UNIDAD 5: ESTRUCTURAS DE DATOS ESTÁTICAS

Profesor: José Ramón Simó Martínez

Contenido

1. Introducción

Los arrays constituyen una de las estructuras de datos estáticas más recurridas en lenguajes de alto nivel. Es por ello que en esta unidad trataremos principalmente este tipo de estructura junto con todas sus operaciones de manipulación (creación, borrado y actualización) y búsqueda.

Los programas informáticos habitualmente tratan con datos de tipo texto y podemos utilizar arrays de caracteres para tratarlos. Sin embargo, en Java ya existe la clase *String* para crear y manipular cadenas de texto de forma más eficiente. Es por ello que en esta unidad estudiaremos las propiedades de este objeto.

No obstante, antes que nada, presentaremos en esta unidad un conjunto de clases útiles de Java que ayudarán a enriquecer las funcionalidades de nuestros programas.

2. Librerías de clases útiles

2.1. La clase Math

Ya vimos en unidades anteriores referencias a esta clase. En este apartado conoceremos algunos de los métodos de cálculo matemático que la clase Math nos ofrece.

En Java disponemos de muchas funciones matemáticas predefinidas que nos ayudan ha obtener el resultado, por ejemplo:

- Del valor absoluto de un número
- De la potencia de números (un número elevado a otro)
- Redondeo de números decimales
- Logaritmos
- etc.

Sin embargo, en este apartado sólo aprenderemos a utilizar la clase `Math` para:

- Calcular potencias
- Calcular raíz cuadrada

En las siguientes unidades iremos ampliando su uso.

`Math` es una clase de java y podemos utilizar sus métodos al igual que hemos hecho con la clase `Scanner`; por ejemplo, en `Scanner` tenemos los métodos `nextInt()`, `nextFloat()`, etc. Además, para utilizar estos métodos en la clase `Scanner` hacemos, por ejemplo:

```
Scanner sc = new Scanner(System.in);

sc.nextInt(); // así podemos utilizar el método nextInt() de Scanner
```

No obstante, para utilizar un método de la clase `Math` no hace falta crear una variable y hacer un `new`. Asimismo, para acceder a un método de esta clase deberemos escribir *Math.nombredelmétodo*, por ejemplo:

```
Math.pow(2,3); // Nos da el resultado de 2 elevado a 3

Math.sqrt(9.0); // Nos da el resultado de la raíz cuadrada de 9
```

> **Nota**
>
> La clase Math pertenece al paquete java.lang y por tanto no hace falta importarla como sí hacíamos con la clase Scanner.

2.1.1. Calcular potencias

`Math.pow(a,b)` devuelve el resultado de la operación de a^b y por tanto podemos almacenar este valor en una variable; pero atención, esta variable debe ser de tipo `double`. Por ejemplo:

```
double resultado = Math.pow(2,3);
```

También podemos hacer el cálculo directamente en la salida del programa:

```
System.out.println(Math.pow(2,3));
```

> **Nota**
>
> En Math.pow(a, b) los valores a y b pueden ser números con decimales o variables tipo double.

2.1.2. Calcular raíz cuadrada

`Math.sqrt(a)` devuelve el resultado de la operación de $\sqrt{a}$ y por tanto podemos almacenar este valor en una variable; pero atención, esta variable debe ser de tipo `double`. Por ejemplo:

```
double resultado = Math.sqrt(9);
```

También podemos hacer el cálculo directamente en la salida del programa:

```
System.out.println(Math.sqrt(9));
```

Un ejemplo más completo de uso de la clase Math sería el siguiente:

```
public class Ejemplo3101
{
    public static void main (String[] args)
    {
        double resultado1, resultado2;

        resultado1 = Math.pow(2,3); // 2 elevado a 3
        resultado2 = Math.sqrt(9); // raíz cuadrada de 9

        System.out.println("El resultado de 2 elevado a 3 es: " + resultado1);
        System.out.println("El resultado de 2 elevado a 3 es: " + resultado2);
    }
}
```

Nota

En Math.sqrt(a) el valor de a puede ser un número con decimales o una variable tipo double.

2.1.3. Otros métodos de la clase math

Un resumen de algunos métodos de la clase Math:

Método	Descripción	Ejemplo de uso	Resultado
abs	Devuelve el valor absoluto de un numero.	int x = Math.abs(2.3);	x = 2;
ceil	Devuelve el entero más cercano por arriba.	double x = Math.ceil(2.5);	x = 3.0;
floor	Devuelve el entero más cercano por debajo.	double x = Math.floor(2.5);	x = 2.0;
round	Devuelve el entero más cercano.	double x = Math.round(2.5);	x = 3.0;
log	Devuelve el logaritmo natural en base e de un número.	double x = Math.log(2.71);	x = 0.9996;
max	Devuelve el mayor de dos entre dos valores.	int x = Math.max(3, 8);	x = 8;
min	Devuelve el menor de dos entre dos valores.	int x = Math.min(3, 8);	x = 3;
random	Devuelve un número aleatorio entre 0 y 1. Se pueden cambiar el rango de generación.	double x = Math.ramdom();	x = 0.206178;
sqlrt	Devuelve la raíz cuadrada de un número.	double x = Math.sqlrt(9);	x = 3.0;
pow	Devuelve un número elevado a un exponente.	double x = Math.pow(2, 10);	x= 1024.0;
…	…	…	…
MÉTODO	DESCRIPCIÓN	Ejemplo de uso	resultado
abs	Devuelve el valor absoluto de un numero.	int x = Math.abs(2.3);	x = 2;
ceil	Devuelve el entero más cercano por arriba.	double x = Math.ceil(2.5);	x = 3.0;
floor	Devuelve el entero más cercano por debajo.	double x = Math.floor(2.5);	x = 2.0;
round	Devuelve el entero más cercano.	double x = Math.round(2.5);	x = 3.0;
log	Devuelve el logaritmo natural en base e de un número.	double x = Math.log(2.71);	x = 0.9996;
max	Devuelve el mayor de dos entre dos valores.	int x = Math.max(3, 8);	x = 8;
min	Devuelve el menor de dos entre dos valores.	int x = Math.min(3, 8);	x = 3;
random	Devuelve un número aleatorio entre 0 y 1. Se pueden cambiar el rango de generación.	double x = Math.ramdom();	x = 0.206178;
sqlrt	Devuelve la raíz cuadrada de un número.	double x = Math.sqlrt(9);	x = 3.0;
pow	Devuelve un número elevado a un exponente.	double x = Math.pow(2, 10);	x= 1024.0;
…	…	…	…

Para consultar el resto de métodos de la clase Math:

https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/Math.html

2.2. La clase Random

La clase Random nos permite generar números aleatorios. Esto nos puede servir para aplicarlo a los siguientes ejemplos:

- El resultado de tirar un dado en un juego.
- El sorteo de la lotería.
- Generar claves encriptadas
- Simular fenómenos físicos reales
- etc.

Esta clase, a diferencia de la clase Math, necesita crear un objeto para utilizarla. En nuestra primera aproximación a la creación de objetos en Java, por tanto, tenemos la clase Random. Un objeto de la clase Random se crea así:

```
Random rand = new Random();
```

Para usar la clase Random debemos importarla así en la cabecera de nuestro código:

```
import java.util.Random;
```

Hay cuatro funciones miembro diferentes que generan números aleatorios:

Función miembro	Descripción	Rango
r.nextInt()	Número aleatorio entero de tipo int	2^{-32} y 2^{32}
r.nextLong()	Número aleatorio entero de tipo long	2^{-64} y 2^{64}
r.nextFloat()	Número aleatorio real de tipo float	[0,1[
r.nextDouble()	Número aleatorio real de tipo double	[0,1[

En caso de necesitar números aleatorios enteros en un rango determinado, podemos trasladarnos a un intervalo distinto, simplemente multiplicando, aplicando la siguiente fórmula general:

```
(int)(rand.nextDouble()*cantidad_números_rango + término_inicial_rango)
```

donde (int) al inicio, transforma un número decimal double en entero int, eliminando la parte decimal.

Por ejemplo, si deseamos números aleatorios enteros comprendidos entre [1,6], que son los lados de un dado, la fórmula quedaría así.

```
(int)(rnd.nextDouble() * 6 + 1);
```

donde 6 es la cantidad de números enteros en el rango [1,6] y 1 es el término inicial del rango.

Más información sobre la clase Random:

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Random.html

2.3. Clases envoltorio (wrapper)

Clases envoltorio (*wrapper*)

- En ocasiones es útil tratar los tipos de datos básicos como objetos.
  `int, byte, short, long, char, boolean, float, double`
- Muchas funciones y clases trabajan con elementos que heredan de la clase Object (la clase que se sitúa en la parte más alta de la jerarquía de objetos en Java).
  No funcionarán directamente con estos tipos básicos.
- Existe una clase envoltorio por cada tipo básico.
- Cada una tiene un único atributo, que es del tipo básico al que "envuelven".

Tipo básico	Clase envoltorio
int	Integer
char	Character
boolean	Boolean
long	Long
double	Double
float	Float
short	Short
byte	Byte

A continuación, veremos la clase Integer como ejemplo de una clase envoltorio en Java:

Constantes

```
int max = Integer.MAX_VALUE;
int min = Integer.MIN_VALUE;
```

Métodos

```
//Pasar de INT a String
int a1 = 45678;
String a2 = Integer.toString( a1 );
//Pasar de String a INT
String b1 = "45678";
int b2 = Integer.parseInt( b1 );
```

```
int b3 = Integer.parseInt(CharSequence s, int beginIndex, int
endIndex, int radix)
```

Ejemplo:

```
cadena = sc.next(); // Lee la siguiente cadena: "13-14"
String[] separada = cadena.split("-");
a = Integer.parseInt( separada[0] ); // Pasar el 13 de texto a número
b = Integer.parseInt( separada[1] ); // Pasar el 14 de texto a número
```

Más información sobre la clase Integer:

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Integer.html

2.3.1. Boxing y unboxing automáticos

Desde la versión 5 de Java, se convierte automáticamente entre las clases envoltorios y sus correspondientes tipos básicos.

- Si se introduce un tipo básico donde se espera un objeto de una clase envoltorio, se llama al constructor correspondiente (boxing).
- Si se introduce un objeto de una clase envoltorio donde se espera un tipo básico, se llama al método de acceso correspondiente (unboxing).

Por ejemplo:

```
Integer x = 5, y = 9;  //Boxing
int z = x + y;    //Unboxing
System.out.printf("%s + %s = %d", x, y, z);
```

2.4. La clase Date

La clase Date nos permite manejar fechas y tiempo. Esta clase es otro objeto de Java al igual que la anterior clase Random. Para crear este objeto haremos lo siguiente:

```
Date nombreVariable = new Date();
```

Para usar la clase Date debemos importarla así en la cabecera de nuestro código:

```
import java.util.Date;
```

Por ejemplo, para obtener la fecha actual del sistema:

```
Date ahora = new Date();

System.out.println(ahora);
```

Si lo hubiéramos ejecutado el 7 de Noviembre de 1977 a las 22:00 horas, nos hubiera mostrado como salida del programa lo siguiente:

```
Mon Nov 04 22:00:00 CET 1977
```

2.4.1. formateado de fecha y hora

Si quisiéramos personalizar la salida por pantalla de la fecha y hora obtenida, podemos utilizar la clase `SimpleDataFormat` y sus métodos, por ejemplo el método format. Ejemplo:

```
SimpleDateFormat formatoDelTexto = new SimpleDateFormat("dd/MMMMM/yyyy
hh:mm:ss");

System.out.println("Fecha y hora actual: " + formatoDelTexto.format(ahora));
```

Mostraría lo siguiente:

```
Fecha y hora actual: 04/noviembre/1977 22:00:00
```

Para usar la clase SimpleDataFomat debemos importarla así en la cabecera de nuestro código:

```
import java.text.SimpleDataFormat;
```

2.4.2. Uso de la clase GregorianCalendar

Una manera de manejar la fecha y hora que obtenemos de la clase Date es utilizando la clase GregorianCalendar.

```
Date d = new Date();

System.out.printf("Tiempo: %d\n", d.getTime()); // Método de Date
System.out.printf("Equivale a: %s\n", d.toString());
GregorianCalendar gc = new GregorianCalendar();

gc.setTime(d);

System.out.printf("La fecha actual es: %d/%d/%d",
gc.get(GregorianCalendar.DAY_OF_MONTH),
gc.get(GregorianCalendar.MONTH) + 1,
gc.get(GregorianCalendar.YEAR));
```

Para usar la clase SimpleDataFomat debemos importarla así en la cabecera de nuestro código:

```
import java.util.GregorianCalendar;
```

2.4.3. ejemplo de uso de fecha y horas

A continuación, veremos unos ejemplos de uso de esta clase:

```
/* 1. Método usado para obtener la fecha y hora actual del sistema de forma
personalizada. */

// Creamos el objeto Date
```

```
Date ahora = new Date()

SimpleDateFormat formatoDelTexto = new SimpleDateFormat("dd/MMMMM/yyyy
hh:mm:ss");
System.out.println("Fecha y hora actual: " + formatoDelTexto.format(ahora));

formatoDelTexto = new SimpleDateFormat("dd-MM-yyyy");
System.out.println("Fecha actual: " + formatoDelTexto.format(ahora));

formatoDelTexto = new SimpleDateFormat("hh:mm:ss");
System.out.println("Hora  actual: " + formatoDelTexto.format(ahora));

formatoDelTexto = new SimpleDateFormat("dd 'de' MMMM 'de' yyyy", new
Locale("ES"));
System.out.println("Fecha con el nombre del mes: " +
formatoDelTexto.format(ahora));

formatoDelTexto = new SimpleDateFormat("EEEE',' dd 'de' MMMM 'de' yyyy", new
Locale("ES"));
System.out.println("Fecha completa: " + formatoDelTexto.format(ahora));


/* 2. Preguntar si una fecha es posterior o anterior a otra. */
System.out.println("¿Es fechaSuma posterior a fechaResta?: " +
fechaSuma.after(fechaResta));

System.out.println("¿Es fechaSuma anterior  a fechaResta?: " +
fechaSuma.before(fechaResta));

/* 3. Súmale 7 días a la fecha de hoy. */
int dias = 20;
Date fecha = new Date();

Calendar cal = new GregorianCalendar();
cal.setTimeInMillis( fecha.getTime() );
cal.add(Calendar.DATE, dias);

Date fechaSuma = new Date( cal.getTimeInMillis() );

formatoDelTexto = new SimpleDateFormat("dd-MM-yyyy");
System.out.println("Hoy + 20 dias: " + formatoDelTexto.format( fechaSuma
) + " (fechaSuma)");

/* 4. Réstale 15 días a una fecha determinada. */
dias = 15;
cal.setTimeInMillis( fecha.getTime() );
cal.add(Calendar.DATE, -dias);

Date fechaResta = new Date( cal.getTimeInMillis() );

formatoDelTexto = new SimpleDateFormat("dd-MM-yyyy");
System.out.println("Hoy - 15 dias: " + formatoDelTexto.format(
fechaResta ) + " (fechaResta)");
```

```
/* 5. Calcular la diferencia entre dos fechas.*/
long horas          = 0;
long minutos      = 0;
long segundos   = 0;

//Calcular la diferencia entre fechas
diferencia = fechaSuma.getTime() - fechaResta.getTime();

//Calcular la diferencia en DIAS y mostrarla por pantalla
System.out.println("La diferencia en dias entre fechaSuma y fechaResta
es: " + diferencia/(1000 * 60 * 60 * 24));

//Calcular la diferencia en HORAS, MINUTOS Y SEGUNDOS y mostrarla por
pantalla
segundos = diferencia / 1000;
horas = segundos / 3600;
segundos = segundos - (horas * 3600);
minutos = segundos / 60;
segundos = segundos - (minutos * 60);

System.out.println("La diferencia en Horas Minutos y Segundos entre
fechaSuma y fechaResta es: " + horas + "h " + minutos + "min " +
segundos + "seg");

/* 6.- Crear una fecha determinad. Nota: el uso de la estructura
try…catch se estudiará en unidades posteriores. Por ahora, se necesita
usar en este tipo de estructura.*/

int día = 7;
int mes = 11;
int ano = 1977;

//Crear la cadena de texto con la fecha del usuario
String fechaString = dia + "-" + mes + "-" + ano;

//Indicar el formato de la fecha
formatoDelTexto = new SimpleDateFormat("dd-MM-yyyy");

Date fechaUsuario = null;
try {
  //Pasar de texto a fecha
  fechaUsuario = formatoDelTexto.parse( fechaString );
  System.out.println(“Fecha usuario:” + formatoDelTexto.format(
fechaUsuario ));
} catch (ParseException e) {
  e.printStackTrace();
  System.out.println(“Fecha INCORRECTA”);
}
```

3. Los arrays

¿Por qué necesitamos los arrays? Los arrays, también conocidos con el nombre de vectores, son elementos que almacenan de manera estructurada un conjunto de valores que pertenecen al mismo tipo de dato (entero, real, carácter, objetos, etc).

Muchas de las soluciones a nivel computacional requieren crear listas de datos. Por ejemplo, si necesitamos almacenar y manipular las notas de 5 alumnos, deberíamos declarar 10 variables:

```
int nota1, nota2, nota3, nota4, nota5;
```

¿Pero qué ocurre si en vez de 5 alumnos son 500 empleados de una multinacional? Evidentemente con la solución anterior nuestro programa sería inviable de tratar y mantener. En este caso, deberíamos de utilizar las estructuras de arrays.

3.1. Declaración y acceso a arrays

3.1.1. Declaración de arrays

Con un array podemos manipular una sola variable pero que a la vez está tratando con múltiples datos. En el caso de los alumnos deberíamos declarar un array de 5 enteros para almacenar cada una de sus notas:

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

Por tanto, podemos deducir que la sintaxis para declarar un array de un tamaño fijo es la siguiente:

```
tipo_de_variable[] nombre_de_variable = new tipo_de_variable[tamaño_del_array];
```

ó

```
tipo_de_variable nombre_de_variable[] = new tipo_de_variable[tamaño_del_array];
```

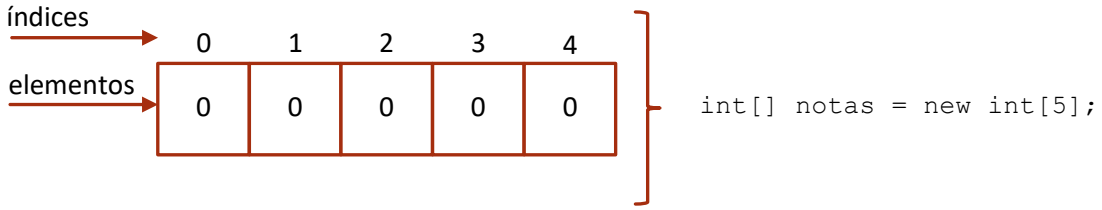
El estilo más habitual es el del primer caso ya que al declarar int[] y ver los corchetes ya puedes interpretar que la variable es de tipo array. En cualquier caso, debes elegir el estilo en el que más te sientas a gusto.

> **Nota:**
>
> En esta unidad estamos tratando con los arrays estáticos, es decir, tiene un número fijo de datos que pueden almacenar. En unidades posteriores estudiaremos las clases que Java ofrece para tratar con arrays dinámicos.

3.1.2. Almacenar y recuperar elementos de un array

Un array se puede visualizar como una lista de elementos donde cada elemento ubica en una posición determinada y localizable llamada índice. Además, en el caso de los arrays de tipos primitivos, cuando se declara el array este inicializa a cero todos sus elementos.

El array de notas de alumnos que hemos creado anteriormente se podría visualizar de la siguiente manera:



Hay que tener en cuenta que los índices del array siguen estas reglas:

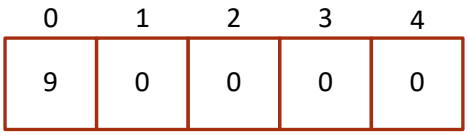
- Primer elemento = índice 0
- Último elemento = tamaño del array – 1

En el ejemplo anterior el tamaño del array es 5 y por tanto el último elemento se sitúa en el índice 4. También podemos hablar de que el primer elemento está en la posición 0 y el último elemento está en la posición 4.

Si quisiéramos almacenar un 9 en la primera posición del array de notas deberíamos escribir la siguiente instrucción:

```
notas[0] = 9;
```

El contenido del array se actualizaría y se vería así:



Podríamos hacer los mismo con todos los elementos del array. Por ejemplo, el código completo en el que se declara el array de notas y se almacenan los valores correspondientes sería:

```
// Creamos el array para guardar 5 notas
int[] notas = new int[5];

// Guardamos cada nota en cada elemento del array
notas[0] = 9;
notas[1] = 7;
notas[2] = 8;
notas[3] = 5;
notas[4] = 3;
```

Y finalmente el contenido de array de notas:

0	1	2	3	4
9	7	8	5	3

Por otra parte, también queremos recuperar alguno de los valores almacenados en el array. En este caso, nos puede interesar recuperar las notas de los alumnos para poder hacer alguna operación aritmética con ellas, como por ejemplo calcular la nota media.

```
int notaAlumno1 = notas[0];
```

Con la anterior instrucción recuperamos la nota guardada en la primera posición del array y la almacenamos en una variable llamada notaAlumno1. Puedes entender mejor este concepto a partir del siguiente ejemplo.

Ejemplo: Calcular la nota media de 5 alumnos de clase

```
public static void main (String[] args)
{
    // Creamos el array para guardar 5 notas
    int[] notas = new int[5];

    // Guardamos cada nota en cada elemento del array
    notas[0] = 9;
    notas[1] = 7;
    notas[2] = 8;
    notas[3] = 5;
    notas[4] = 3;

    float suma = notas[0] + notas[1] + notas[2] + notas[3] + notas[4];

    float media = suma / 5;

    System.out.println("La nota media de clase es: " + media);
}
```

3.1.3. Asignar valores iniciales en la declaración del array

Puede que necesitemos declarar el array con unos valores iniciales, bien por los conozcamos a priori o bien porque los necesitemos en la solución de nuestro problema.

Siguiendo con el ejemplo de las notas el array se declararía e inicializaría con la siguiente instrucción:

```
int[] notas = new int[] {9,7,8,5,3};
```

ó

```
int[] notas = {9,7,8,5,3};
```

Como ves el segundo caso es más simple y es estilo más habitual que utilizaremos.

Podemos modificar el anterior ejemplo pero con el array ya rellenado de notas en su propia declaración:

```
public static void main (String[] args)
{
    // Declaramos e iniciamos el array con 5 notas
    int[] notas = {9,7,8,5,3};

    float suma = notas[0] + notas[1] + notas[2] + notas[3] + notas[4];

    float media = suma / 5;

    System.out.println("La nota media de clase es: " + media);
}
```

3.1.4. Recorrer los elementos de array: bucle foreach

La suma de cada uno de los valores del array de nuestro ejemplo puede resultar algo engorrosa y más aún en el caso de que tuviéramos que sumar las notas de 50 alumnos, por ejemplo.

Para ello es mejor utilizar una estructura repetitiva y habitualmente una buena opción sería el bucle for.

Por ejemplo, si modificamos el código anterior para que sume las notas una a una lo haga con un bucle for, el fragmento de código que haría esto quedaría así:

```
// Recorremos el array con un for
for (int i = 0; i < 5; i++)
    suma += notas[i];
```

Sin embargo, en Java existe la estructura de control `foreach` la cual está diseñada para recorrer y recuperar elementos de un array de una forma más eficiente (o sencilla).

El código equivalente al ejemplo anterior con la estructura `foreach` sería:

```
// Recorremos el array con un foreach
for (int notaAlumno : notas)
    suma += notaAlumno;
```

La sintaxis de este bucle es:

```
for (tipo_de_dato elemento: array)
{
   // Procesar el elemento
}
```

En la primera iteración del bucle se guarda notas[0] en notaAlumno y se suma, en la segunda iteración se guarda notas[1] en notaAlumno y se suma, y así sucesivamente hasta llegar al último elemento del array y termina el bucle.

> **Nota**
>
> En otros lenguajes como C#, PHP o Javascript la sintaxis para bucle foreach se utiliza el nombre foreach para la estructura. Sin embargo, cabe tener en cuenta que en Java la palabra clave no se llama foreach si no for. Utilizamos la denominación foreach para diferenciarlo del bucle for estándar.

3.1.5. Los límites del array: uso de constantes y el método length

En los arrays estáticos debemos tener en cuenta el tamaño que le hemos dado a nuestro array. Un array tiene por tanto un rango limitado de índices de acceso según su tamaño; si N es el tamaño de nuestro array el rango será desde el índice 0 hasta N-1. Si accedemos a un índice que esté fuera de este rango, nuestro programa compilaría, no obstante, nos daría el siguiente error en la posterior ejecución del programa y terminaría:

```
java.lang.ArrayIndexOutOfBoundsException
```

En el ejemplo que estamos desarrollando, si escribimos la siguiente instrucción:

```
notas[5] = 10;
```

Tendríamos el problema descrito, ya que el tamaño de nuestro array es de 5 y sólo podemos acceder a los índices de 0 a 4.

A la hora de recorrer el array debemos tener en cuenta que no accedemos fuera del rango de valores disponibles. Podemos utilizar tres técnicas:

- Definir una constante con el tamaño máximo del array.
- Utilizar el método `length` de la clase array que nos da su tamaño.
- Una mezcla de las dos anteriores.

Un ejemplo utilizando la primera técnica:

```
public static void main (String[] args)
{
    final int MAX_ALUMNOS  = 5;

    int[] notas = new int[MAX_ALUMNOS];

    Scanner sc = new Scanner(System.in);

    for (int i = 0; i < MAX_ALUMNOS; i++)
        notas[i] = sc.nextInt();
}
```

Un ejemplo utilizando la segunda técnica:

```
public static void main (String[] args)
{
    int[] notas = new int[5];

    Scanner sc = new Scanner(System.in);

    for (int i = 0; i < notas.length; i++)
        notas[i] = sc.nextInt();

}
```

Y por último, podemos hacer una mezcla de las dos anteriores:

```
final int MAX_ALUMNOS = 5;

int[] notas = new int[MAX_ALUMNOS];

Scanner sc = new Scanner(System.in);

for (int i = 0; i < notas.length; i++)
    notas[i] = sc.nextInt();
```

De esta forma, si necesitamos modificar el tamaño del array es más limpio e intuitivo hacer desde la constante declarada y no dentro del array. Además, más adelante podemos utilizar el método `length` que nos asegura que no nos saldremos del rango del array por la parte superior.

> Nota
>
> Cuidado, los índices de un array no pueden ser negativos y por tanto debemos ser precavidos en el valor inicial de la variable contador del bucle.

Otro ejemplo del uso del método `length` de la clase array sería para el ejemplo del cálculo de la media de los alumnos, ya que para obtener la media debemos saber el total de alumnos:

```
public static void main (String[] args)
{
    int[] notas = {9,7,8,5,3};

    float suma = notas[0] + notas[1] + notas[2] + notas[3] + notas[4];

    float media = suma / notas.length; // utilizo la propiedad length

    System.out.println("La nota media de clase es: " + media);
}
```

Un resumen de lo estudiado en el apartado 4.3:

- Un array nos permite crear y manipular listas de datos, como por ejemplo, almacenar las notas de los alumnos de una clase.

- Se pueden crear arrays de cualquier tipo de dato: int, float, double, etc. Sin embargo, todos los elementos del mismo arrays deben ser del mismo tipo.
- Un array se puede declarar vacío (todo a cero) o con unos valores iniciales.
- Los arrays que estamos estudiando se conocen como estáticos, ya que se definen previamente con un tamaño limitado. Hay que tener cuidado con el rango de valores de un array.
- Los arrays se recorren con un bucle for o foreach.
- Resumen sintaxis con el ejemplo del array de notas:

Tipo Java	Rango de valores
Crear un array de tamaño 5:	`int [] notas  = new int[5]`
Guardar una nota en la primera posición:	`notas[0] = 9;`
Guardar una nota en la última posición:	`notas[4] = 3;`
Obtener el último elemento del array:	`int notaAlumno1 = notas[4];`
Obtener el tamaño del array:	`int numeroAlumnos = notas.length;`

3.2. Operaciones más comunes con arrays

3.2.1. Elemento máximo o mínimo de un array

Para encontrar el máximo o el mínimo de los elementos de un array, tomaremos el primero de los elementos como valor provisional, y compararemos con cada uno de los demás, para ver si está por encima o debajo de ese máximo o mínimo provisional, y actualizarlo si fuera necesario.

El siguiente fragmento de código muestra el algoritmo descrito para calcular el elemento máximo de un array:

```
int notaMax = notas[0]; // tomamos 1er. elemento como máximo
for (int i = 1; i < notas.length; i++)
{
    if (notas[i] > notaMax)
        notaMax = notas[i];
}
```

Cuidado, un error típico es inicializar la variable del máximo directamente a 0:

```
int notaMax = 0; // Error!
```

3.2.2 Copia de un array

A veces, en un programa, necesitamos duplicar un array o parte de un array. En estos casos, podrías tener la tentación de asignar un array a otro array. Por ejemplo, si tenemos dos arrays llamados notas1 y notas2:

notas2 = notas1; // ¡Error! Haciendo esto no se copia

Sin embargo, esta instrucción no copia el contenido de un array sobre otro. El motivo de que esto no funcione lo explicaremos en los temas de orientación a objetos que veremos en las siguientes unidades. Por ahora, debes entender que no debes hacer esto para copiar dos arrays.

Evidentemente sí que se pueden copiar arrays, pero debes hacer siguiendo algunas de estas tres técnicas:

- Usar un bucle para copiar elemento a elemento de un array a otro.
- Usar el método `arraycopy` de la clase `System`.
- Usar el método `clone` para copiar arrays. Este método lo estudiaremos en unidades posteriores.

Para copiar elemento a elemento de un array a otro podemos utilizar el siguiente bucle:

```
for (int i = 1; i < notas.length; i++)
    notas2[i] = notas1[i]
```

Y con el método `arraycopy` lo deberíamos hacer con la siguiente instrucción:

```
System.arraycopy(notas1,0,notas2,0,notas1.length);
```

Esta instrucción la podríamos leer así: “Copia el array notas1 (notas1) desde el inicio (0) hacia array notas2 (notas2) desde su inicio (0) y copia notas1 completamente (notas1.length)”.

La sintaxis del método `arraycopy` es:

```
System.arraycopy (array_origen, índice_origen, array_destino, elementos_a_copiar);
```

> Nota:
>
> Hay que tener en cuenta que en ambos ejemplos el tamaño del array notas2 debe ser menor o igual que notas1.

3.2.3. Insertar elementos en el array

Un elemento se puede insertar en un array de tres formas:

- Al final del array.
- En posiciones intermedias.

Insertar elementos al final del array

Para insertar un elemento al final del array es lo más sencillo. Sólo debes tener en cuenta el índice del último elemento que se ha insertado en el array. Sin embargo, debemos comprobar si el array está lleno, ya que en tal caso no podremos insertar un elemento nuevo en ese array a no ser que lo redimensionemos.

Por ejemplo:

```
if (cantidad < capacidad)
{
    notas[cantidad] = 5;
    cantidad++;
}
```

Hasta ahora hemos rellanado completamente nuestro array con un bucle for y por tanto la comprobación anterior no hace falta. Este ejemplo se aplica cuando tenemos un array que no está lleno y vamos insertando elementos en él en cualquier momento del programa.

Insertar elementos en posiciones intermedias

En este caso, y siempre que haya espacio libre en el array, debemos desplazar todos los elementos hacia la derecha desde la posición donde queramos insertar el nuevo elemento. Finalmente, insertaremos el nuevo elemento en dicha posición.

La clave está en que este movimiento de desplazamiento debe empezar desde el final para que cada elemento que se mueve no sobreescriba el que estaba a continuación de él. Además, debemos actualizar el contador de elementos, para indicar que hay el array tiene un elemento más.

El algoritmo sería el siguiente:

```
for (i = cantidad; i > posicionInsertar; i--)
    notas[i] = notas[i-1];
notas[posicionInsertar] = 9;
cantidad++;
```

3.2.4. Borrar elementos de un array

Si queremos borrar el elemento que hay en una cierta posición de un array, los que estaban a continuación deberán desplazarse "hacia la izquierda" para que no queden huecos. Como en el caso anterior, deberemos actualizar el contador, pero ahora para indicar que el array tiene un elemento menos.

```
for (i = posicionBorrar; i < cantidad-1; i++)
    notas[i] = notas[i+1];
cantidad--;
```

> Nota:
>
> En este algoritmo se supone que posicionBorrar puede ser 0 hasta la cantidad de elementos que hay en el array menos 1.

3.2.5. Buscar elementos en un array

Existen dos formas de buscar un elemento dentro de un array:

- Búsqueda lineal
- Búsqueda binaria o dicotómica

Búsqueda secuencial o lineal

El algoritmo para buscar elementos en un array de forma secuencial es el más sencillo e intuitivo. Simplemente deberemos comparar cada elemento del array con el elemento a buscar.

En caso de encontrar el elemento podríamos:

- Notificar al usuario que se ha encontrado el elemento.
- Almacenar la posición en la que se ha encontrado el elemento.
- Almacenar el éxito de haber encontrado el elemento.

El ejemplo siguiente hace referencia al segundo caso:

```
int[] notas1 = {9,7,8,5,3};

// Búsqueda lineal
int elementoBuscado = 8;
int posicionBuscado = -1;

for (int i = 0; i < notas1.length; i++)
    if (notas1[i] == elementoBuscado)
        posicionBuscado = i;
```

La variable posicionBuscado la iniciamos a -1 para indicar que en un principio no hemos encontrado el elemento dentro del array. En caso de no encontrarlo, posicionBuscado seguirá valiendo -1.

El anterior algoritmo no es muy eficiente, ya que en caso de encontrar el elemento sigue recorriendo el array hasta el final. Efectivamente, la idea sería que el bucle terminara cuando se haya encontrado el elemento. ¿Cómo lo harías?

Búsqueda binaria o dicotómica

El algoritmo de búsqueda binaria o dicotómica es un poco más complejo, pero es mucho más eficiente que la búsqueda secuencial. A priori el array debe estar ordenado. El array se dividirá en dos para buscar el elemento en una parte del array o en otra y así sucesivamente hasta encontrar, o no, el elemento.

En el siguiente ejemplo aplicamos el algoritmo de búsqueda binaria al array de notas que hemos estado utilizando en los anteriores ejemplos. Cabe destacar que el array de notas debe estar necesariamente ordenado antes de iniciar la búsqueda:

```
public static void main (String[] args)
{
    int[] notas = {3,5,7,8,9};

    int inferior = 0;
    int superior = notas.length - 1;
    int centro;

    int posicion = -1;

    int elementoBuscado = 5;
    while(inferior <= superior && posicion == -1)
    {
        centro = (inferior+superior)/2;

        if (elementoBuscado == notas[centro])
            posicion = centro;
        else if (elementoBuscado < notas[centro])
            superior = centro - 1;
        else if (elementoBuscado > notas[centro])
            inferior = centro + 1;
    }

    System.out.println("Elemento buscado: " + elementoBuscado);
    System.out.println("Encontrado en posición: " + posicion);
}
```

3.2.6 Ordenación de arrays

Para terminar con las operaciones más habituales sobre los arrays, cabe comentar alguno de los algoritmos que sirven para ordenar los elementos de un array:

- Burbuja
- Inserción
- Selección
- Quicksort

Aunque estos algoritmos se estudian en profundidad en niveles universitarios, en el caso de los CFGS DAM/DAW en el módulo de Programación no entraremos en más detalle. Java ya tiene implementados estos algoritmos y podemos utilizar sus métodos de ordenación muy fácilmente (por ejemplo, con el método sort del array).

Sin embargo, para aquellos y aquellas que tengan interés en saber más sobre los algoritmos de ordenación, os dejo el siguiente enlace:

https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento

3.3. La clase Arrays

Aunque no hayamos entrado en los conceptos de la programación orientada a objetos, durante el curso ya hemos tratado de manejar algunos objetos, sus propiedades y métodos, por ejemplo:

- La clase `System`: System.out.println()
- La clase `Scanner` y sus métodos nextInt(), nextFloat()
- La clase `Math` con sus métodos (pow, random...) y propiedades (Math.PI).
- etc.

De la misma manera los arrays tiene unos métodos, o herramientas si lo quieres pensar así, que nos simplifican la elaboración de los programas. Estos se encuentran en la clase `Arrays`.

Por ejemplo, si queremos ordenar un array de números enteros, sólo tenemos que utilizar el método `sort`. En el caso de que queramos ordenar el array de notas de nuestros ejemplos:

```
Arrays.sort(notas);
```

Por tanto, la sintaxis será:

```
Arrays.sort(nombre_array);
```

Cabe destacar que, al igual que hacíamos con la clase `Scanner`, debemos importar la clase `Array` a nuestro programa:

```
import java.util.Arrays;
```

Veamos un ejemplo completo:

```
import java.util.Arrays;

public class OrdenarArraySort
{
    public static void main (String[] args)
    {
        int[] notas = {9,7,8,5,3};

        System.out.print("Notas antes de ordenar: ");
        for (int i = 0; i < notas.length; i++)
            System.out.print(notas[i] + " ");

        System.out.println("");

        Arrays.sort(notas);

        System.out.print("Notas después de ordenar: ");
        for (int i = 0; i < notas.length; i++)
            System.out.print(notas[i] + " ");
    }
}
```

La salida por pantalla será:

```
Notas antes de ordenar: 9 7 8 5 3
Notas después de ordenar: 3 5 7 8 9
```

Otros métodos interesantes de la clase `Arrays` son:

Métodos	Descripción

fill	Permite rellenar un array unidimensional con un terminado valor. Sus argumentos son el array a rellenar y el valor deseado. Por ejemplo, para rellenar con todo a -1 un array donde almacenemos notas:
	
	`int[] notas = new int[10];`
	`Arrays.fill(notas, -1);`
	
	También podemos decidir desde qué índice hasta qué índice rellenamos:
	
	`Arrays.fill(notas, 5, 8, -1); // almacena -1 desde la posición 5 hasta la 7 del array notas`

equals	Compara dos arrays y devuelve true si son iguales (false en caso contrario). Se consideran iguales si son del mismo tipo, tamaño y contienen los mismos valores. `Arrays.equals(notas1, notas2); // notas1 y notas2 son arrays`

binarySearch	Permite buscar un elemento de forma super eficiente y rápida en un array ordenado (atención, eso es importante, que esté ordenado). Devuelve el índice del elemento buscado. Por ejemplo:
	
	`int notas[] = {9, 7, 8, 5, 3};`
	`Arrays.sort(notas);`
	`Arrays.binarySearch(notas, 5); //Devuelve el índice 3 que es donde está el elemento 5`

> **Nota**
>
> Toda la información de la clase Arrays la puedes encontrar en su documentación oficial:
>
> https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html

3.4. Arrays multidimensionales

En los anteriores apartados estudiamos como usar arrays unidimensionales para guardar una colección de elementos de forma lineal. Sin embargo, Java permite crear arrays bidimensionales también llamados matrices o tablas. En estas estructuras puedes almacenar y manejar elementos como si tuvieras una tabla.

> **Nota**
>
> Utilizaremos la denominación matriz para referirnos a los arrays bidimensionales.

Por ejemplo, la siguiente tabla que muestra las notas de los alumnos en cada una de las evaluaciones del curso, puede ser almacena en una matriz llamada notasCurso.

	1a. Evaluación	*2a. Evaluación*	*3a. Evaluación*
Baby yoda	9	10	10
Luke	3	4	5
Leia	9	8	10
Rey	8	9	9

En este apartado, vamos a estudiar:

- Cómo declarar e inicializar una matriz bidimensional.
- Cómo acceder a los elementos de una matriz bidimensional.

3.4.1. Declarar e inicializar una matriz bidimensional

La sintaxis es muy parecida a la declaración de un array unidimensional, aunque debemos añadir dos corchetes:

```
tipo_de_variable[][] nombre_de_variable = new tipo_de_variable[nfilas][ncolumnas];
```

Debemos indicar, además, dos tamaños para la matriz. Un tamaño para el número de filas (nfilas) y un tamaño para el número de columnas (ncolumnas) de nuestra matriz. Si piensas en la estructura de una tabla quizá te ayude a visualizarlo.

Siguiendo el ejemplo de la tabla anterior, para declarar la matriz `notasCurso`:

```
int[][] notasCurso = new int[4][3]; // 4 filas y 3 columnas
```

Y si quisiéramos rellenar los datos del alumno Baby yoda:

```
notasCurso[0][0] = 9;  // fila 0, columna 0
notasCurso[0][1] = 10; // fila 0, columna 1
notasCurso[0][2] = 10; // fila 0, columna 2
notasCurso[1][0] = 3;  // fila 1, columna 0
notasCurso[1][1] = 4;  // fila 1, columna 1
notasCurso[1][2] = 5;  // fila 1, columna 2
notasCurso[2][0] = 9;  // fila 2, columna 0
notasCurso[2][1] = 8;  // fila 2, columna 1
notasCurso[2][2] = 10; // fila 2, columna 2
notasCurso[3][0] = 8;  // fila 3, columna 0
notasCurso[3][1] = 9;  // fila 3, columna 1
notasCurso[3][2] = 9;  // fila 3, columna 2
```

Los datos de la tabla de alumnos se almacenan en este tipo de estructura, con los índices de filas y columnas para acceder a cada elemento:

		índices columnas		
		0	1	2
índices filas	0	9	10	10
	1	3	4	5
	2	9	8	10
	3	8	9	9

Sin embargo, hay una forma más sencilla de inicializar la matriz con unos valores predeterminados:

```
int[][] notasCurso = {
  {9,10,10},
  {3,4,5},
  {9,8,10},
  {8,9,9},
};
```

3.4.2. Recorrer una matriz bidimensional

Una vez tengamos la matriz rellanada con datos, podremos recorrer los elementos de la matriz para mostrarlos por pantalla.

Como tenemos un array bidimensional, a diferencia del array unidimensional, deberemos utilizar un bucle anidado: un bucle para recorrer las filas y otro bucle para recorrer las columnas.

```
public static void main (String[] args)
{
    int[][] notasCurso = {
      {9,10,10},
      {3,4,5},
      {9,8,10},
      {8,9,9},
    };

    for (int i = 0; i < notasCurso.length; i++) // Recorre filas
    {
        System.out.print("Notas del alumno " + i + ": ");

        for (int j = 0; j < notasCurso[i].length; j++) // Recorre columnas
            System.out.print(notasCurso[i][j] + " ");

        System.out.println();
    }
}
```

Como puedes observar en este código, para controlar que estamos dentro del rango del tamaño de la fila o columna, utilizamos el método `length`

Sin embargo, debes fijarte que a diferencia de como lo hacíamos con el array unidimensional, se debe indicar de que array queremos obtener el tamaño (el de las filas o el de las columnas):

```
notasCurso.length; // nos devuelve el número de filas de la matriz notasCurso
notasCurso[i].length; // nos devuelve el número de columnas que tiene la fila i
```

> **Nota**
>
> Una matriz es un array de arrays. Por tanto, podríamos visualizar que una matriz tiene N arrays (n filas) de M elementos cada uno. Por ejemplo, en el ejemplo anterior, tenemos una matriz compuesta por 4 arrays y cada array está compuesto de 3 elementos.

Finalmente, hay que tener en cuenta que las operaciones y restricciones que se aplican a un array unidimensional también valen para los arrays multidimensionales.

4. Cadenas de caracteres

Ya sabemos utilizar el tipo de dato *char* para almacenar un carácter.

```
char letra = 'a';
```

Un texto no es nada más y nada menos que una cadena de caracteres. Por tanto, ahora que ya conocemos el concepto de array, podríamos crear la cadena "apto" como un array de tipo `char`:

```
char[] calificacion = {'a', 'p', 't', 'o'};
```

Aunque como puedes comprobar esta forma de crear cadenas no es eficiente. ¿Y si tuviéramos que crear textos más largos? Es por ello que Java incluye el tipo String, que ha sido especialmente diseñado para la manejar cadenas.

En la unidad 3 ya se hizo una introducción al tipo de dato String. Sin embargo, no se profundizó en los métodos de los que este tipo de dato dispone. En los siguientes apartados se hará una descripción detallada del potencial de la clase String para crear y manipular cadenas de texto.

> **Nota**
>
> Recuerda que String es una clase. Y al igual que la clase Scanner, Math o Arrays, proporciona una serie de herramientas (métodos) que incluyen algoritmos que nos facilitan la tarea de programar.

Estas son las operaciones más habituales que se hacen con una cadena de texto:

- Creación de una cadena.
- Leer la cadena desde la entrada estándar.
- Acceder a un carácter de la cadena.
- Determinar la longitud de la cadena.
- Concatenar con otras cadenas.
- Comparar con otras cadenas.
- Extraer una subcadena.
- Buscar texto dentro de la cadena.
- Expresiones regulares

En los siguientes apartados estudiaremos estas operaciones haciendo uso de los objetos de Java más adecuados.

4.1. La clase String

Hasta ahora hemos utilizado la clase String como variable para crear datos de tipo cadena. A continuación, estudiaremos los métodos que proporciona esta clase para manipular cadenas de texto.

4.1.1 Declaración, creación e inicialización

Existen diferentes formas de construir un String. A continuación, se muestra un ejemplo en el que se construye a partir de una secuencia de carateres encerrados entre comillas dobles (“”) y a través de un array de elementos de tipo char.

```
String cadena1 = "Ada Lovelace";

System.out.println(cadena1);

char[] arrayCaracteres = {'A','d','a',' ','L','o','v','e','l','a','c','e'};

String cadena2 = new String(arrayCaracteres);

System.out.println(cadena2);
```

> **Nota**
>
> Una vez que se crea e inicializa un String este es inmutable y no se puede modificar.

4.1.2. Lectura desde teclado

En la unidad 3 ya aprendimos a obtener una cadena de texto desde la entrada estándar (teclado) con la clase Scanner mediante el método `nextLine()`:

```
Scanner sc = new Scanner(System.in);
String nombre = sc.nextLine();
```

Cuidado, no confundir `nextLine()` con `next()`. Por ejemplo, si escribimos “Ada Lovelace” en la entrada del programa y pulsamos la tecla enter:

- `nextLine()`: obtiene una cadena hasta encontrar el retorno de carro (la tecla enter). Por tanto, obtiene la cadena “Ada Lovelace”.
- `next()`: obtiene una cadena hasta encontrar el carácter espacio. En este caso, la cadena que obtiene es “Ada”.

4.1.3. Acceder a un carácter de la cadena

En la unidad 3 también aprendimos a obtener un carácter desde teclado con el método:

```
charAt(int posicion)
```

Y lo utilizábamos de esta forma:

```
Scanner sc = new Scanner(System.in);
String nombre = sc.nextLine().charAt(0);
```

Aunque lo utilizáramos junto con la clase Scanner, este método pertenece realmente a la clase String. Si descomponemos el anterior ejemplo en más instrucciones se puede entender mejor:

```
Scanner sc = new Scanner(System.in);
String nombre = sc.nextLine(); // Obtiene "Ada Lovelace"

char caracter = nombre.charAt(0); // Obtiene 'A'
```

Hasta ahora le hemos pasado un 0 (cero) al charAt() para obtener el primer carácter de la cadena. En cierta manera, esto se puede ver como un truco para obtener un carácter de entrada.

Sin embargo, podemos obtener cualquier carácter de la cadena modificando ese valor. Por ejemplo, si queremos obtener la segunda letra de "Ada Lovelace":

```
char caracter = nombre.charAt(1); // Me da el carácter 'd'
```

Ten en cuenta que la cadena es un array y por tanto sus índices funcionan de la misma manera.

4.1.4. Longitud de la cadena

Al igual que en los arrays, se puede saber cuántas letras forman una cadena con length:

```
String nombre = "Grace Hooper";
System.out.println( nombre.length() ); // 12
```

Otro ejemplo más completo donde mostramos carácter a carácter una cadena de texto de:

```
char[] letras = new char[nombre.length()]; // Creamos array de caracteres

letras = nombre.toCharArray(); // Convierte nombre a array de caracteres

for (int i = 0; i < nombre.length(); i++)
{
    System.out.print(letras[i] + " "); // Mostramos letra a letra la cadena
}
```

Como puedes observar hemos utilizado otra operación sobre String:

```
nombre.toCharArray();
```

Esto convierte la variable nombre, que es de tipo String, en un array de caracteres.

4.1.5. Concatenar cadenas

En el caso de que queramos concatenar (juntar) dos podemos utilizar el método `concat`. La instrucción que se muestra a continuación concatena la cadena `bienvenida` y la cadena `nombre`, y la cadena resultante se guarda en otra cadena llamada `saludo`:

```
String bienvenida = "Bienvenida ";
String nombre = "Hedy Lamarr";
String saludo = bienvenida.concat(nombre);
```

O también:

```
String bienvenida = "Bienvenida ";
String saludo = bienvenida.concat("Hedy Lamarr");
```

Aunque debido a que la concatenación de cadenas es una operación muy habitual en los programas, Java permite hacer de una manera más simple. Es una operación que hemos ido haciendo durante el curso como puedes ver en el siguiente ejemplo:

```
String saludo = bienvenida + nombre;
```

También se permite concatenar cadenas con números, y en este caso el número se convertirá a String automáticamente:

```
System.out.println("Nota final: " + 7.2);
```

O también:

```
String mensajeNota = "Nota final: " + 7.2;
System.out.println(mensajeNota);
```

Pero recuerda, que si haces operaciones aritméticas dentro del operador de concactenación '+', debes ponerlas entre paréntesis:

```
System.out.println("Nota final: " + (7.2 + 2);
```

4.1.6. Comparar cadenas

Ya sabemos cómo ver si una cadena tiene exactamente un cierto valor o si dos cadenas son iguales o no. Para ello empleamos la operación `equals` (y no ==). Siendo s1 y s2 variables de tipo String: `s1.equals(s2);`

Sin embargo, no sabemos comprobar qué cadena es mayor que otra (cuál aparecería la última de las dos en un diccionario), y se trata de algo que es necesario si deseamos ordenar textos. El operador mayor que (>), que usamos con los números, no se puede aplicar directamente en cadenas. En su lugar, debemos emplear el

operador compareTo, el cual devolverá un número mayor que 0 si nuestra cadena es mayor que la que indicamos como parámetro (o un número negativo si nuestra cadena es menor, o 0 (cero) si son iguales):

```
String apellidoAlu1 = "Simo";
String apellidoAlu2 = "Martinez";
if (apellidoAlu1.compareTo(apellidoAlu2) > 0)
   System.out.println("Los alumnos en orden alfabético son " + apellidoAlu2 + " " + apellidoAlu1);
else
   System.out.println("Los alumnos en orden alfabético son " + apellidoAlu1 + apellidoAlu2);
```

4.1.7. Extraer una subcadena

Como hemos visto anteriormente, podemos extraer un carácter de una cadena con charAt. Además, también podemos obtener una subcadena de la cadena utilizando substring. Por ejemplo:

```
String mensaje1 = "Bienvenido a Programación";
String mensaje2 = mensaje1.substring(0,13) + "Bases de Datos";
System.out.println(mensaje2); // Muestra "Bienvenido a Bases de Datos"
```

Se puede deducir que la subcadena se toma desde la posicion inicial (por ejemplo, 0) hasta una posición final (sin incluir esa posición final). Por ejemplo, "Bienvenido a " tiene 13 caracteres (cuenta el último espacio), y queremos obtener la subcadena desde el principio.

4.1.8. Buscar en una cadena

La inmensa mayoría los editores de texto, navegadores, etc, ofrecen la opción de buscar alguna palabra dentro del texto. En los String, para ver si una cadena contiene un cierto texto, podemos usar `indexOf` (puede leerse como posición de), que nos dice en qué posición se encuentra la cadena buscada. En caso de que sea 0 será la primera, y en caso de que no se encuentre, -1.

Un ejemplo de su uso es el siguiente:

```
String mensaje = "Ada Lovelace fue la primera mujer programadora de la historia";
int posicionPalabra = mensaje.indexOf("Lovelace");
if ( posicionPalabra >= 0)
    System.out.println("Palabra encontrada en la posición: " + posicionPalabra);
```

Salida:

```
Palabra encontrada en la posición: 4
```

Ten en cuenta que empieza contando los caracteres de la cadena desde 0.

4.1.9. Otras operaciones con cadenas

Podemos utilizar otras operaciones útiles para tratar cadenas:

- Convertir cadena a minúsculas: `cadena.toLowerCase();`
- Convertir cadena a mayúsculas: `cadena.toUpperCase();`
- Eliminar espacios en blanco: `cadena.trim();`
- Comprobar si la cadena está vacía: `cadena.isEmpty();`

> Nota
>
> Toda la información de la clase String la puedes encontrar en su documentación oficial:
>
> https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

4.2. Expresiones regulares con cadenas de texto

A menudo necesitaremos escribir código que valide la entrada del usuario, como por ejemplo comprobar si el dato es un número, una cadena con todos los caracteres en minúscula, o si es el DNI. Este tipo de problemas se pueden solucionar adecuadamente utilizando *expresiones regulares*.

Una expresión regular (abreviada como **regex** en inglés) es una carácter o conjunto de caracteres (cadena) que describe un patrón de búsqueda. De esta manera, podemos encontrar, reemplazar o separar una cadena según un determinado patrón de búsqueda.

> **Nota**
>
> Las expresiones regulares son una herramienta extremadamente útil y potente, muy utilizadas tanto por administradores de sistemas como programadores web con lenguajes de tipo script.

El método que maneja principalmente las expresiones regulares con la clase String es `matches`. Este método devuelve `true` si la cadena que se examina coincide con la expresión regular.

En principio, `matches` es muy similar a `equals`. Por ejemplo, las siguientes dos instrucciones son evaluadas como `true`.

```
String lenguaje = "Java";

lenguaje.matches("Java"); // true
lenguaje.equals("Java"); // true
```

Sin embargo, la herramienta `matches` es mucho más potente. No solo puede confirmar la coincidencia de una determinada cadena con otra, sino también un conjunto de cadenas que siguen un patrón determinado (expresión regular). Por ejemplo, a partir de estos tres mensajes:

```
String mensaje1 = "Java mola";
String mensaje2 = "Java es divertido";
String mensaje3 = "Java es potente";
```

Podemos aplicar el método `matches`, el cual evaluará las siguientes instrucciones como true:

```
mensaje1.matches("Java.*"); // true
mensaje2.matches("Java.*"); // true
mensaje3.matches("Java.*"); // true
```

`"Java.*"` es una expresión regular. Describe el patrón de una cadena que empieza por la palabra Java seguida por cero o más caracteres.

Otro ejemplo:

```
String codigo = "440-02-4534";
codigo.matches(\\d{3}-\\d{2}-\\d{4}); // true
```

En el anterior ejemplo `\\d` representa un único dígito, y `\\d{3}` representa 3 dígitos.

A continuación, se hace un resumen en forma de tablas de los meta caracteres disponibles que pueden utilizarse en expresiones regulares.

Símbolos comunes en expresiones regulares

Regex	Descripción
.	Un punto indica cualquier carácter
^regex	El símbolo ^ indica el principio del String. En este caso el String debe contener la expresión al principio.
regex$	El símbolo $ indica el final del String. En este caso el String debe contener la expresión al final.
[abc]	Los corchetes representan una definición de conjunto. En este ejemplo el String debe contener las letras a ó b ó c.
[abc][12]	El String debe contener las letras a ó b ó c seguidas de 1 ó 2
[^abc]	El símbolo ^ dentro de los corchetes indica negación. En este caso el String debe contener cualquier carácter excepto a ó b ó c.
[a-z1-9]	Rango. Indica las letras minúsculas desde la a hasta la z (ambas incluidas) y los dígitos desde el 1 hasta el 9 (ambos incluidos)
A|B	El carácter | es un OR. A ó B
AB	Concatenación. A seguida de B

Meta caracteres

Regex	Descripción
\d	Dígito. Equivale a [0-9]
\D	No dígito. Equivale a [^0-9]
\s	Espacio en blanco. Equivale a [\t\n\x0b\r\f]

\S	No espacio en blanco. Equivale a [^\s]
\w	Una letra mayúscula o minúscula, un dígito o el carácter _ Equivale a [a-zA-Z0-9_]
\W	Equivale a [^\w]
\b	Límite de una palabra.

Cuantificadores

Regex	Descripción
{X}	Indica que lo que va justo antes de las llaves se repite X veces
{X,Y}	Indica que lo que va justo antes de las llaves se repite mínimo X veces y máximo Y veces. También podemos poner {X,} indicando que se repite un mínimo de X veces sin límite máximo.
*	Indica 0 ó más veces. Equivale a {0,}
+	Indica 1 ó más veces. Equivale a {1,}
?	Indica 0 ó 1 veces. Equivale a {0,1}

Ejemplo de uso de expresiones regulares:

```
String trabalenguas = "Traba tu lengua el traba de trabalenguas";

// Ejemplo 1: devuelve false, ya que la cadena tiene más caracteres
System.out.println("Ejemplo 1: " + trabalenguas.matches("Traba"));

// Ejemplo 2: devuelve true, siempre y cuando no cambiemos la cadena Traba
System.out.println("Ejemplo 2: " + trabalenguas.matches("Traba.*"));

// Ejemplo 3: devuelve true, siempre que uno de los caracteres se cumpla
System.out.println("Ejemplo 3: " + trabalenguas.matches(".*[tlg].*"));

// Ejemplo 4: devuelve false, ya que ninguno de los caracteres están
System.out.println("Ejemplo 4: " + trabalenguas.matches(".*[xyz].*"));

// Ejemplo 5: devuelve true, si quitamos los caracteres delante de ? del String origina seguira devolviendo true
System.out.println("Ejemplo 5: " + trabalenguas.matches("Tra?ba tu le?ngua el tr?aba de tr?abale?nguas"));

// Ejemplo 6: devuelve false, ya tenemos una T mayúscula al comienzo del String
System.out.println("Ejemplo 6: " + trabalenguas.matches("[a-z].*"));

// Ejemplo 7: devuelve true, ya tenemos una T mayúscula al comienzo del String
System.out.println("Ejemplo 7: " + trabalenguas.matches("[A-Z].*"));

String cadena = "abc1234";

// Ejemplo 8: devuelve true, ya que debe repetirse alguno de los caracteres al menos una vez
System.out.println("Ejemplo 8: " + cadena.matches("[abc]+.*"));

// Ejemplo 9: devuelve true, ya que debe repetirse un valor numérico 4 veces
System.out.println("Ejemplo 9: " + cadena.matches("[abc]+\\d{4}"));

// Ejemplo 10: devuelve true, ya que debe repetirse un valor numérico entre 1 y 10 veces
System.out.println("Ejemplo 9: " + cadena.matches("[abc]+\\d{1,10}"));
```

4.2.1. Reemplazar subcadenas

Muchas veces necesitaremos reemplazar una palabra por otro dentro de un texto. Esto se puede hacer de forma sencilla con `replaceAll`:

```
String noticia = "Rafa Nadal golpeó la pelota con su raqueta mientras comía una pelota";
String noticiaFake = noticia.replace("pelota", "naranja");
System.out.println(noticiaFake);
```

Salida: `Rafa Nadal golpeó la naranja con su raqueta mientras comía una naranja`

El primer parámetro de `replaceAll` es la cadena buscada y el segundo parámetro es la cadena por la que se va a reemplazar.

> **Nota**
>
> Los String son inmutables y es por ello que necesitamos guardar en otro String el resultado de la cadena reemplazada.

Sin embargo, también es posible que queramos reemplazar una cadena según un patrón de caracteres determinado. Ahora ya sabemos que lo podemos resolver con una expresión regular.

Asimismo, `replaceAll` acepta como primer parámetro una expresión regular. En el siguiente ejemplo reemplazamos el patrón “ab”, pero solamente el que aparece al principio de la cadena1:

```
String cadena1 = "abc bca abzd ab cdabs";
String cadena2 = noticia.replaceAll("^ab", "xy");
System.out.println(cadena2);
```

Salida: `xyc bca abzd ab cdabs`

4.2.2. Separar cadena en subcadenas

Una operación relativamente frecuente, pero trabajosa, es descomponer una cadena en varios fragmentos que estén delimitados por ciertos separadores. Por ejemplo, podríamos descomponer una frase en varias palabras que estaban separadas por espacios en blanco.

Si lo queremos hacer "de forma artesanal", podemos recorrer la cadena buscando y contando los espacios (o los separadores que nos interesen). Así podremos saber el tamaño del array que deberá almacenar las palabras (por ejemplo, si hay dos espacios, tendremos tres palabras). En una segunda pasada, obtendremos las subcadenas que hay entre cada dos espacios y las guardaríamos en el array. No es especialmente sencillo.

Afortunadamente, Java nos permite hacerlo con `split`, que crea un array a partir de los fragmentos de la cadena, usando el separador que le indiquemos, así:

```
String ejemplo = "Java C# Python C++";

String[] ejemploPartido = ejemplo.split(" ");

for (int i = 0; i < ejemploPartido.length; i++)
    System.out.println("Lenguaje " + i + " = " + ejemploPartido[i]);
```

Salida:

```
Lenguaje 0 = Java
Lenguaje 1 = C#
Lenguaje 2 = Python
Lenguaje 3 = C++
```

Aunque también podemos usar expresiones regulares:

```
String ejemplo = "Java C# Python C++";

String[] ejemploPartido = ejemplo.split("\\s"); // \\s es una regex de espacio en blanco

for (int i = 0; i < ejemploPartido.length; i++)
    System.out.println("Lenguaje " + i + " = " + ejemploPartido[i]);
```

Obtenemos la misma salida que en el ejemplo anterior.

5. Bibliografía

Documentación oficial: https://docs.oracle.com/en/java/javase/17/docs/api/index.html

Librerías de clases útiles: Apuntes de José Chamorro del CFGS DAW del IES Sant Vicent Ferrer (Algemesí).