

Complemento 1

Programación básica y utilización de objetos en Java

Acceso a Datos (Desarrollo de aplicaciones multiplataforma)
Carlos Alberto Cortijo Bon



Esta obra está bajo una [Licencia Creative Commons Atribución-NoComercial-SinDerivadas 4.0 Internacional](https://creativecommons.org/licenses/by-nc-nd/4.0/)

Índice

1. Programa mínimo en Java	1
2. Parámetros de línea de comandos	1
3. Tipos de datos básicos en Java	2
4. Entrada y salida estándar y de error	3
4.1. Salida hacia la salida estándar y de error	4
4.2. Entrada desde la entrada estándar	6
5. Clases <i>wrapper</i> para tipos básicos	8
5.1. Creación de objetos de tipos <i>wrapper</i>	8
5.2. Métodos de las clases <i>wrapper</i> para lectura de datos de tipos básicos	9
6. Arrays en Java	9
6.1. Paso de <i>arrays</i> a métodos	11
6.2. Bucle for para iterar sobre los elementos de un <i>array</i>	12
7. Clases y objetos	13
7.1. Constructores y variables de instancia	14
7.2. Creación de objetos	15
7.3. El valor null	15
7.4. Métodos	15
7.5. Acceso a variables de instancia y métodos de una clase	16
7.6. Variables de clase y métodos estáticos	16
8. Tipos enumerados	17
8.1. Tipos enumerados básicos	18
8.2. Uso de tipos enumerados básicos	18
8.3. Iteración sobre todos los valores de un tipo enumerado	20
8.4. Modificación y copia de los contenidos de un <i>array</i>	20
9. Biblioteca estándar de clases de Java	22
10. La clase String para cadenas de caracteres	24
11. La clase Math para operaciones matemáticas	28
12. La clase Random para generación de números aleatorios	29

El siguiente documento contiene una breve introducción al lenguaje Java pensada para quienes ya tienen un buen conocimiento previo de un lenguaje de programación orientado a objetos similar, como por ejemplo C++ o C#.

1. Programa mínimo en Java

El siguiente programa en Java es el típico programa que simplemente escribe un texto en la salida estándar.

```
package minimo;

public class Minimo {

    public static void main(String[] args) {

        System.out.println("Hola.");

    }

}
```

Cuando se ejecuta un programa en Java, se ejecuta lo que hay en el método `main`. En este caso, envía un texto a la salida estándar (`System.out`), haciendo uso del método `println`.

Método `main` como punto de entrada a un programa

```
public static void main(String[] args)
```

En Java, el punto de entrada a un programa es el método `main` de una clase pública (`public`).

Tiene como parámetro `String[] args`, que contiene los valores de los parámetros de línea de comandos.

El número de parámetros de línea de comandos es `args.length`.

El método `main` es estático porque:

- En Java no puede existir código que no esté incluido en un método, ni métodos que no estén incluidos en una clase. Por tanto, el punto de entrada a un programa escrito en Java debe ser un método que pertenece a una clase. Y a este método se le da el nombre `main`. Tampoco puede existir en Java una clase que no pertenezca a un paquete. En este ejemplo, este método pertenece al paquete `minimo`.
- El propósito del método `main` es servir como punto de entrada a un programa. Si no fuera estático, habría que crear una instancia de la clase a la que pertenece para poder ejecutarlo. Esto sería una complicación innecesaria. De hecho, en la práctica, la clase que contiene el método `main` casi nunca contiene ningún otro método que no sea estático.

2. Parámetros de línea de comandos

Como ya se ha comentado, los parámetros de línea de comandos están disponibles en el método `main`, que es el punto de entrada de un programa en Java. Este es un `array` de `String`. Los dos programas siguientes escriben los valores de todos los parámetros de línea de comandos.

Parámetros de línea de comandos (variante 1)

Parámetros de línea de comandos (variante 2)

```
package escribeargs;

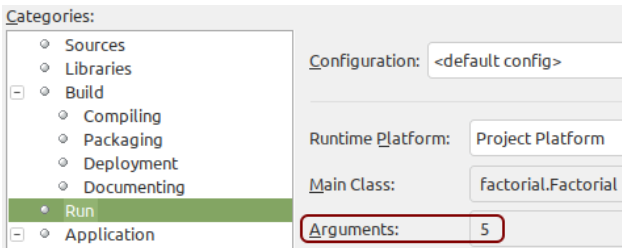
public class EscribeArgs {
```

<pre>public static void main(String[] args) {</pre>	
<pre> for (int i = 0; i < args.length; i++) { System.out.printf("%d: %s\n", i, args[i]); }</pre>	<pre> int i = 0; for (String arg: args) { System.out.printf("%d: %s\n", i, arg); i++; }</pre>
<pre>}</pre>	

Especificación de valores para argumentos de línea de comandos en entornos de desarrollo

Los entornos de desarrollo suelen tener una opción para especificar los valores de los parámetros de línea de comandos. En NetBeans, por ejemplo, se pueden indicar valores para los argumentos de línea de comandos desde las propiedades del proyecto. Para ello, se pulsa con el botón derecho del ratón sobre el proyecto, y se selecciona la opción *Properties*. Dentro de la caja de diálogo que aparece, se pueden indicar los argumentos de línea de comandos en el apartado *Run*, en la opción *Arguments*.

Se muestra un ejemplo para un programa que calcula el factorial de un número introducido como parámetro de línea de comandos. Este programa de ejemplo solo utiliza un parámetro de línea de comandos. Y en este ejemplo de ejecución se especifica el valor 5 para él. Si hay varios valores, se introducen separados por espacios. Es posible especificar un valor para un parámetro de línea de comandos que incluya un espacio. Para ello, debe especificarse el valor entre comillas dobles.



Categories:

- Sources
- Libraries
- Build
 - Compiling
 - Packaging
 - Deployment
 - Documenting
- Run**
- Application

Configuration: <default config>

Runtime Platform: Project Platform

Main Class: factorial.Factorial

Arguments: 5

```
public static void main(String[] args) {
    int n;
    if (args.length > 0) {
        n = Integer.parseInt(args[0]);
    } else {
        Scanner scan = new Scanner(System.in);
        System.out.printf("Introduzca n: ");
        n = scan.nextInt();
    }
    int factorial = 1;
    while (n > 1) {
        factorial = factorial * n;
        n--;
    }
    System.out.printf("factorial: %d\n", factorial);
}
```

factorial.Factorial > main >

out - Factorial (run) x

```
run:
factorial: 120
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Tipos de datos básicos en Java

En la siguiente tabla se muestran las características de los tipos de datos básicos existentes en Java.

Acceso a Datos. Complemento 1: Programación básica y utilización de objetos en Java

Tipo	Descripción	Longitud (bits)	Valor mínimo	Valor máximo	Valor por defecto
int	Entero	32	-2^{31} (Integer.MIN_VALUE)	$2^{31}-1$ (Integer.MAX_VALUE)	0
long	Entero largo	64	-2^{63} (Long.MIN_VALUE)	$2^{63}-1$ (Long.MAX_VALUE)	0
short	Entero corto	16	-2^{15} (Short.MIN_VALUE)	$2^{15}-1$ (Short.MAX_VALUE)	0
float	Número real con precisión simple	32	$1.40239846 \times 10^{-45}$ (Float.MIN_VALUE)	$3.40282347 \times 10^{38}$ (Float.MAX_VALUE)	0
double	Número real con doble precisión	64	$4.9406564584124654 \times 10^{-324}$ (Double.MIN_VALUE)	$1.7976931348623157 \times 10^{308}$ (Double.MAX_VALUE)	0
byte	Byte	8	-2^8 (Byte.MIN_VALUE)	2^8-1 (Byte.MAX_VALUE)	0
char	Carácter Unicode	16	'\u0000'	'\uFFFF'	'\u0000'
boolean	Valor de verdad true (cierto) o false (falso). Internamente se representa, en general, como int. Pero puede representarse de manera más compacta en arrays.				False

Nota: Para los tipos double y float, los valores mínimo y máximo indicado son para el valor absoluto del número. Estos tipos permiten representar un intervalo de números simétrico con respecto al 0. Pero los valores muy cercanos a cero no se pueden representar. Esto se conoce como *underflow*.

En ocasiones es necesario o conveniente utilizar determinados sufijos para los literales de algunos de estos tipos.

- **l** o **L** para long. Es preferible **L**, porque la letra **l** puede confundirse fácilmente con el número 1.
- **f** o **F** para float.
- **d** o **D** para double. En principio no es necesario, porque un literal que representa un número con decimales se interpreta automáticamente como de tipo double.

Se muestran a continuación varios ejemplos.

```
int a = 2000000000;
long b = 2000000000000000000L;
short c = 32767;
byte d = 45;
byte e = 'A';
char f = 'Ñ';
float g = 54f;
double h = 54d;
String saludo = "Hola";
```

Los números se pueden expresar en bases distinta de la decimal, que se indica mediante un prefijo para el valor. Estos son los siguientes:

- **0x** para hexadecimal (base 16). Por ejemplo: **0x32F6**.
- **0b** para binario (base 2). Por ejemplo: **0b1110101**.

4. Entrada y salida estándar y de error

En Java se pueden realizar operaciones de entrada desde la entrada estándar y de salida hacia la salida estándar y de error.

	Asociada por defecto a	Representación en Java
Entrada estándar	Teclado	System.in
Salida estándar	Pantalla (terminal de texto)	System.out
Salida de error	Pantalla (terminal de texto)	System.err

4.1. Salida hacia la salida estándar y de error

Para mostrar información en la salida estándar (**System.out**) se pueden utilizar varios métodos:

print	Muestra la información
println	Igual que la anterior, pero al final se escribe un separador de línea, de manera que lo que se escriba a continuación se hará al principio de la siguiente línea.

El programa mínimo en Java visto anteriormente utilizaba el método **print** para enviar una cadena de caracteres a la salida estándar. El siguiente programa hace esto mismo, pero utilizando el método **println** para añadir un salto de línea al final, de manera que lo que se escriba a continuación no aparezca en la misma línea, sino en la siguiente. A continuación, envía un texto con información diversa a la salida estándar. Para ello, utiliza el operador de concatenación de cadenas **+**. Esto es un tanto engorroso, pero enseguida se verá una manera más conveniente de hacerlo.

```
package mostrarinformacion;

public class MostrarInformacion {

    public static void main(String[] args) {
        System.out.println("Hola.");
        String saludo = "Hola";
        String nombre = "Carlos";
        int dias = 9;
        System.out.println(saludo + " " + nombre + ", hace " + dias
            + " que no te veo.");
    }
}
```

Para evitar componer el texto utilizando la concatenación de cadenas, se puede utilizar el método **printf**. Con este, se especifica un patrón para el texto, en el que se incluyen unos marcadores o *placeholders* para indicar dónde irá determinada información de determinados tipos, cuyo valor se proporciona, por separado, a continuación. Es preferible utilizar el método **printf** y evitar la concatenación de cadenas.

printf	<p>Permite especificar un formato o patrón para la escritura de la información. Este es una cadena de caracteres en la cual se insertan determinados marcadores o <i>placeholders</i>. Estos especifican un lugar en el que se escribirá determinada información, cuyo valor se proporciona a continuación, en un parámetro adicional. Debe haber un parámetro a continuación para cada marcador, y debe ser del tipo especificado en el marcador. Los principales marcadores, cada uno para un tipo de datos distinto, son los siguientes:</p> <ul style="list-style-type: none"> %s Cadena de caracteres %d Número entero. Puede usarse con los tipos <code>int</code> y <code>long</code>. %f Número real, con decimales. Puede usarse con los tipos <code>double</code> y <code>float</code>. %c Carácter. Puede utilizarse con el tipo <code>char</code>. %b Valor booleano, que puede tomar valores <code>true</code> o <code>false</code>. <p>Aparte de estos, se pueden utilizar otros que no corresponden a ningún dato que se indique a continuación, sino que representan caracteres especiales que permiten dar determinado formato a la salida.</p> <ul style="list-style-type: none"> %n Salto de línea. Lo que se escriba a continuación aparecerá al principio de la siguiente línea. Es equivalente a <code>'\n'</code>. %% % <p>Como ejemplo de uso, valga el siguiente:</p> <pre>int i = 102; String t = "Hola"; System.out.printf("Número: %d, texto: %s %s\n", i, t, ".");</pre> <p>Número: 102, texto: Hola .</p>
---------------	--

Acceso a Datos. Complemento 1: Programación básica y utilización de objetos en Java

El siguiente programa hace lo mismo que el anterior, pero utilizando `printf` en lugar de `println`.

```
package mostrarinformacion;

public class MostrarInformacion {

    public static void main(String[] args) {

        System.out.println("Hola.");

        String saludo = "Hola";
        String nombre = "Carlos";
        int dias = 9;
        System.out.printf("%s %s, hace %d días que no te veo.\n",
            saludo, nombre, dias
        );

    }

}
```

Todo lo anterior sirve también para escribir en la salida de error (`System.err`).

A continuación se muestran algunas opciones que se pueden utilizar en los marcadores o *placeholders* para formatear cada dato que se escribe.

%10s	Escribir una cadena de caracteres con 10 caracteres en total, rellenando con espacios, y justificado a la derecha.
%-10s	Escribir una cadena de caracteres con 10 caracteres en total, rellenando con espacios, y justificado a la izquierda.
%6d	Añadir espacios a la izquierda hasta completar una anchura de 6 caracteres. Útil para escribir los números alineados a la derecha.
%06d	Añadir ceros a la izquierda hasta completar una anchura de 6 caracteres.
%8f	Escribir un número real (con decimales, float o double) con una anchura total de 8 caracteres, incluyendo dígitos enteros, separador de decimales, y dígitos decimales. Se escribe justificado a la derecha.
%.2f	Escribir con 2 dígitos decimales
%6.2f	Escribir con 6 caracteres en total, incluyendo uno para el punto decimal, con dos dígitos decimales.
%-6.2f	Escribir justificado a la izquierda, con 6 caracteres en total, incluyendo uno para el punto decimal, y con dos dígitos decimales.

Actividad Complemento 1.1

Crea un programa que escriba ordenados, y justificados a la derecha, los números de 0 a 110.

Actividad Complemento 1.2

Crea un programa que, dadas las horas en una variable `int horas` y los minutos en una variable `int min`, escriba la hora en formato `hh:ss`, donde `hh` son las horas y `ss` los minutos, ambos con dos dígitos y completando, si es necesario, con ceros por la izquierda.

Actividad Complemento 1.3

Crea un programa que asigne un valor a una variable `String nombre` y después escriba su valor seguido de su longitud, que se puede obtener con `nombre.length()`. El nombre se debe escribir justificado a la izquierda y ocupando un espacio de 40 caracteres, completando con espacios por la derecha. La longitud se debe escribir justificada a la derecha y ocupando un espacio de 4 caracteres, completando con espacios por la izquierda.

Para asegurarte de que se hace bien, haz que tras escribir esto se asigne un nuevo valor a `nombre` y se haga lo mismo

con su nuevo valor. Asegúrate de que funciona bien cuando la longitud de **nombre** es distinta cada vez. Puedes escribir una primera línea que tenga varias veces 0123456789, para ayudarte a comprobar las longitudes.

4.2. Entrada desde la entrada estándar

Para leer de manera sencilla datos de diferentes tipos desde la entrada estándar (`System.in`), se puede construir un **Scanner** sobre ella. Esta clase tiene métodos que permiten obtener datos de los tipos básicos antes explicados.

Tipo	Método
String	<code>nextLine()</code>
int	<code>nextInt()</code>
long	<code>nextLong()</code>
float	<code>nextFloat()</code>
double	<code>nextDouble()</code>
boolean	<code>nextBoolean()</code>

Todos estos métodos pueden lanzar una excepción de la clase **InputMismatchException** si los datos introducidos no corresponden al tipo esperado.

El siguiente programa lee un número entero desde la entrada estándar. Pero si no se ha introducido un número, se produce una excepción de tipo **InputMismatchException**. A continuación se verá cómo se puede capturar y gestionar esta excepción, para que el programa no termine abruptamente.

```
package leeentero;

import java.util.Scanner;

public class LeeEntero {

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);

        System.out.printf("Introduce número entero: ");
        int num = s.nextInt();
        System.out.printf("Número: %d\n", num);

    }

}
```

Este programa construye un `Scanner s` para la entrada estándar `System.in` con `new Scanner(System.in)`. Después intenta leer un número de este `Scanner` con `nextInt()`.

El siguiente programa muestra cómo obtener un dato de cada tipo desde la entrada estándar, mostrando un mensaje de error cuando el dato introducido no corresponde al tipo esperado.

Cuando el texto introducido no tiene un formato apropiado para poder interpretarlo como un dato del tipo que se espera, se lanza una excepción de tipo `InputMismatchException`. Esta se captura y se muestra un mensaje de error. Hay que leer de nuevo el valor con `s.next()`, porque cuando sucede el error, este se devuelve al *stream* y, si no se hiciera así, se volvería a obtener en la siguiente operación de lec-

tura. El valor obtenido, que no tiene el formato que se esperaba, se puede utilizar para mostrar un mensaje de error. Si sucede cualquier error, se termina la ejecución del programa con `return`.

```
package leetiposbasicos;

import java.util.InputMismatchException;
import java.util.Scanner;

public class LeeTiposBasicos {

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);

        System.out.printf("Introduce texto: ");
        String linea = s.nextLine();
        System.out.printf("Texto: %s\n", linea);

        System.out.printf("Introduce número entero: ");
        int num;
        try {
            num = s.nextInt();
            System.out.printf("Número: %d\n", num);
        } catch (InputMismatchException e) {
            System.out.printf("ERROR: El valor introducido (%s) no es un entero.\n",
                s.next());
            return;
        }

        System.out.printf("Introduce número entero largo: ");
        long numGrande;
        try {
            numGrande = s.nextLong();
            System.out.printf("Número: %d\n", numGrande);
        } catch (InputMismatchException e) {
            System.out.printf("ERROR: El valor introducido no es un entero largo.\n",
                s.next());
            return;
        }

        System.out.printf("Introduce número real: ");
        double numRealD;
        try {
            numRealD = s.nextDouble();
            System.out.printf("Número: %f\n", numRealD);
        } catch (InputMismatchException e) {
            System.out.printf("ERROR: El valor introducido no es un número real.\n",
                s.next());
            return;
        }
    }
}
```

Atención: separadores de decimales y de miles para la clase `Scanner`

En los lenguajes de programación se utiliza un punto como separador de decimales. Este es el separador de decimales que se utiliza en países de lengua inglesa.

Cuando se lee información de la entrada estándar y se intenta convertir a un tipo `double` o `float`, se asume un separador de decimales que depende de la configuración regional del sistema operativo. En el caso de España, por ejemplo, el separador de decimales es una coma. Por lo tanto, debe introducirse 5,54 y no 5.54 para un valor de 5 unidades y 54 centésimas.

Además hay que tener en cuenta que, según la configuración regional para España, un punto se interpretará como separador de miles. Si está en un lugar apropiado para ello, sencillamente se ignorará. Si no, se lanzará una excepción de tipo `InputMismatchException`. Por ejemplo, 5.542 se interpreta como 5542, y 5.54 provocará una excepción de tipo `InputMismatchException`.

5. Clases *wrapper* para tipos básicos

En Java existen clases correspondientes para cada uno de los tipos básicos, según se muestra en la siguiente tabla. Se conocen como clases *wrapper* o envoltorio.

	Tipo básico	Clase	Longitud en bits
Entero	<code>int</code>	<code>Integer</code>	32
Entero largo	<code>long</code>	<code>Long</code>	64
Entero corto	<code>short</code>	<code>Short</code>	16
Número real	<code>float</code>	<code>Float</code>	32
Número real con doble precisión	<code>double</code>	<code>Double</code>	64
Byte	<code>byte</code>	<code>Byte</code>	8
Carácter Unicode	<code>char</code>	<code>Character</code>	16
Booleano	<code>boolean</code>	<code>Boolean</code>	1

5.1. Creación de objetos de tipos *wrapper*

Estas clases *wrapper* tienen un tratamiento especial. Se puede usar el operador de asignación para asignar valores de un tipo básico a un objeto de la clase *wrapper* correspondiente (*autoboxing*), y viceversa (*unboxing*).

Autoboxing	Unboxing
<code>Integer i1 = 4;</code>	<code>int i2 = i1;</code>
<code>Long l1 = 4L;</code>	<code>long l2 = l1;</code>
<code>Short s1 = 4;</code>	<code>short s2 = s1;</code>
<code>Float f1 = 4.56f;</code>	<code>float f2 = f1;</code>
<code>Double d1 = 4.56;</code>	<code>double d2 = d1;</code>
<code>Byte b1 = (byte) 234;</code>	<code>byte b2 = b1;</code>
<code>Character c1 = 'ñ';</code>	<code>char c2 = c1;</code>
<code>Boolean b11 = true;</code>	<code>boolean b12 = b11;</code>

Nota: se puede también utilizar la sintaxis habitual en Java para crear objetos de una clase *wrapper*. Por ejemplo:

```
Integer i3 = new Integer(4).
```

Pero esto está *deprecated* (considerado obsoleto, y que se puede eliminar en versiones futuras).

5.2. Métodos de las clases *wrapper* para lectura de datos de tipos básicos

Todas las clases *wrapper* tienen métodos con nombre `parseTTT`, donde *TTT* es el nombre de la clase, que permiten obtener un valor del tipo básico asociado a partir de un `String` que contiene su representación como cadena de caracteres. Si esto no es posible, porque la cadena de caracteres no tiene un formato apropiado, producen una excepción de la clase `NumberFormatException`.

Estos métodos son muy útiles para leer datos desde los argumentos de línea de comandos, porque estos se obtienen como valores de tipo `String` desde el parámetro `String[] args` del método `main`.

El siguiente programa de ejemplo intenta convertir el primer argumento de línea de comandos a un `int`, y el segundo a un `double`. Si el formato de cualquiera de ellos no es correcto, se genera una excepción de tipo `NumberFormatException`. Pero se captura, se muestra un mensaje de error, y se termina la ejecución del programa con `return`.

```
package parsingtiposbasicos;

public class ParsingTiposBasicos {

    public static void main(String[] args) {

        if (args.length < 2) {
            System.out.println("ERROR: indicar un número entero y uno con decimales.");
            return;
        }

        try {
            int n = Integer.parseInt(args[0]);
            System.out.printf("1-Número entero: %d\n", n);
        } catch (NumberFormatException e) {
            System.out.printf("Valor incorrecto: %s, no es un entero.\n", args[0]);
            return;
        }

        try {
            double d = Double.parseDouble(args[1]);
            System.out.printf("2-Número real: %f\n", d);
        } catch (NumberFormatException e) {
            System.out.printf("Valor incorrecto: %s, no es un número real.\n", args[1]);
            return;
        }

    }

}
```

6. Arrays en Java

Para crear un *array* en memoria hay que utilizar el operador `new`, especificando su longitud.

```
int[] temp;  
temp = new int[7];
```

Se pueden combinar las dos sentencias anteriores en una.

```
int[] temp = new int[7];
```

¡Atención! Declaración vs creación de arrays

Una cosa es la declaración de un *array* y otra la reserva de espacio en memoria para almacenarlo. Se podría declarar el *array* simplemente con `int[] temp`. Pero esto no crearía ningún *array* en la memoria, y no se permitiría su uso.

```
public static void main(String[] args) {  
    int[] temp;  
    System.out.println(temp.length);  
}
```

variable temp might not have been initialized

(Alt-Enter shows hints)

Cuando se crea un *array*, se asigna un valor inicial a todos sus elementos, que depende del tipo para el que se define el *array*. Para el *array* anterior, por ser un *array* de tipo `int`, se asignaría el valor inicial 0.

Junto con los datos en sí, se guarda información acerca del *array*. En particular, su longitud. Para cualquier *array* `a`, se puede obtener su longitud con `a.length`.

Se pueden escribir los contenidos del *array* anterior con el siguiente código de programa. Se puede comprobar que solo hay ceros.

```
for (int i = 0; i < temp.length; i++) {  
    System.out.print(temp[i]);  
}
```

No se permite el acceso a elementos fuera de los límites del *array*. Si un programa lo hace, se producirá una excepción de tipo **ArrayIndexOutOfBoundsException**.

Actividad Complemento 1.4

Crea un programa en Java que cree un *array* de 10 enteros y les asigne las 10 primeras potencias de 2, empezando por 1. Es decir: 1, 2, 4, 8, ... Después debe mostrar sus contenidos, para verificar que son correctos.

También se pueden asignar valores a todos los elementos del *array* en el momento de su creación, como se muestra en el siguiente ejemplo. Este declara un *array* de 7 enteros (`int`) y les asigna valores determinados. Después utiliza un bucle para escribir sus valores. La longitud del *array* `temp` la obtiene con `temp.length`.

```
package muestraarray;  
public class MuestraArray {  
    public static void main(String[] args) {  
        int[] temp = {20, 12, 18, 25, 19, 22, 24};  
        for(int i=0; i < temp.length; i++) {  
            System.out.printf("t[%d]=%d\n", i, temp[i]);  
        }  
    }  
}
```

6.1. Paso de *arrays* a métodos

El siguiente programa incluye un método que devuelve la suma de los elementos de un *array*.

Java

```
package operarray;

public class OperArray {
    public static long sumaArray(int[] nums) {
        long suma = 0;
        for (int i = 0; i < nums.length; i++) {
            suma += nums[i];
        }
        return suma;
    }
    public static void main(String[] args) {

        int[] arr = {1, 4, -2, 0, 6, -5, 3};

        System.out.printf("Suma: %d.\n",
            sumaArray(arr));

    }
}
```

Se pueden pasar a un método *arrays* bidimensionales y también de más dimensiones. En el siguiente ejemplo, se pasa un *array* bidimensional a un método que representa un tablero de juego para el juego de los barquitos a un método que dibuja el tablero de juego. Cada fila del *array* bidimensional es a su vez un *array* unidimensional.

```
package barquitos;

public class Barquitos {

    public static void dibujaTablero(char[][] tab) {
        char fila = 'A';
        for (int iFil = 0; iFil < tab.length; iFil++) {
            System.out.printf("%c ", fila++);
            char col = '1';
            for (int iCol = 0; iCol < tab[iFil].length; iCol++) {
                System.out.print(tab[iFil][iCol]);
                col++;
            }
            System.out.println();
        }
        char col = '1';
        System.out.print(" ");
        for (int iCol = 0; iCol < tab[0].length; iCol++) {
            System.out.printf("%c", col++);
        }
        System.out.println();
    }

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

```

char[][] tab = {
    {'#', ' ', '#', '#', ' ', ' '},
    {' ', ' ', ' ', ' ', ' ', ' '},
    {'#', ' ', '#', ' ', '#', ' '},
    {'#', ' ', '#', ' ', ' ', ' '},
    {'#', ' ', ' ', ' ', ' ', '#'}
};

dibujaTablero(tab);
}
}

```

6.2. Bucle `for` para iterar sobre los elementos de un *array*

Existe una variante del bucle `for` que hace muy sencilla la iteración sobre los elementos de un *array*. A continuación se muestra una variación de un ejemplo anterior que utiliza esta técnica para iterar sobre los elementos de un *array* unidimensional. Eso sí, con esta forma sencilla de iteración no está disponible directamente el índice de cada elemento.

Sin iterador	Con iterador
<pre> package muestraarray; public class MuestraArray { public static void main(String[] args) { int[] temp = {20, 12, 18, 25, 19, 22, 24}; for(int i = 0; i < temp.length; i++) { System.out.printf("t[%d]=%d\n", i, temp[i]); } } } </pre>	
	<pre> int i = 0; for (int unaTem: temp) { System.out.printf("t[%d]=%d\n", i, unaTem); i++; } </pre>

También se pueden utilizar iteradores para iterar sobre *arrays* bidimensionales, como se muestra en el siguiente ejemplo, que es también una variación de un ejemplo anterior. De manera similar se puede iterar sobre *arrays* de más dimensiones.

Sin iterador	Con iterador
<pre> public static void dibujaTablero(char[][] tab) { char fila = 'A'; for (int iFil = 0; iFil < tab.length; iFil++) { System.out.printf("%c ", fila++); char col = '1'; for (int iCol = 0; iCol < tab[iFil].length; iCol++) { System.out.print(tab[iFil][iCol]); col++; } System.out.println(); } char col = '1'; } </pre>	
	<pre> for (char[] unaFila : tab) { for (char unaPos : unaFila) { System.out.print(unaPos) } } </pre>

```
System.out.print(" ");
for (int iCol = 0; iCol < tab[0].length; iCol++) {
    System.out.printf("%c", col++);
}
System.out.println();
}
```

7. Clases y objetos

En Java cabe distinguir entre tipos básicos y clases. Los tipos básicos tienen valores elementales que se almacenan en memoria en uno o varios bytes. En Java son `int`, `long`, `short`, `float`, `double`, `byte`, `char`, `boolean`.

Los objetos pertenecen a una clase. O dicho de otra forma, un objeto es una instancia de una clase a la que pertenece. Una clase incluye tanto datos como métodos, que son operaciones que se ejecutan sobre una instancia de la clase.

A continuación se muestra la definición de una clase `Personaje`.

```
class Personaje {

    // miembros de datos
    private final String nomPers;
    private int x;
    private int y;

    // constructor
    public Personaje(String nombre, int x, int y) {
        this.nomPers = nombre;
        this.x = x;
        this.y = y;
    }

    // métodos
    public String getNomPers() {
        return nomPers;
    }
    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public void avanzaArriba(int numPasos) {
        y -= numPasos;
    }

    public void avanzaDerecha(int numPasos) {
        x += numPasos;
    }
}
```

```

    }

    public void avanzaAbajo(int numPasos) {
        y += numPasos;
    }

    public void avanzaIzquierda(int numPasos) {
        y -= numPasos;
    }
}

```

Los nombres tanto de clases como de variables siguen la convención de los nombres en forma de joroba de camello. Las jorobas serían las iniciales de palabras, que están siempre en mayúsculas. La primera letra es un caso particular. Está en mayúsculas en los nombres de clases, y en minúsculas en los nombres de variables. Algunos ejemplos:

Nombres de clases	Nombres de variables
Personaje	Personaje pers
MedioTransporte	Personaje personaje
	Personaje unPersonaje
	MedioTransporte avion1
	MedioTransporte unAvion
	MedioTransporte elMelillero
	MedioTransporte elBarcoDelArroz

7.1. Constructores y variables de instancia

Las variables de instancia de una clase tienen un valor distinto para cada objeto o instancia de la clase. En la clase Personaje del ejemplo anterior, son nomPers, x e y.

Los constructores son un tipo especial de métodos. Se invocan en el momento en que se crea un objeto o instancia de una clase. Tienen el mismo nombre de la clase, y pueden tener parámetros. En este caso particular, pero bastante típico por lo demás, lo único que hace el constructor es asignar valores a los miembros de datos de la clase a partir de valores pasados como parámetros al constructor. Para diferenciar entre el parámetro x del constructor y el miembro de datos x de la clase, este último se representa como **this.x**. Esto significa la variable de instancia x del propio objeto sobre el que se ejecuta el método, **this**.

```

class Personaje {

    // miembros de datos
    private final String nomPers;
    private int x;
    private int y;

    // constructor
    public Personaje(String nombre, int x, int y) {
        this.nomPers = nombre;
        this.x = x;
        this.y = y;
    }

    (...)
}

```

Cuando una variable se define como **final**, su valor no se puede modificar una vez que se asigna por primera vez. Es el caso de la variable de instancia nomPers. Este es el nombre del personaje, y no se puede

cambiar, una vez que se le asigna un valor en el constructor. El valor de las variables `x` e `y`, en cambio, sí se puede modificar, y se hace en los métodos `avanzaArriba`, `avanzaDerecha`, `avanzaAbajo` y `avanzaIzquierda`.

7.2. Creación de objetos

Para construir un objeto de la clase, o una instancia de la clase, se utiliza **new**. Con ello se crea un objeto de la clase y se ejecuta un constructor sobre él. Con el siguiente ejemplo, se crean varios objetos de la clase `Personaje`.

```
Personaje gato = new Personaje("Jynx", 5, 3);
Personaje raton1 = new Personaje("Pixy", 10, 2);
Personaje raton2 = new Personaje("Dixy", 20, 14);
```

7.3. El valor `null`

El valor **null** es un valor especial que se puede asignar a cualquier variable de una clase cualquiera, que representa un objeto no existente o no conocido.

También se puede asignar este valor a una variable de tipo *array*. Esto es porque, de hecho, las variables son objetos en Java.

Cuando se intenta acceder a una variable de instancia o a un método de un objeto con valor **null**, se produce una excepción de la clase `NullPointerException`. Por ello, es conveniente verificar previamente su valor, como en el siguiente ejemplo.

```
int[] nums;
(...)
if(nums == null) {
    System.out.println("Array null");
} else {
    System.out.printf("Longitud del array: %d\n", nums.length);
}
```

7.4. Métodos

Los métodos de una clase se ejecutan sobre un objeto de la clase. Ya se ha hablado de un tipo particular de métodos, los constructores, que se ejecutan cuando se crea una instancia de una clase.

Un método puede devolver un valor de un tipo determinado que se especifica antes del nombre del método. Para ello se usa una sentencia **return** con el valor que devuelve el método. Un método para el que se especifica un tipo **void** no devuelve ningún valor, y en él no se puede utilizar la sentencia **return**.

<pre>public int getX() { return x; }</pre>	<pre>public void avanzaDerecha(int numPasos) { x += numPasos; }</pre>
--	---

La clase anterior incluye varios métodos cuyo nombre empieza por `get`, y que devuelven cada uno el valor de una variable de instancia o miembro de datos, pero no modifican el valor de ninguna variable de instancia del objeto. Este es un tipo frecuente de métodos, conocidos como *getters*. En el contexto particular de estos métodos, el nombre de la variable de instancia no se puede confundir con ningún parámetro ni ninguna variable local del método. Por ello no se

antepone `this` a su nombre, pero podría hacerse. El *getter* para la variable de instancia `x` incluiría, entonces, la sentencia `return this.x`.

Los métodos *getter* anteriores permiten obtener información acerca del objeto, y en particular acerca de su estado. El estado de un objeto viene dado por los valores de sus variables de instancia. En este caso son `nombre`, `x` e `y`.

Los métodos `avanzaArriba`, `avanzaDerecha`, `avanzaAbajo` y `avanzaIzquierda` se declaran con tipo `void` y, por tanto, no devuelven ningún valor. Por otra parte, sí modifican los valores de variables de instancia.

7.5. Acceso a variables de instancia y métodos de una clase

Para referirse a algo de una clase, sean una variable de instancia o un método, se añade un punto y el nombre de la variable de instancia o del método. Se muestra a continuación un ejemplo en el que se crea un objeto de la clase `Personaje` y se ejecutan varios de sus métodos. Una vez creado el objeto, se obtienen y muestran las coordenadas de la posición inicial, se realiza un movimiento y se obtienen y muestran las coordenadas de la nueva posición.

```
Personaje gato = new Personaje("Jynx", 5, 3);
int posX = gato.getX();
int posY = gato.getY();

System.out.printf("Posición inicial: (%d, %d)\n", posX, posY);

gato.avanzaDerecha(1);

System.out.printf("Posición final: (%d, %d)\n", gato.getX(), gato.getY());
System.out.printf("Posición final: (%d, %d)\n", gato.x, gato.y);
```

Lo único que no funciona del código anterior es la última sentencia, en la que se accede directamente a las variables de instancia `x` e `y`. No se puede acceder a ellas desde fuera de la clase porque se han declarado como `private`. Si no se hubiera hecho así, se podría no solo obtener el valor de esas variables de instancia, lo que podría no ser tan grave, sino también modificarlos, lo que sí sería indeseable. Porque se quiere que solo se pueda cambiar la posición de la manera en que lo hacen los métodos `avanzaArriba`, `avanzaDerecha`, `avanzaAbajo` y `avanzaIzquierda`.

En cambio, los métodos que permiten consultar el estado de un objeto de la clase y modificarlo se declaran como `public`. Con esto se permite el acceso a ellos desde fuera del propio objeto.

Actividad Complemento 1.5

Añade a la clase anterior un método `avanza(PuntoCardinal pCard, int numPasos)` que mueva el personaje un número de pasos determinados hacia el punto cardinal dado. Utiliza el tipo enumerado `PuntoCardinal` desarrollado para *Error: no se encontró el origen de la referencia*. En el método `main`, crea un *driver* o programa de prueba que cree un objeto en una posición inicial, muestre la posición inicial, mueva el objeto varias veces utilizando los métodos anteriores (utiliza cada uno al menos una vez), y por último muestre la nueva posición. La trayectoria debe venir dada por un array de elementos del tipo enumerado `PuntoCardinal`, que representa una secuencia de movimientos.

7.6. Variables de clase y métodos estáticos

Cuando en una clase se define una variable o un método como `static`, no es una variable o un método de instancia, sino de clase. Una variable estática no tiene un valor diferente para cada instancia de la clase, sino un valor único para ella. En otras palabras, su valor es un atributo de la clase, y no de ninguna instancia particular suya. Un método estático o de clase solo puede acceder a variables o métodos estáticos o de clase.

La clase `Integer` de la biblioteca estándar de clases de Java tiene algunas variables de clase o `static`, como se puede ver en su documentación en <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Integer.html>.

Field Summary

Fields

Modifier and Type	Field	Description
static final int	BYTES	The number of bytes used to represent an int value in two's complement binary form.
static final int	MAX_VALUE	A constant holding the maximum value an int can have, $2^{31}-1$.
static final int	MIN_VALUE	A constant holding the minimum value an int can have, -2^{31} .
static final int	SIZE	The number of bits used to represent an int value in two's complement binary form.
static final Class<Integer>	TYPE	The Class instance representing the primitive type int.

Algunas de estas variables estáticas o de clase son:

static final int MIN_VALUE	Mínimo valor que puede tomar el valor entero representado por una instancia de la clase.
static final int MAX_VALUE	Máximo valor que puede tomar el valor entero representado por una instancia de la clase.
static final int BYTES	Número de bytes utilizados para representar un entero.

Estos valores se definen en variables estáticas porque son atributos de la clase y no de ninguna instancia suya particular. Además se definen como **final**, es decir, que su valor no puede cambiar, de manera que son constantes de clase. Sus nombres, siguiendo las reglas de nomenclatura habituales en Java, tienen todas las letras en mayúsculas y con las palabras separadas por guiones bajos.

Para hacer referencia a ellas, dado que son variables de clase y no de instancia, se utiliza el nombre de la clase seguido de un punto, como se muestra en el siguiente ejemplo.

```
System.out.println("Los números enteros que se pueden representar en Java están entre %d y %d.\n", Integer.MIN_INT, Integer.MAX_INT);
```

También se pueden definir métodos como **static**. Estos no se ejecutan sobre ningún objeto de la clase. En ellos, por tanto, no se puede utilizar el identificador **this**. Y como ya se ha dicho, solo pueden utilizar variables y métodos estáticos.

La clase `Integer` tiene también algunos métodos estáticos. Por ejemplo, los que permiten obtener un valor entero a partir de su representación en forma de cadena de caracteres. Son estáticos porque el valor que devuelven solo depende de los valores que se les pasan en sus parámetros, y no del estado de ninguna instancia particular de la clase.

static int <code>parseInt</code> (String s)	Devuelve el número entero que representa un String.
static int <code>parseUnsignedInt</code> (String s)	Devuelve el número entero sin signo que representa un String.
static int <code>parseInt</code> (String s, int radix)	Devuelve el número entero que representa un String. El método anterior del mismo nombre asume que el número está en base 10. Con este método se puede indicar la base de numeración.

Se hace referencia a un método estático con el nombre de la clase seguido de un punto y del nombre del método, como se muestra en el siguiente ejemplo.

```
int n = Integer.parseInt("81");
```

8. Tipos enumerados

Un tipo enumerado es un tipo de datos especial que admite un conjunto cerrado de posibles valores predefinidos y constantes. Por ejemplo:

- Las direcciones IZQUIERDA, DERECHA, ARRIBA y ABAJO.
- Los días de la semana LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO y DOMINGO.
- Los reinos de los seres vivos eucariotas: ANIMALES, VEGETALES, HONGOS, PROTOZOOS, ALGAS.

8.1. Tipos enumerados básicos

Un tipo enumerado se define con la palabra reservada `enum`. Un tipo enumerado para las direcciones se podría definir de la siguiente forma.

```
public enum Direccion {  
    IZQUIERDA, DERECHA, ARRIBA, ABAJO  
}
```

Se puede utilizar la opción `public`, como en este ejemplo, para que se pueda utilizar el tipo fuera de la clase en que se define.

8.2. Uso de tipos enumerados básicos

Se puede declarar una variable de este tipo y asignarle un valor de la siguiente forma:

```
Direccion dir = Direccion.IZQUIERDA;
```

Los tipos enumerados se prestan mucho a su uso con sentencias `switch` en las que se hace una cosa distinta para cada uno de sus posibles valores. Para ellas hay una sintaxis más tradicional (que es la misma que existe en los lenguajes de programación C y C++), y una nueva (*rule switch*), que se introdujo a partir de una determinada versión de Java. Se muestra un ejemplo a continuación, expresado con ambas variantes.

Sintaxis clásica	Sintaxis <i>rule switch</i>
<pre>Direccion dir = Direccion.ABAJO; int x = 0; int y = 0;</pre>	
<pre>switch (dir) { case IZQUIERDA: x--; break; case DERECHA: x++; break; case ARRIBA: y--; break; case ABAJO: y++; break; }</pre>	<pre>switch (dir) { case IZQUIERDA -> x--; case DERECHA -> x++; case ARRIBA -> y--; case ABAJO -> y++; }</pre>
<pre>System.out.printf("Fin: (%d,%d)\n", x, y);</pre>	

Actividad Complemento 1.6

Crea un programa con un *array* con tipo base *Direccion*, que contenga varias direcciones (elementos de tipo *Direccion*) que representan sucesivos desplazamientos desde una posición inicial. Por ejemplo:

```
Direccion[] desplazamientos = {
    Direccion.ARRIBA,
    Direccion.DERECHA,
    Direccion.DERECHA,
    Direccion.ABAJO,
    Direccion.IZQUIERDA
}
```

Define dos variables *x* e *y* con un valor inicial cualquiera. Utiliza un bucle *for* para obtener uno a uno los desplazamientos contenidos en el *array*, y dentro de él una sentencia *switch* como la anterior para calcular la nueva posición, a partir de la anterior, después de cada desplazamiento. Haz que el programa escriba al principio la posición inicial y al final la posición final.

Pueden utilizarse cláusulas *case* con más de un valor, y cláusulas *default*, como se muestra en los siguientes ejemplos.

El primero es con el tipo *Direccion* ya visto.

Sintaxis clásica	Sintaxis <i>rule switch</i>
<pre>Direccion dir = Direccion.IZQUIERDA; switch (dir) { case IZQUIERDA: case DERECHA: System.out.println("Horizontal."); break; case ARRIBA: case ABAJO: System.out.println("Vertical."); break; }</pre>	<pre>switch (dir) { case IZQUIERDA, DERECHA -> System.out.println("Horizontal."); case ARRIBA, ABAJO -> System.out.println("Vertical."); }</pre>

El siguiente es con un nuevo tipo *DiaSemana* para los días de la semana.

Sintaxis clásica	Sintaxis <i>rule switch</i>
<pre>package dias; public class Dias { enum DiaSemana { LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO } public static void main(String[] args) { DiaSemana dia = DiaSemana.VIERNES; switch (dia) {</pre>	<pre>switch (dia) {</pre>

<pre> case SABADO: case DOMINGO: System.out.println("Festivo"); break; default: System.out.println("Laborable"); } </pre>	<pre> case SABADO, DOMINGO -> System.out.println("Festivo"); default -> System.out.println("Laborable"); } </pre>
<pre> } } </pre>	

8.3. Iteración sobre todos los valores de un tipo enumerado

Un tipo enumerado se implementa en Java con una clase con un conjunto finito y cerrado de posibles valores. Esta clase tiene varios métodos:

- El método estático `values()`, que devuelve un *array* con todos los posibles valores.
- El método no estático `name()`, que devuelve una descripción textual del objeto.
- El método `ordinal()`, que devuelve un número con la posición del objeto en la lista de posibles valores.

```

enum Direccion {
    IZQUIERDA, DERECHA, ARRIBA, ABAJO
}

for (Direccion dir : Direccion.values()) {
    System.out.printf("%d: %s\n", dir.ordinal(), dir.name());
}

```

La salida del programa anterior es la siguiente.

```

0: IZQUIERDA
1: DERECHA
2: ARRIBA
3: ABAJO

```

8.4. Modificación y copia de los contenidos de un *array*

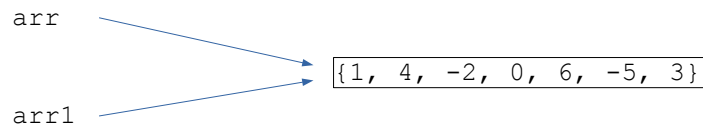
Se pueden modificar los contenidos de un *array* en un método al que se le pasa el *array* en un parámetro, como en el siguiente ejemplo, cuya salida se muestra al lado.

<pre> int[] arr = {1, 4, -2, 0, 6, -5, 3}; escribeArray(arr); System.out.println(); sumaEntero(arr, 4); escribeArray(arr); System.out.println(); </pre>	<pre> {1, 4, -2, 0, 6, -5, 3} {5, 8, 2, 4, 10, -1, 7} </pre>
---	--

Acceso a Datos. Complemento 1: Programación básica y utilización de objetos en Java

Al método `escribeArray` se le pasa una referencia al *array* `arr`. Cualquier variable de un tipo no básico contiene una referencia a un objeto. Y los *arrays*, en Java, son un tipo particular de objetos.

Con la sentencia `int[] arr1 = arr` no se crea un nuevo *array* `arr`, sino que se asigna a `arr1` una referencia al mismo objeto que `arr`.

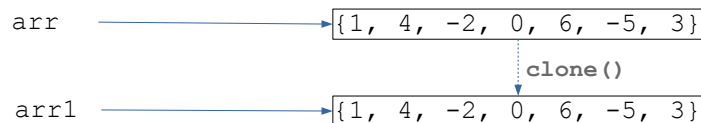


Por ello, como se puede comprobar con el siguiente programa, las operaciones hechas con `arr1` se pueden ver en `arr`, porque ambas variables contienen una referencia al mismo objeto.

```
int[] arr = {1, 4, -2, 0, 6, -5, 3};
int[] arr1 = arr;
sumaEntero(arr1, 4);
escribeArray(arr);
System.out.println();
escribeArray(arr1);
System.out.println();
```

```
{5, 8, 2, 4, 10, -1, 7}
{5, 8, 2, 4, 10, -1, 7}
```

En otros casos puede interesar no modificar el *array* original, sino una copia, dejando el *array* original inalterado. Se puede hacer una copia de un *array* con el método `clone`, como en el siguiente ejemplo. Con `clone` se crea un objeto nuevo al que referencia `arr1`, y `arr` sigue apuntando al mismo objeto, que queda inalterado.



```
int[] arr = {1, 4, -2, 0, 6, -5, 3};
int[] arr1 = arr.clone();
sumaEntero(arr1, 4);
escribeArray(arr);
System.out.println();
escribeArray(arr1);
System.out.println();
```

```
{1, 4, -2, 0, 6, -5, 3}
{5, 8, 2, 4, 10, -1, 7}
```

Actividad Complemento 1.7

Añade a la clase de ejemplo anterior `OperArray` un método `invertir` que invierta los contenidos de un *array*. Si se le pasa, por ejemplo, un *array* cuyos contenidos son `{1, 2, 3}`, debe cambiar sus contenidos para que sean `{3, 2, 1}`. El método debe realizar los cambios sobre el propio *array*, no crear y devolver un *array* nuevo.

Asegúrate de que el método hace lo apropiado cuando se le pasa un *array* `null`.

En el método `main`, escribe un *driver* que al principio cree un *array* con los números del 4 al 24. Después debe crear una copia de este *array*, utilizando `clone`, y después debe darle la vuelta a sus contenidos (los de esta última copia). Por último, debe escribir ambos *arrays*, el inicial y la copia,

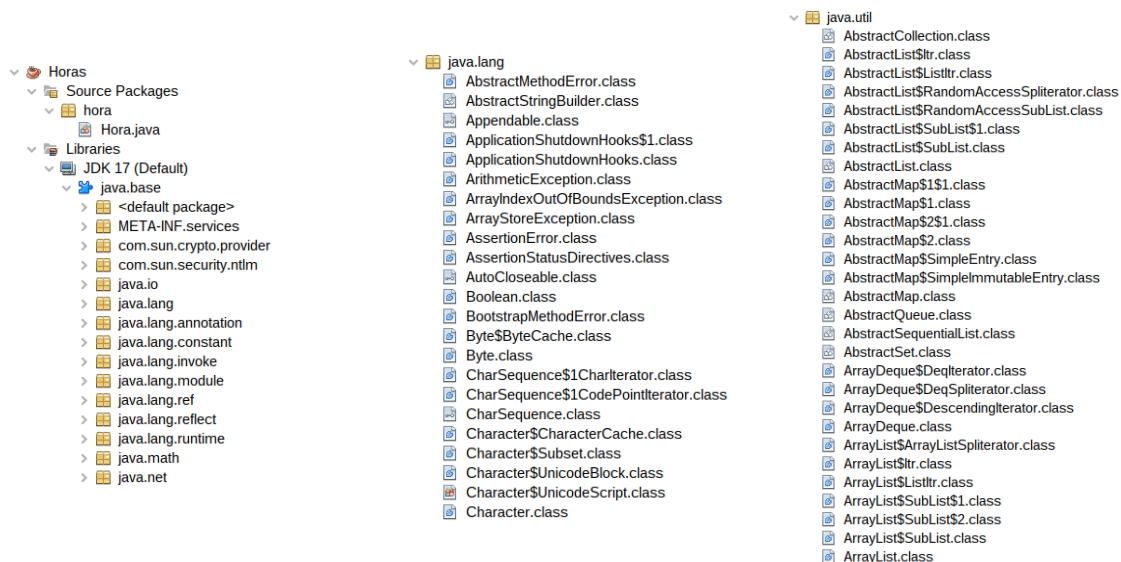
Actividad Complemento 1.8

En el método `main`, escribe un *driver* para la clase `OperArray` que, utilizando métodos estáticos de esta clase,

1. Crea un *array* con los números del 4 al 24.
2. Crea una copia de este *array*, utilizando `clone`.
3. Da la vuelta a los contenidos de esta copia.
4. Escribe el *array* original y su copia, que tendrá los mismos contenidos pero al revés.

9. Biblioteca estándar de clases de Java

La biblioteca estándar de clases de Java forma parte del JDK de Java y está estructurada en una jerarquía de paquetes. Esta jerarquía se puede visualizar dentro de cualquier proyecto, en el apartado “Libraries”, dentro del JDK utilizado, y dentro de él en el módulo “java.base”. Dentro de cada paquete se pueden ver las clases incluidas en él.



Directivas `import`

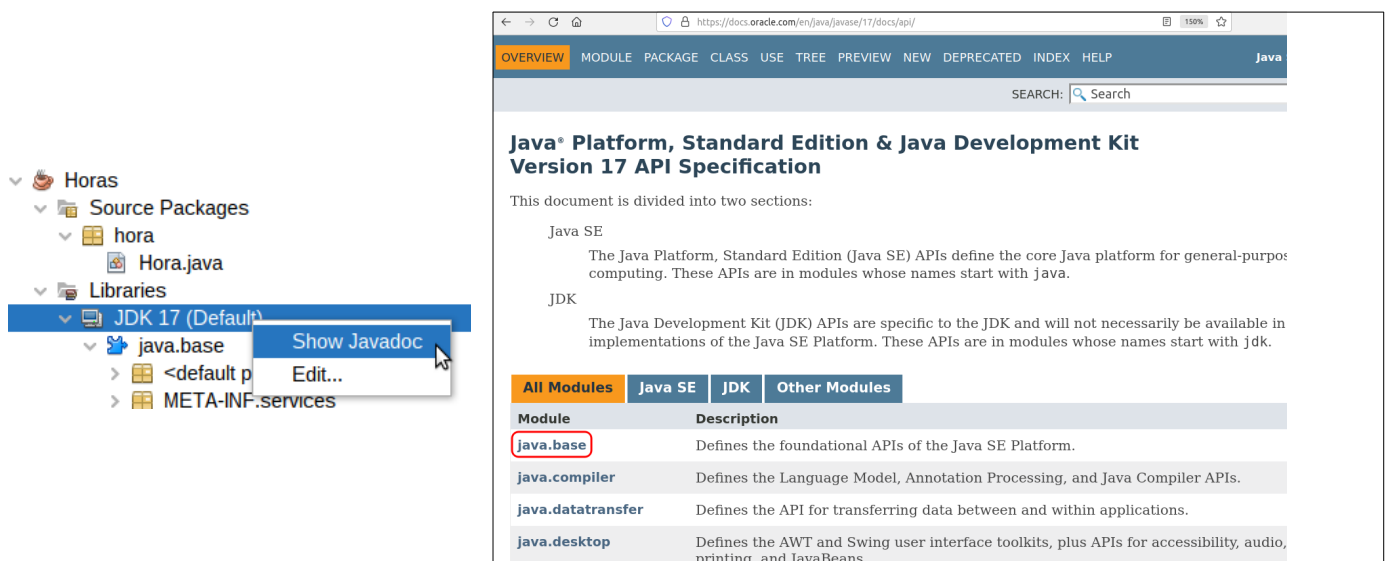
Para utilizar en una clase otra clase de un paquete distinto al de la propia clase (lo que es siempre el caso cuando se utiliza una clase de la biblioteca estándar de clases de Java), hay que incluir al principio una directiva `import`.

Ya se han visto algunas en los programas de ejemplo:

```
import java.util.InputMismatchException
import java.util.Scanner
```

Esto no es necesario para las clases dentro de `java.lang`, donde están clases básicas del lenguaje, como por ejemplo `String` y las clases *wrapper* para los tipos básicos: `Integer`, `Long`, `Short`, `Float`, `Double`, `Byte`, `Character` y `Boolean`.

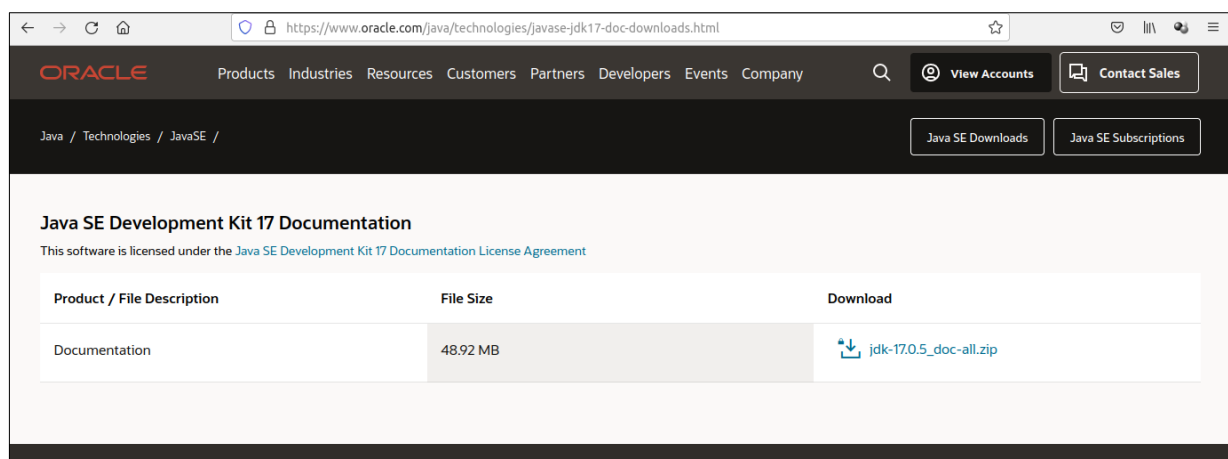
La biblioteca estándar de clases de Java está documentada en un conjunto de documentos en formato HTML, disponibles públicamente en Internet. Estos son los llamados Javadocs. Pulsando con el botón derecho del ratón, se puede mostrar esta documentación oficial. En este ejemplo, se puede ver que corresponde a la versión del JDK utilizada en el proyecto, la 17. La documentación de las clases de la biblioteca estándar de clases de Java está dentro de la del módulo `java.base`.



The image shows two side-by-side screenshots. On the left, an IDE's project explorer displays a tree structure with 'Libraries' expanded, showing 'JDK 17 (Default)' and its sub-packages like 'java.base'. A context menu is open over 'JDK 17 (Default)' with the 'Show Javadoc' option highlighted. On the right, a browser window displays the Oracle Java 17 API Specification page. The page title is 'Java® Platform, Standard Edition & Java Development Kit Version 17 API Specification'. It includes a search bar and a table of modules. The 'java.base' module is highlighted in the table.

Module	Description
java.base	Defines the foundational APIs of the Java SE Platform.
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
java.datatransfer	Defines the API for transferring data between and within applications.
java.desktop	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, printing, and JavaBeans.

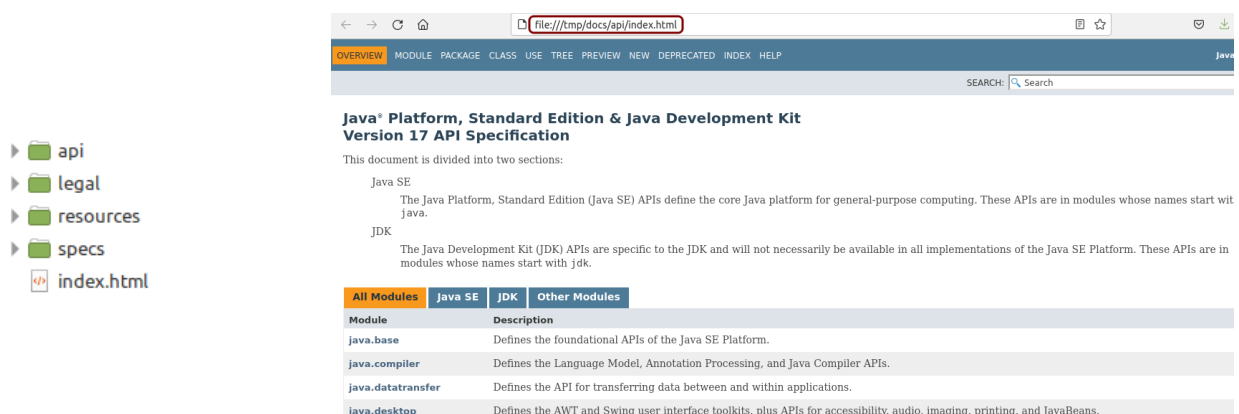
También se pueden descargar los Javadocs para consultarlos en local, sin necesidad de conexión a Internet. Para ello se puede hacer la búsqueda, por ejemplo, por texto “Javadocs Java 17 download”. Esto conducirá, seguramente, a la página web <https://www.oracle.com/java/technologies/javase-jdk17-doc-downloads.html>.



The image shows a screenshot of the Oracle website's 'Java SE Development Kit 17 Documentation' download page. The page has a dark header with the Oracle logo and navigation links. Below the header, there's a section titled 'Java SE Development Kit 17 Documentation' with a link to the 'Documentation License Agreement'. A table lists the available download:

Product / File Description	File Size	Download
Documentation	48.92 MB	jdk-17.0.5_doc-all.zip

Se puede descargar el fichero y entonces se tienen los siguientes contenidos dentro del nuevo directorio que se crea. Abriendo `index.html` en un navegador, se tiene acceso a los mismos contenidos disponibles en la web, pero en local, sin necesidad de acceder a Internet.



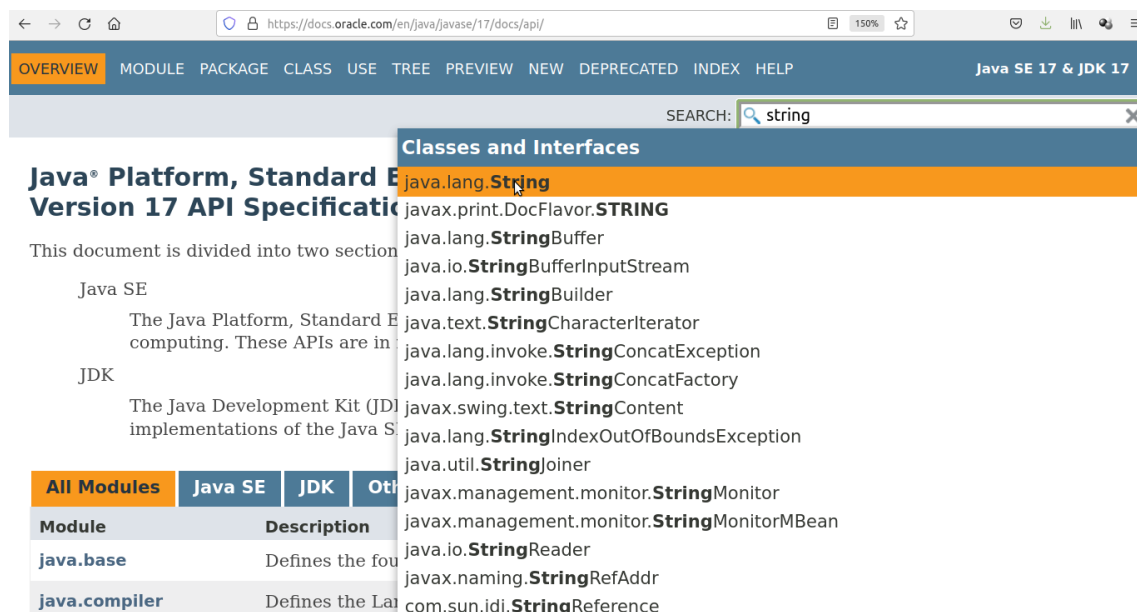
The image shows two side-by-side screenshots. On the left, a file explorer displays a directory structure with folders like 'api', 'legal', 'resources', and 'specs', and a file named 'index.html'. On the right, a browser window displays the local 'index.html' file, which is a copy of the Oracle Java 17 API Specification page. The browser's address bar shows the local file path: 'file:///tmp/docs/api/index.html'.

10. La clase String para cadenas de caracteres

Los objetos de la clase `String` contienen cadenas de caracteres en codificación UTF-16. Esta es una codificación de Unicode. Otras codificaciones de Unicode son UTF-8 y UTF-32.

La longitud en caracteres del `String` se puede obtener con el método `length()`. El método `getBytes()` permite obtener un `array` de bytes (`byte[]`) que representan el `String` en UTF-16 (por defecto) o en otras codificaciones de texto.

Se puede consultar su documentación buscando por el nombre de la clase. Cuando se introduce `String`, salen muchos resultados. Pero interesa la clase `java.lang.String`. El prefijo `java.lang` indica que es una clase de la biblioteca estándar de clases de Java.



Hay diversas maneras de crear un `String`. La más directa ya se ha utilizado en muchos ejemplos:

```
String saludo = "Hola";
```

También se puede crear un objeto de tipo `String` con `new`, aunque no se suele hacer de esta manera.

```
String saludo = new String("Hola");
```

Y también se puede crear a partir de un `array` de `char`:

```
char data[] = {'H', 'o', 'l', 'a'};
String str = new String(data);
```

Con la clase `String` se puede utilizar el operador `+` para concatenar cadenas, de la manera que se muestra en los siguientes ejemplos.

```
String nombre = "Titanio";
String simbolo = "Ti";
int numAtomico = 22;
double masaAtomica = 47.867;

System.out.println("Elemento " + nombre + " (" + simbolo
    + "), número atómico: " + numAtomico + ", masa atómica: " + masaAtomica);
```

Se puede utilizar el truco que se muestra a continuación para convertir datos de diversos tipos a un `String`.

```
int numAtomico = 22;
double masaAtomica = 47.867;
String numAtomicoS = "" + numAtomico;
String masaAtomicaS = "" + masaAtomica;
```

Una vez que se sabe crear objetos de tipo `String` por distintos medios, se plantea la cuestión de la igualdad entre distintos `String`.

Primero conviene precisar algunos términos y poner en cuestión algunas posibles ideas preconcebidas.

Operador == para tipos básicos y objetos

Con tipos básicos como `int`, `double` o `float`, el operador `==` devuelve `true` siempre que las dos cosas que se comparan tengan el mismo valor. Es decir, comprueba la **igualdad** de las dos cosas que se comparan.

<pre>int a = 2; int b = 2; System.out.printf("2 == 2: %b\n", 2 == 2); System.out.printf("a == 2: %b\n", a == 2); System.out.printf("a == b: %b\n", a == b);</pre>	<pre>2 == 2: true a == 2: true a == b: true</pre>
--	---

Las cosas que se comparan en las sentencias de ejemplo anteriores (las variables `a` y `b` y el literal `2`) son iguales, es decir, tienen igual valor, pero no son la misma cosa, es decir, no son idénticas. Las variables `a` y `b` ocupan distintos lugares en la memoria, y el valor literal `2` ocupa otro lugar en la memoria o en un registro del microprocesador.

También se puede utilizar el operador `==` con objetos de la clase `String`.

<pre>String a = "hola"; String b = "hola"; String c = new String("hola"); System.out.printf("\"hola\" == \"hola\": %b\n", "hola" == "hola"); System.out.printf("a == \"hola\": %b\n", a == "hola"); System.out.printf("a == b: %b\n", a == b); System.out.printf("c == \"hola\": %b\n", c == "hola"); System.out.printf("a == c: %b\n", a == c);</pre>	<pre>"hola" == "hola": true a == "hola": true a == b: true c == "hola": false a == c: false</pre>
---	---

En este caso particular, podría esperarse que el operador `==` devolviera siempre `true` para comparaciones entre estas variables y el literal `"hola"`. Pero no siempre es así. No lo es, en particular, con la variable `c`.

Eso es porque el operador `==`, cuando se utiliza con objetos, comprueba la **identidad** y no la igualdad de los objetos. Es decir, comprueba que son el mismo objeto, es decir, que están en la misma posición de memoria.

Si se quiere verificar la igualdad entre objetos, hay que utilizar el **método `equals()`** en lugar del operador `==`.

<pre>String a = "hola"; String b = "hola"; String c = new String("hola"); System.out.printf("%b\n", "hola".equals("hola")); System.out.printf("%b\n", a.equals("hola")); System.out.printf("%b\n", a.equals(b)); System.out.printf("%b\n", c.equals("hola")); System.out.printf("%b\n", a.equals(c));</pre>	<pre>true true true true true</pre>
--	-------------------------------------

Una última aclaración. Pero entonces, ¿por qué el operador `==` devolvía `true` con las comparaciones entre `a`, `b` y `"hola"`? Las variables de tipo `String` son en realidad referencias a un lugar de la memoria en el que está almacenado un objeto de esta clase. Cuando se asigna un valor a una variable de tipo `String`, la máquina virtual de Java no crea un nuevo objeto `String` en memoria si ya existe uno con el mismo valor, sino que le asigna una referencia al que ya existe. Pero no se puede confiar en que siempre sea o vaya a seguir siempre siendo así. Por lo tanto, para comparar el valor de dos `String`, siempre hay que utilizar el método `equals()`.

Algunos de los métodos más interesantes de la clase `String` son los siguientes. Se ejecutan sobre un `String` dado, que en adelante se denominará “el `String`”, “la cadena” o `this`.

Cuadro Complemento 1.1: Métodos de la clase *String*

<code>int length()</code>	Devuelve la longitud en caracteres (no en bytes).
<code>char charAt(int index)</code>	Devuelve el carácter (char) en la posición dada (index). La primera posición es 0.
<code>int compareTo(String anotherString)</code>	<p>Compara el String con otro dado (anotherString). Devuelve:</p> <ul style="list-style-type: none"> • Un número positivo si es mayor en orden alfabético. • Un número negativo si es menor en orden alfabético. • 0 si son iguales
<code>int compareToIgnoreCase(String str)</code>	Igual que el anterior, pero no diferencia entre mayúsculas y minúsculas.
<code>boolean contains(String str)</code>	<p>Devuelve:</p> <ul style="list-style-type: none"> • true si <code>this</code> contiene a <code>str</code>, es decir, si <code>str</code> es una subcadena de <code>this</code>. • false en caso contrario.
<code>boolean startsWith(String prefix)</code>	<p>Devuelve:</p> <ul style="list-style-type: none"> • true si <code>this</code> comienza con <code>prefix</code> • false en caso contrario.
<code>boolean endsWith(String suffix)</code>	<p>Devuelve:</p> <ul style="list-style-type: none"> • true si <code>this</code> termina con <code>suffix</code>. • false en caso contrario.
<code>boolean equals(Object anObject)</code>	<p>Ya se ha hablado de este método. Devuelve:</p> <ul style="list-style-type: none"> • true si <code>anObject</code> es un String y es igual (no necesariamente idéntico) a <code>this</code>. • false en caso contrario.
<code>boolean equalsIgnoreCase(Object anObject)</code>	Lo mismo, pero sin diferenciar entre mayúsculas y minúsculas.
<code>int indexOf(int ch)</code>	Varios métodos que devuelven la primera posición en que aparece un carácter <code>ch</code> o un String <code>str</code> . Los que tienen el parámetro <code>fromIndex</code> buscan a partir de la posición dada por ese índice. El índice 0 representa la primera posición. Si no aparece, devuelven -1.
<code>int indexOf(int ch, int fromIndex)</code>	
<code>int indexOf(String str)</code>	
<code>int indexOf(String str, int fromIndex)</code>	
<code>int lastIndexOf(int ch)</code>	Varios métodos que devuelven la última posición en que aparece un carácter <code>ch</code> o un String <code>str</code> . Los que tienen el parámetro <code>fromIndex</code> buscan hacia atrás a partir de la posición dada por ese índice. El índice 0 representa la primera posición. Si no aparece, devuelven -1.
<code>int lastIndexOf(int ch, int fromIndex)</code>	
<code>int lastIndexOf(String str)</code>	
<code>int lastIndexOf(String str, int fromIndex)</code>	
<code>boolean isEmpty()</code>	<p>Devuelve:</p> <ul style="list-style-type: none"> • true si la cadena tiene longitud 0. Es decir, si es igual a "". • false en caso contrario.
<code>boolean isBlank()</code>	<p>Devuelve:</p> <ul style="list-style-type: none"> • true si la cadena solo tiene caracteres que representan un espacio en blanco. • false en caso contrario. <p>El carácter más habitual que representa un espacio en blanco es ' ', pero no es el único en Unicode. Para más información se puede consultar la documentación del método <code>public static boolean isWhitespace(int codePoint)</code> de la clase <code>Character</code>.</p>
<code>String repeat(int count)</code>	Devuelve el resultado de concatenar el String el número de veces dado por <code>count</code> .

Acceso a Datos. Complemento 1: Programación básica y utilización de objetos en Java

<code>String replace(char oldChar, char newChar)</code>	Devuelve un <code>String</code> resultante de sustituir todas las ocurrencias del <code>oldChar</code> por el <code>newChar</code> .
<code>String replace(CharSequence target, CharSequence replacement)</code>	Devuelve un <code>String</code> resultante de sustituir la primera ocurrencia de la cadena de caracteres <code>target</code> por la cadena de caracteres <code>replacement</code> . Se puede pasar un <code>String</code> para ambos parámetros.
<code>String replaceAll(String regex, String replacement)</code>	Devuelve un <code>String</code> resultante de sustituir todas las ocurrencias de <code>regex</code> por <code>replacement</code> . <code>regex</code> puede ser una cadena literal o una expresión regular. Más adelante se estudiarán las expresiones regulares. Por ahora baste decir que una cadena literal es un caso particular de expresión regular.
<code>String replaceFirst(String regex, String replacement)</code>	Igual que el método anterior, pero solo se sustituye la primera ocurrencia de <code>regex</code> .
<code>String[] split(String regex)</code>	Divide el <code>String</code> en varios, sirviendo cada ocurrencia de <code>regex</code> como separador entre ellos, y devuelve un <code>array</code> con todos ellos. Como ya se ha comentado, una cadena de caracteres literal es un caso particular de expresión regular.
<code>String strip()</code>	Devuelven el <code>String</code> resultante de eliminar los espacios en blanco que pueda tener al principio y al final.
<code>String trim()</code>	Es preferible <code>strip</code> porque, al contrario que <code>trim</code> , está preparado para Unicode. Reconoce como caracteres en blanco no solo " " y los otros que <code>trim</code> reconoce como tales, sino otros tipos de caracteres en blanco de Unicode.
<code>String stripLeading()</code>	Funciona como <code>strip</code> , pero solo elimina espacios en blanco del principio.
<code>String stripTrailing()</code>	Funciona como <code>strip</code> , pero solo elimina espacios en blanco del final.
<code>String substring(int beginIndex)</code>	Devuelve la subcadena que comienza en el índice dado y llega hasta el final.
<code>String substring(int beginIndex, int endIndex)</code>	Devuelve la subcadena que comienza en el primer índice dado y llega hasta el índice anterior al índice dado.
<code>String toLowerCase()</code>	Devuelve el <code>String</code> en minúsculas.
<code>String toUpperCase()</code>	Devuelve el <code>String</code> en mayúsculas.

Actividad Complemento 1.9

Crea un programa que encuentre todas las ocurrencias de un `String` dentro de otro. Utilizar para ello el método `indexOf(String str, int fromIndex)`. La primera vez debe llamarse con `fromIndex=0`. Las siguientes veces debe llamarse con la posición siguiente a la devuelta la vez anterior. Se termina cuando se obtiene `-1`, lo que significa que no se ha encontrado el `String`.

Actividad Complemento 1.10

Obtener la cadena de caracteres (`String`) inversa a una dada. Por ejemplo: la cadena inversa de "ordenador" es "rodanedro". ¿Se te ocurre más de una manera de hacerlo?

Actividad Complemento 1.11

Averiguar si una cadena es un palíndromo. Un palíndromo es una cadena de caracteres que se lee igual del derecho que del revés. Por ejemplo: "reconocer". Puedes utilizar lo que has hecho para el anterior ejercicio. O puedes hacerlo de manera más eficiente, sin necesidad de crear una cadena de caracteres nueva.

Actividad Complemento 1.12

Igual que el ejercicio anterior, pero eliminando antes todos los espacios. De esa forma, se reconocerán como palíndromos "se van a sus naves" o "yo hago yoga hoy". Averiguar si una cadena es un palíndromo. Un palíndromo es una cadena de caracteres que se lee igual del derecho que del revés. Por ejemplo: "reconocer". Puedes utilizar lo que has hecho para el anterior ejercicio. O puedes hacerlo de manera más eficiente, sin necesidad de crear una cadena de caracteres nueva.

Actividad Complemento 1.13

Como el ejercicio anterior, pero ignorando signos de puntuación y sustituyendo previamente vocales acentuadas por las correspondientes sin acentuar. De esta forma, deberían reconocerse como palíndromos todos los que se muestran en https://es.wikipedia.org/wiki/Pal%C3%ADndromo#En_espa%C3%B1ol. Haz un programa lo más genérico que sea posible. Convierte previamente todo a mayúsculas o minúsculas para que una letra mayúscula y otra minúscula se reconozcan como iguales. Utiliza un *array* de *char* para especificar todos los caracteres que hay que eliminar previamente, donde estarán todos los símbolos de puntuación. Puedes utilizar dos *String* para las correspondencias entre vocales con acentos u otros signos diacríticos y las mismas sin ellos. Por ejemplo: "áéíóüü" y "aeiouu".

Actividad Complemento 1.14

Crea un programa que sustituya cadenas *%num* por cadenas dentro de un *array* de *String* *strArr*. Debe sustituir *%1* por *strArr[0]*, *%2* por *strArr[1]*, y así hasta *%n* por *strArr[n-1]*, donde *n* es la longitud del *array* *strArr*. Utiliza el método *replaceAll* de la clase *String*.

Por ejemplo: se define inicialmente un *array* *String[] strArr* como sigue:

```
String[] strArr={ "Isabel", "Juan", "estaba muy feliz", "había comprado una bici" }
```

Y una variable *String* *texto* como sigue:

```
String texto="%1 le dijo a %2 que %3. %2 preguntó por qué %3. %1 respondió que %4." }
```

El programa debe sustituir las ocurrencias de *%1*, *%2*, *%3* y *%4* por los correspondientes *String*, para obtener un *String* con el siguiente contenido:

```
"Isabel le dijo a Juan que estaba muy feliz. Juan preguntó por qué estaba muy feliz.
Isabel respondió que había comprado una bici."
```

11. La clase *Math* para operaciones matemáticas

La clase *Math* contiene muchos métodos estáticos (*static*) para realizar muchas operaciones con datos de los distintos tipos básicos numéricos existentes en Java. A saber: *double*, *float*, *double* e *int*, y. Se muestran a continuación algunos de ellos.

		<i>double</i>	<i>float</i>	<i>int</i>	<i>long</i>
<i>abs</i> (<i>x</i>)	Valor absoluto	✓	✓	✓	✓
<i>max</i> (<i>x</i> , <i>y</i>)	Máximo de dos números	✓	✓	✓	✓
<i>min</i> (<i>x</i> , <i>y</i>)	Mínimo de dos números	✓	✓	✓	✓
<i>round</i> (<i>x</i>)	Valor entero más cercano al argumento, <i>long</i> si es un <i>double</i> o <i>int</i> si es un <i>float</i> .	✓	✓		
<i>ceil</i> (<i>x</i>)	Número más próximo por debajo que es igual a un entero.	✓			
<i>floor</i> (<i>x</i>)	Número más próximo por encima que es igual a un entero.	✓			
<i>rint</i> (<i>x</i>)	Valor más cercano al argumento, de tipo <i>double</i> e igual a un número	✓			

	entero.				
<code>signum(x)</code>	Signo: 0, 1.0 o -1.0, según el argumento sea 0, positivo, o negativo.	✓	✓		
<code>sqrt(x)</code>	Raíz cuadrada	✓			
<code>cbrt(x)</code>	Raíz cúbica	✓			
<code>random()</code>	Devuelve un valor aleatorio mayor o igual que 0 y menor que 1.	✓			

import static

Los métodos estáticos del paquete `Math` se pueden referenciar de varias maneras.

1. Anteponiendo el nombre del paquete. Por ejemplo: `Math.sqrt(2)`.
2. Con `import static java.lang.Math.sqrt` al principio, no hace falta anteponer el nombre de la clase en las llamadas a los métodos estáticos. Se podría usar directamente `sqrt(2)`.

Si se usan varios métodos de la clase `Math`, podría ser más cómodo importar estáticamente todos los métodos con `import static java.lang.Math.*`. Entonces se pueden utilizar todos los métodos sin anteponerles el nombre de la clase: `sqrt(2)`, `abs(-2)`, `max(a, b)`.

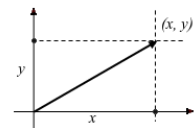
En general, la práctica más habitual es anteponer el nombre de la clase al del método. Por ejemplo: `Math.sqrt(2)`.

Actividad Complemento 1.15

Crea un programa que lea por teclado las coordenadas x e y de un punto. Ambas pueden ser negativas. Si la coordenada x es negativa, el punto está a la izquierda del eje vertical (y). Si la coordenada y es negativa, el punto está por debajo del eje horizontal (x).

Una vez leídos los valores de x e y , el programa debe calcular y mostrar:

- Distancia al eje x , y distancia al eje y .
- Distancias máxima y mínima a los ejes. Para ello debe utilizarse un método de la clase `Math`, no una sentencia condicional `if`.
- De qué eje está más cerca el punto, si el eje x o el eje y .
- Distancia del punto al origen de coordenadas.



12. La clase Random para generación de números aleatorios

La clase `Random` permite obtener números aleatorios. Los números no son realmente aleatorios, sino que se generan a partir de un valor inicial o semilla. A partir de una misma semilla o valor inicial, se obtiene siempre la misma secuencia de números. Por ello se dice que los números son pseudoaleatorios. Si como semilla o valor inicial se toma uno que dependa de la hora, y que tenga en cuenta hasta los microsegundos, cada vez que se genere una secuencia se obtendrá una secuencia diferente.

Cuadro Complemento 1.2: Métodos de la clase Random

<code>Random()</code> <code>Random(long seed)</code>	Creadores. El segundo establece una semilla determinada. El primero asigna un valor para la semilla que con mucha probabilidad será distinto que para cualquier otra invocación del constructor.
---	--

Acceso a Datos. Complemento 1: Programación básica y utilización de objetos en Java

<pre>int nextInt(int bound) int nextInt(int origin, int bound) int nextInt() boolean nextBoolean() double nextDouble(...) double nextFloat(...) double nextLong(...)</pre>	<p>Métodos que devuelven valores aleatorios del tipo en cuestión, que aparece en su nombre, con una distribución uniforme. Esto último quiere decir que todos los valores en el rango de posibles valores tienen igual probabilidad de ser obtenidos.</p> <p>Si se especifica un límite superior <code>bound</code>, se obtiene un valor entre 0 y ese, pero sin llegar a él. Si se especifica un límite inferior <code>origin</code>, se obtienen valores mayores o iguales que él.</p> <p>Existen métodos <code>nextDouble</code>, <code>nextFloat</code> y <code>nextLong</code> análogos a <code>nextInt</code>.</p> <p><code>nextDouble()</code> y <code>nextFloat()</code> devuelven un número entre 0 y 1.</p> <p><code>nextInt()</code> devuelve un número entre -2^{32} y $2^{32}-1$.</p>
<pre>void nextBytes(byte[] bytes)</pre>	<p>Rellena un <i>array</i> proporcionado con bytes con valores aleatorios.</p>
<pre>double nextGaussian() double nextGaussian(double mean, double stddev)</pre>	<p>Devuelven un número con una distribución de probabilidad normal. La probabilidad es más alta para números más cercanos a la media, y disminuye conforme se alejan de ella. Con el primer método, la media μ (<code>mean</code>) es 0 y la desviación estándar σ (<code>stddev</code>) es 1.</p> <p>Con el segundo se puede especificar la media y la desviación estándar.</p>

Actividad Complemento 1.16

Escribe un programa que escriba 20 números enteros aleatorios entre 1 y 90.

Actividad Complemento 1.17

Escribe un programa que escriba todos los números de 1 a 90, pero obtenidos de manera aleatoria y sin que ninguno se repita. Para ello, anotar los números que han salido en un *array* con tipo base `boolean`. Inicialmente, todas las posiciones del *array* contendrán un valor `false`. Cuando sale un número, hay que anotar el valor `true` en la posición correspondiente, para saber que ya ha salido. Si se genera un número aleatorio y ese número ya ha salido, hay que obtener el siguiente número que no haya salido, consultando el *array* a partir de la siguiente posición. Si se llega al final, hay que seguir por la primera posición del *array*. Hay que llevar la cuenta de los números que se han generado, para generar exactamente 90.

Escribir código genérico, de manera que 90 se defina en una constante de clase (`static final int`)

Actividad Complemento 1.18

Escribir un programa que seleccione y escriba un valor al azar de entre los que están definidos en un tipo enumerado. Utilizar el siguiente tipo enumerado, utilizado en ejemplos anteriores.

```
enum Direccion {
    IZQUIERDA, DERECHA, ARRIBA, ABAJO
}
```

Ayuda: `Direccion.values()` devuelve un *array* con todos los valores del tipo enumerado `Dirección`, como se vio con el siguiente programa de ejemplo.

```
for (Direccion dir : Direccion.values()) {
    System.out.printf("%d: %s\n", dir.ordinal(), dir.name());
}
```