

Salida por consola (I)

- Una de las operaciones más básicas que proporciona el API es la que nos permite mostrar mensajes de texto en la consola del sistema
- Para ello, disponemos de la clase **System** que proporciona acceso a diferentes recursos del sistema. Uno de sus atributos, **out**, está “conectado” a la **salida estándar** del sistema (generalmente la pantalla o consola) y nos proporciona una serie de métodos para *escribir* en ella:
 - `System.out.print(mensaje)`, mostrará la cadena de texto *mensaje*.
 - `System.out.println(mensaje)`, mostrará la cadena de texto *mensaje* y añadirá un salto de línea al final
 - `System.out.printf(formato, lista_valores)`, permite realizar salidas formateadas. *formato* será una cadena de texto que incluirá **campos de reemplazo** que serán sustituidos por los valores de la *lista_valores*

Salida por consola (II)

- La consola emplea un **cursor** que determina la primera posición a partir de la cual se va a imprimir un nuevo mensaje
- A medida que se van imprimiendo nuevos caracteres, este cursor se va desplazando hasta quedar a continuación del último carácter impreso
- Vimos como en Java podemos hacer uso de una serie de **secuencias de escape**, que pueden ir incluídas en nuestros textos, y que alteran la forma en que se imprimen esos mensaje al modificar la posición del cursor
- Una de estas secuencias de escape, *nueva línea* (**\n**), se suele incluir habitualmente al final de los mensajes, para que la nueva impresión comience en la primera posición de la siguiente línea
- Por esta razón, la clase *System* de Java incluye ambos métodos, **print()** y **println()**, el segundo de los cuales incluye el salto de línea final

Salida por consola (III)

❑ Componiendo mensajes...

- El argumento *mensaje* de cualquiera de los métodos de impresión será la cadena de caracteres a imprimir. Java tratará dicha cadena como un objeto de tipo *String*.
- Este objeto *String* podrá crearse a partir de literales de tipo cadena de caracteres y variables, concatenados mediante el operador (+). Con independencia de su tipo, Java obtendrá una representación “*textual*” de los valores de la variables para poder imprimirlas. Por ejemplo:

```
int edad = 9;  
System.out.println("Tengo " + edad + " años");
```

Java interpreta las cadenas "Tengo " y " años", encerradas **entre comillas**, como valores **literales**. Sin embargo, *edad* (**sin comillas**), se interpreta como una **variable**. Java convierte su valor en un texto y los concatena

Salida por consola (III)

❑ Mostrando texto con *printf*...

- El método *System.out.printf* (f significa *formato*) permite mostrar datos con un formato específico (número de decimales, caracteres de relleno,...)
- Veamos un ejemplo:

```
System.out.printf("Hola %s!%n", "Mundo");
```

La salida de la sentencia anterior sería:

```
Hola Mundo!
```

- El primer argumento del método *printf*, en nuestro caso "Hola %s!%n", es un *String* que puede incluir texto y *secuencias de formato* (o *campos de reemplazo*). Cada una de estas secuencias, será reemplazada en el texto impreso por un valor o comportamiento. Por ejemplo, %s se sustituye por la cadena "Mundo", mientras que %n provoca un salto de línea

Salida por consola (IV)

- Las *secuencias de formato* tienen la siguiente sintaxis:

% [flags] [ancho] [.precisión] carácter_de_conversión ;

carácter conversión	imprime
c	caracter individual
C	caracter individual (mayúsculas)
b	booleano
B	booleano (mayúsculas)
d	número entero
f	número en punto flotante
e	número en punto flotante (notación científica)
s	cadena de caracteres
S	cadena de caracteres (mayúsculas)
n	salto de línea
tM	fecha y hora (M modificador de formato)

- flags*, establece ciertos modificadores de salida, especialmente con números
- ancho*, espacio de caracteres mínimo que ocupará el campo
- .precisión*, número de dígitos decimales en números de punto flotante

Salida por consola (V)

❑ Algunos ejemplos...

```
boolean b = true;
char c = 'x';
String s = "Texto de ejemplo";    // Creación de un objeto String

System.out.printf("Valores boolean: %b, char: %c y string: %s%n", b, c, s);
System.out.printf("En mayúsculas: %B, %C y %S", b, c, s);
```

```
Valores boolean: true, char: x y String: Texto de ejemplo
En mayúsculas: TRUE, X y TEXTO DE EJEMPLO
```

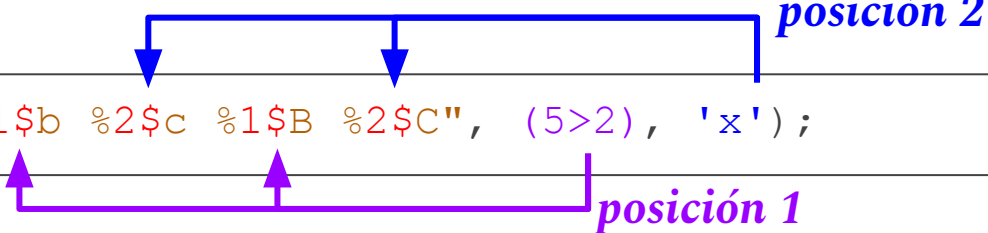
Fíjate como, por cada *campo de formato* (menos `%n` -> salto línea), se especifica un valor en la lista y se van emparejando en orden. Dichos valores pueden ser literales, variables o expresiones más complejas

```
System.out.printf("Valores boolean: %b, char: %c y string: %s%n", b, c, s);
```



Salida por consola (VI)

- Podemos también establecer directamente el **emparejamiento** entre un campo de formato y un valor de la lista. Para ello, indicaremos en el campo de formato la posición que ocupa en la lista el valor que nos interesa (empezando en 1). Por ejemplo:



```
System.out.printf("%1$b %2$c %1$B %2$C", (5>2), 'x');
```

La salida de la sentencia anterior será:

```
true x TRUE X
```

- También podemos especificar el ancho del campo en la salida, así como su alineación izquierda/derecha (derecha *por defecto*):

```
System.out.printf("'%1$10s' '%1$-10S'%n", "hola"); // campos de 10 caracteres
```

```
'      hola' 'hola      '
```

Salida por consola (VII)

- Emplearemos la secuencia de formato **%d** para la impresión de valores enteros (*byte, char, short, int, long*).
- Para números en punto flotante usaremos los códigos **%f** y **%e** (*notación científica*). Además, podremos especificar a cuántos dígitos decimales queremos **redondear** el valor
- En ambos casos, podremos indicar mediante el *flag* **' , '** si queremos que se muestre el separador de millares

```
System.out.printf("%d %f %n", 123, 5.128);    // un entero y un punto flotante
System.out.printf("%1$f %1$.2f %n", 123.456); // sin y con redondeo a 2 decimales
System.out.printf("% ,.2f %n", 12300.456);    // con redondeo y separador de millares
System.out.printf("% .2e %n", 123.456);       // con redondeo y notación científica
```

```
123 5,128000
123,456000 123,46
12.300,46
1,23e+02
```


Salida por consola (VIII)

- Por defecto, como separadores de decimales y millares, se emplearán los signos establecidos por la configuración regional del sistema
- Java proporciona una variante de *printf* en la que podemos pasarle como argumento la configuración regional que deseamos usar. Para ello, crearemos un objeto de la clase *java.util.Locale* con la configuración local deseada, o usaremos alguna de las que ya tiene predefinidas

```
import java.util.Locale;
. . .
System.out.printf("df: %,f %n", 12300.456); // config. reg. del sistema (defecto)
System.out.printf(Locale.US, "US: %,f %n", 12300.456); // aplica config. reg. US
                                                    // predefinida en Locale
Locale locES = new Locale("es", "ES"); // creamos config. regional para es_ES
System.out.printf(locES, "ES: %,f %n", 12300.456); // aplicamos config. es_ES
```

```
df: 12.300,456000
US: 12,300.456000
ES: 12.300,456000
```

Salida por consola (IX)

- Por último, también disponemos de secuencias de formato para valores de fechas y horas, permitiéndonos extraer campos individuales

carácter conversión	imprime
tH	hora
tM	minutos
tS	segundos
tp	am/pm
tz	zona horaria
tA	día de la semana (texto)
td	día del mes (numérico 2-dig)
tB	mes (texto)
tm	mes (numérico 2-dig)
tY	año (4-dig)
ty	año (2-dig)

Salida por consola (y XII)

❑ Ejemplo impresión de fecha/hora

```
//: PrintDateDemo.java
/**
 * Salida formateada de fecha/hora actual
 */

import java.time.ZonedDateTime;

class PrintDateDemo {
    public static void main(String[] args) {
        // Obtenemos fecha/hora actual con información de zona
        ZonedDateTime date = ZonedDateTime.now();

        System.out.printf("Hoy es %1$tA, %1$td de %1$tB de %1$tY %n", date);
        System.out.printf("Son las %1$tH:%1$tM:%1$tS [%1$tp] %n", date);
        System.out.printf("En la zona horaria %s [%tz] %n", date.getZone(), date);
    }
}

/* Output:
Hoy es lunes, 12 de agosto de 2019
Son las 00:23:01 [am] +0200
En la zona horaria Europe/Madrid [+0200]
*///:~
```

Entrada de datos (I)

- Aunque disponemos en Java de diferentes alternativas para leer la entrada de datos estándar, una de las formas más simples es a través de la clase *java.util.Scanner*, pues nos permite obtener directamente valores de tipos primitivos sin necesidad de realizar ninguna conversión
- Para poder utilizar cualquiera de los métodos de *Scanner*, deberemos primeramente crear un objeto de la misma. Algunos de sus métodos son:
 - *nextBoolean()*, lee un valor *boolean* introducido por el usuario
 - *nextByte()*, lee un valor de tipo *byte*
 - *nextDouble()*, lee un valor de tipo *double*
 - *nextFloat()*, lee un valor de tipo *float*
 - *nextInt()*, lee un valor de tipo *int*
 - *nextLine()*, lee toda la entrada como un *String*
 - *nextLong()*, lee un valor de tipo *long*
 - *nextShort()*, lee un valor de tipo *short*

Entrada de datos (II)

- La llamada a cualquiera de los métodos anteriores de *Scanner* es bloqueante, es decir, el programa se detendrá hasta que pulsemos la tecla *[Return]*, momento en el cual se devolverá el control al programa y se le “pasará” lo que haya introducido el usuario
- Si al llamar a alguno de esos métodos, el programa recibe un dato no convertible al tipo esperado, se producirá un error y el programa finalizará. Más adelante veremos cómo gestionar estos errores
- Al introducir valores en punto flotante, Java espera que lo hagamos utilizando la **configuración regional** del sistema. En nuestro caso, deberíamos emplear la **coma** en los datos de entrada para separ decimales (los literales en el programa usan el *punto* como separador decimal). Podemos modificar este comportamiento mediante el método *useLocale()* y establecer la configuración regional para la entrada que nos interese

Entrada de datos (y III)

❑ Ejemplo de uso de *java.util.Scanner*

```
//: ScannerDemo.java
import java.util.Scanner;
import java.util.Locale;

class ScannerDemo {
    public static void main(String[] args) {
        // Creamos el objeto Scanner
        Scanner entrada = new Scanner(System.in);

        // Cambiamos la configuración regional para usar '.' decimal
        // en lugar de ',' decimal (opcional)
        entrada.useLocale(Locale.US);

        System.out.println("Introduce tu nombre, edad y estatura (metros):");

        String nombre = entrada.nextLine();
        int edad = entrada.nextInt();
        double estatura = entrada.nextDouble(); // usamos separador decimal según config. regional

        System.out.println("Nombre: " + nombre);
        System.out.println("Edad: " + edad);
        System.out.println("Estatura (m): " + estatura);
    }
}
```