string using the specified locale, format string, and arguments.
byte[]	getBytes()	Encodes this String into a sequence of bytes using the platform's default charset, storin
void	getBytes(int srcBegin, int srcEnd, byte[] dst, int dstBegin)	**Deprecated.**

String.valueOf

Pasa un valor de cualquier tipo a String

static String valueOf (tipo valor)

```
String cad;
cad = String.valueOf(1234); //cad = "1234"
cad = String.valueOf(-12.34); //cad = "-12.34"
cad = String.valueOf('C'); //cad = "C"
cad = String.valueOf(false); //cad = "false"
```

Comprar cadenas

Un error común es comparar dos variables de tipo cadena utilizando el operador de comparación (`==`). Este operador no se puede utilizar con `String` debido a que es una clase y no un tipo primitivo. Por ello, para comparar cadenas usaremos:

- `boolean equals(String otra)`: compara la cadena que invoca el método con otra. El resultado de la comparación se indica devolviendo `true` o `false`, según sean iguales o distintas. Para que las cadenas se consideren iguales deben estar formadas por la misma secuencia de caracteres, distinguiendo mayúsculas de minúsculas. Veamos un ejemplo:

```
String cad1 = "Hola mundo";
String cad2 = "Hola mundo";
String cad3 = "Hola, buenos días"
boolean iguales;
iguales = cad1.equals(cad2); //iguales vale true
iguales = cad1.equals(cad3); //iguales vale false
```

Comparar cadenas

- `int compareTo(String cadena)`: compara alfabéticamente la cadena invocante y la que se pasa como parámetro, devolviendo un entero cuyo valor determina el orden de las cadenas de la forma:
  - 0: si las cadenas comparadas son exactamente iguales.
  - negativo: si la cadena invocante es menor alfabéticamente que la cadena pasada como parámetro, es decir, va antes por orden alfabético.
  - positivo: si la cadena invocante es mayor alfabéticamente que la cadena pasada, es decir, va después.

```
String cad1 = "Alondra";
String cad2 = "Nutria";
String cad3 = "Zorro";
System.out.println(cad2.compareTo(cad1)) //valor mayor que 0
//"Nutria" está después que "Alondra" alfabéticamente
System.out.println(cad2.compareTo(cad3)) //valor menor que 0
//"Nutria" está antes que "Zorro" alfabéticamente
```

Concatenar cadenas

El operador `+` sirve para unir o concatenar dos cadenas. Veamos su funcionamiento con un ejemplo:

```
String nombre = "Miguel";
String apellidos = "de␣Cervantes␣Saavedra";
String nombreCompleto = nombre + apellidos;
System.out.println(nombreCompleto); //"Miguelde Cervantes Saavedra"
```

Obtención de un carácter de la cadena

- `char charAt(int posicion)`: devuelve el carácter que ocupa el índice `posicion` en la cadena que invoca el método. Hay que tener mucha precaución con no utilizar una posición que se encuentre fuera de rango, ya que esto provocará un error y la terminación abrupta del programa. Veamos un ejemplo:

```
String frase = "Nació con el don de la risa";
System.out.println(frase.charAt(4)); //muestra el carácter 'ó'
char c = frase.charAt(30); //¡error! No existe la posición 30
```

Obtención de subcadenas

- `String substring(int inicio)`: devuelve la subcadena formada desde la posición `inicio` hasta el final de la cadena. Lo que se devuelve es una copia y la cadena invocante no se modifica.

```
String cad1 = "Una mañana, al despertar de un sueño intranquilo";
String cad2 = cad1.substring(28); //cad2 vale "un sueño intranquilo"
```

- `String substring(int inicio, int fin)`: hace lo mismo que la anterior, devolviendo la subcadena comprendida entre los índices `inicio` y el anterior a `fin`.

```
String cad1 = "Una␣mañana,␣al␣despertar␣de␣un␣sueño␣intranquilo";
String cad2 = cad1.substring(15, 36); //cad2 = "despertar de un sueño"
```

Longitud de una cadena

- `int length()`: devuelve el número de caracteres (longitud) de una cadena. Una vez conocida la longitud, podemos usar, sin miedo a generar un error, cualquier índice comprendido entre 0 y el índice del último carácter, que es la longitud de la cadena menos 1.

```
int longitud;
String cad1 = "Hola", cad2 = "";
longitud = cad1.length(); //devuelve 4
longitud = cad2.length(); //devuelve 0
```

Búsqueda

- `int indexOf(String cadena)`: sirve para buscar la primera ocurrencia de una cadena.

Comprobaciones

- Vacía
  - public boolean isEmpty()
- Prefijos y sufijos
  - public boolean startsWith(String prefix)
- Contiene
  - public boolean contains(CharSequence s)

Otras operaciones

- Cambiar a mayúsculas y minúsculas
  - toLowerCase(), toUpperCase()
- Reemplazar
  - replace
- Dividir
  - split

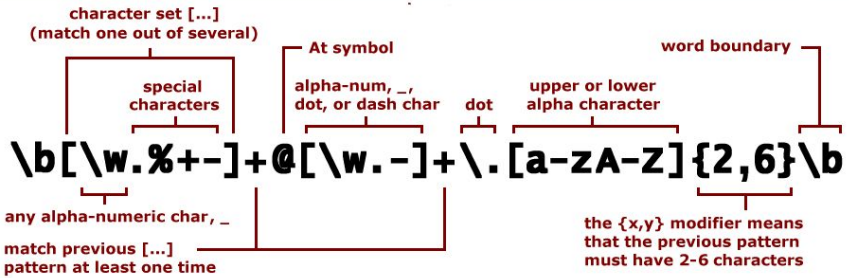
String: Ejercicios

Ejercicios del 16 al 28

Expresiones regulares

Las expresiones regulares son una herramienta para trabajar con cadenas en Java:

- Validar entradas
- Buscar patrones
- Sustituciones en texto



Clases:

- ***Pattern***
- ***Matcher***

Paquete `java.util.regex`

Expresiones regulares

1. Importar el paquete

Primero, necesitas importar el paquete:

```
import java.util.regex.*;
```

2. Crear un patrón

Puedes crear un patrón utilizando el método `compile` de la clase `Pattern`:

```
String regex = "a*b"; // Ejemplo de expresión regular
Pattern pattern = Pattern.compile(regex);
```

3. Crear un matcher

Luego, puedes crear un `Matcher` que se utilizará para buscar coincidencias en una cadena:

```
String input = "aaab";
Matcher matcher = pattern.matcher(input);
```

4. Comprobar coincidencias

Puedes usar varios métodos de la clase `Matcher` para comprobar coincidencias:

- find(): Busca la siguiente coincidencia en la cadena.
- matches(): Comprueba si toda la cadena coincide con el patrón.
- lookingAt(): Comprueba si el comienzo de la cadena coincide con el patrón.

Expresiones regulares: Ejemplo


```
import java.util.regex.*;

public class EjemploRegex {
    public static void main(String[] args) {
        String regex = "a*b"; // Expresión regular
        String input = "aaab"; // Cadena de entrada

        // Compilar el patrón
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(input);

        // Comprobar si hay coincidencias
        if (matcher.matches()) {
            System.out.println("La cadena coincide con el patrón.");
        } else {
            System.out.println("La cadena NO coincide con el patrón.");
        }
// Buscar coincidencias
        if (matcher.find()) {
            System.out.println("Se
encontró una coincidencia: " +
matcher.group());
        } else {
            System.out.println("No se
encontraron coincidencias.");
        }
    }
}
```

Expresiones regulares

Métodos útiles

- **group()**: Devuelve la última coincidencia encontrada.
- **start()**: Devuelve el índice de inicio de la última coincidencia.
- **end()**: Devuelve el índice final de la última coincidencia.

Ejemplos de expresiones regulares

- `\d` : Coincide con un dígito (0-9).
- `\w` : Coincide con un carácter de palabra (letras, dígitos y guiones bajos).
- `\s` : Coincide con un espacio en blanco.
- `.` : Coincide con cualquier carácter excepto un salto de línea.
- `^` : Indica el inicio de una cadena.
- `$` : Indica el final de una cadena.

Recursos y referencias

- Libro que tenéis en la biblioteca y en el departamento
- Diapositivas en Moodle
- Ejercicios de refuerzo y de ampliación
- Vuestros apuntes (portfolio)
- Examen: ¿miércoles 20?
- Proyecto: ¿?