

## ENTRADA POR TECLADO

Hay varias formas de leer datos de teclado, nosotros utilizaremos por el momento únicamente dos: la clase Scanner y la clase Console.

Debemos saber antes de empezar:

- La pantalla es *System.out*.
- El teclado es *System.in*

Más adelante, entenderemos mejor qué es *System.out* y *System.in*.

## ENTRADA POR TECLADO CON LA CLASE SCANNER

La clase Scanner es una clase de la librería java que requiere, el uso de una sentencia import:

```
import java.util.Scanner;
```

las sentencias import se estudiarán más adelante, por el momento límitate a escribirla y observa que se escribe al principio del fichero (antes de la definición de clase)

La clase Scanner sirve para analizar texto. ¿De dónde sale el texto que analiza Scanner?. Hay varias fuentes, puede ser un fichero de texto, un String, ..., y lo que nos interesa ahora, puede ser el texto que introducimos por teclado.

¿Cómo indicamos que queremos que Scanner "lea del teclado"?

1. Se importa la clase Scanner

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

2. Se crea un objeto Scanner y al constructor se le pasa el objeto *System.in* que no es más que el teclado

```
new Scanner(System.in)
```

3. Se utilizan métodos de la clase Scanner para acceder al texto introducido por teclado.

```
sc.next()
```

Ejemplo:

```
import java.util.Scanner;//1
```

```
public class App {  
    public static void main(String[] args) throws Exception {  
        Scanner sc= new Scanner(System.in);//2  
        System.out.println("teclea una palabra");  
        String palabra= sc.next();//3  
        System.out.println("has tecleado la palabra :"+ palabra);  
    }  
}
```

## Dos formas de leer el texto del teclado con la clase Scanner

El teclado envía caracteres unicode hacia la CPU que llamaremos simplemente como "texto". Cada vez que pulsamos enter en el teclado enviamos una línea de texto al programa, en nuestro caso su destino es el objeto Scanner. El texto que tecleamos por tanto se va almacenando dentro del objeto Scanner asociado al teclado. A continuación dicho texto lo podemos analizar o leer de dos formas básicas:

- De token en token utilizando next() y derivados. Más o menos un token es una palabra. Es decir que leemos de palabra en palabra
- De línea en línea utilizando el método nextLine(). Una línea puede contener muchos tokens(muchas palabras).

También es posible mezclar ambas formas por ejemplo intercalando el uso de nextLine() y next() a nuestra conveniencia.

## Leer de token en token con el método next()

Un objeto Scanner almacena el texto en una tira de caracteres. Esta tira de caracteres la "ve" dividida en "tokens" (trozos). La división en tokens la hace considerando que hay un carácter llamado "separador" o "delimitador" que marca el fin de un token y el principio de otro. Por defecto, los delimitadores son:

- espacio en blanco ' '
- Tabulador '\t'
- Salto de línea '\n'.

El método de lectura más básico de la clase Scanner es con el método next() que lee un token y lo devuelve como un String.

Ejemplo: Prueba el siguiente código tecleando en la ejecución la frase "uno dos y tres"

```
import java.util.Scanner;
class Unidad2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Teclea tres palabra separadas por espacios: ");
        String miString1=sc.next();
        String miString2=sc.next();
        String miString3=sc.next();
        System.out.println("palabra1: "+miString1);
        System.out.println("palabra2: "+miString2);
        System.out.println("palabra3: "+miString3);
    }
}
```

T D T D T D T  
uno  dos  y  tres\n

T = Token  
D = delimitador  
\n = enter  
 = espacio en blanco

Si ejecuto este programa e introduzco por teclado "uno dos y tres" observo que al hacer 3 next() no llego a leer la palabra *tres* tal y como era de esperar.

```
Teclea tres palabras separadas por espacios:  
una dos y tres  
palabra1: una  
palabra2: dos  
palabra3: y
```

Analicemos el código anterior, suponemos que el usuario ve por pantalla "Teclea tres palabras separadas por espacios: " pero todavía no escribió nada:

1. El primer next() detecta que no tienen nada que leer y provoca que el programa "espere" a que el usuario escriba algo por teclado.
2. Cuando el usuario escribe en el teclado, todo se va almacenando en un buffer del teclado. En el momento que el usuario oprime la tecla enter el contenido del buffer del teclado, con el enter incluido, se envía al programa, es decir, se envía "un dos y tres\n"
3. ¿Dónde almacena nuestro programa el envío del teclado "un dos y tres\n"? En el objeto de clase Scanner sc. Sobre la información de este objeto, y no sobre el buffer del teclado, trabajan los métodos next
4. El primer next() empieza a leer de caracter en caracter y no para hasta encontrarse con un delimitador, el resultado es que forma el primer token "uno". Queda por procesar por tanto dentro del objeto Scanner " dos y tres \n". Fijate que el next() dejó en el buffer del scanner el delimitador (un espacio en blanco)
5. El segundo next() analiza " dos y tres \n". Cuando comienza el escaneo de un next() avanza hasta que encuentra el primer caracter no separador, por lo tanto,

desprecia el separador espacio en blanco del comienzo y coge el token "dos" ya que después de dos observa un separador y sabe que debe parar. Deja al programa lo siguiente para procesar " y tres \n"

6. El tercer next() analiza " y tres\n", desprecia el separador espacio en blanco del comienzo y coge el token "y" . Deja al programa lo siguiente para procesar " tres \n"
7. el programa sigue su ejecución, hace los println() y como no hay más instrucciones que lean del teclado "tres \n" se queda sin procesar. Tampoco importa en este caso.

Por lo tanto, vemos que next() reacciona frente a los separadores:

- Al empezar leer, despreciando todos los separadores hasta llegar a un caracter no separador.
- Una vez que comenzó a leer caracteres no separadores, si se encuentra un separador para de leer.

Ejemplo: razona que la siguiente tira de caracteres introducida por teclado leída con el programa anterior genera el mismo resultado porque la única diferencia son los delimitadores extra que se ignoran

"una \ndos\t \n\ny tres"

**Ejemplo:** Utilizando next() como en el ejemplo anterior, conseguir la siguiente ejecución.

```
Teclea tres palabra separadas por espacios:  
una dos y tres  
palabra1: una  
palabra2: dos  
palabra3: tres
```

Si queremos que el tercer String contuviera "tres", una solución es hacer 4 next(), pero el 3º no imprimimos su valor por pantalla.

```
import java.util.Scanner;
```

```
class Unidad2 {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Teclea tres palabra separadas por espacios: ");  
        String miString1=sc.next();  
        String miString2=sc.next();  
        sc.next();//para leer "y" pero no la imprimimos  
        String miString3=sc.next();  
    }  
}
```

```

        System.out.println("palabra1: "+miString1);
        System.out.println("palabra2: "+miString2);
        System.out.println("palabra3: "+miString3);
    }
}

```

## Otros métodos basados en next()

Todo lo que se introduce por teclado llega al objeto Scanner como caracteres de texto. El siguiente ejemplo acepta tres números por teclado pero no podemos hacer con ellos una suma aritmética ya que trata lo introducido como strings.

```

import java.util.Scanner;
class Unidad2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String primero, segundo, tercero;

        System.out.println("tres numeros: ");
        primero=sc.next();
        segundo=sc.next();
        tercero=sc.next();
        System.out.println("la suma de los tres es: "+(primero+segundo+tercero));
    }
}

```

```

C:\Users\pilt>java Unidad2
tres numeros:
1 2 3
la suma de los tres es: 123
C:\Users\pilt>

```

Si a continuación cambio las variables *String* por *int* obtengo un error de tipos ya que *next()* devuelve un *String* y no se puede meter en una variable *int*

```

import java.util.Scanner;
class Unidad2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int primero, segundo, tercero;

        System.out.println("tres numeros: ");
        primero=sc.next();
        segundo=sc.next();
        tercero=sc.next();
        System.out.println("la suma de los tres es: "+(primero+segundo+tercero));
    }
}

```

```

C:\Users\pilt>javac Unidad2.java
Unidad2.java:8: error: incompatible types: String cannot be converted to int
    primero=teclado.next();
                    ^

```

Por lo tanto, e tengo que convertir los String en un int, por ejemplo con Integer.parseInt() que ya utilizamos en ejemplos anteriores

```

import java.util.Scanner;
class Unidad2 {
    public static void main(String[] args) {

```

```

Scanner sc = new Scanner(System.in);
int primero, segundo, tercero;

System.out.println("tres numeros: ");
primero=Integer.parseInt(sc.next());
segundo=Integer.parseInt(sc.next());
tercero=Integer.parseInt(sc.next());
System.out.println("la suma de los tres es: "+(primero+segundo+tercero));
}
}

```

O más cómodo ya que usamos un objeto Scanner sería usar el método `nextInt()` que también se encarga de convertir el siguiente token a int

```

import java.util.Scanner;
class Unidad2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int primero, segundo, tercero;

        System.out.println("tres numeros: ");
        primero=sc.nextInt();
        segundo=sc.nextInt();
        tercero=sc.nextInt();
        System.out.println("la suma de los tres es: "+(primero+segundo+tercero));
    }
}

```

```

tres numeros:
1 2 3
la suma de los tres es: 6

```

Por lo tanto `nextInt()`  $\Leftrightarrow$  `next()+Integer.parseInt()`

Lógicamente, si uso `nextInt()` se asume que el String contiene un entero de no ser así se produce una excepción en tiempo de ejecución.

El siguiente ejemplo espera que se introduzca por teclado un string convertible a un entero como "3" o "9999", pero "hola que tal" no es convertible a int.

```


import java.util.Scanner;
class Unidad2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduce un entero: ");
        int miEntero= sc.nextInt();
    }
}

```

```

Introduce un entero: hola que tal
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:943)

```



De la misma manera existe `nextDouble()`, `nextFloat()`, `nextBoolean()` y otros que además de leer un token como un `String` a continuación convierten el `String` en otro tipo de dato.

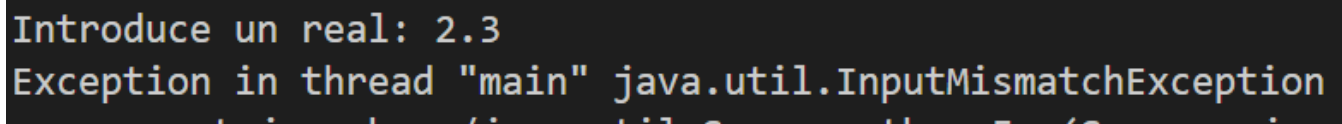
### **¿Coma o punto decimal, para introducir reales por teclado?**

Podemos leer reales cómodamente con `nextFloat()` y `nextDouble()`.

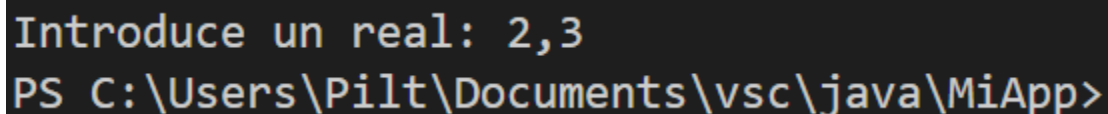
Los ingleses utilizan para escribir números el punto decimal y la coma de millar. Los europeos hacemos al revés, usamos coma decimal y puntos de millar. Parece que la batalla la están ganando los ingleses aunque de momento conviven los dos sistemas.

```
import java.util.Scanner;
class Unidad2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduce un real: ");
        double miReal= sc.nextDouble();
    }
}
```

Al introducir números con decimales por teclado, debemos tener en cuenta que dependiendo de cómo esté configurado la localidad e idioma del equipo y del IDE en el que estamos trabajando, a veces debemos introducir el número con punto decimal y a veces con coma decimal. Con el código de arriba, en el equipo que se probó el código es necesario teclear con coma decimal y por lo tanto 2.3 genera excepción pero 2,3 no



```
Introduce un real: 2.3
Exception in thread "main" java.util.InputMismatchException
```



```
Introduce un real: 2,3
PS C:\Users\Pilt\Documents\vsc\java\MiApp>
```

### **Leer de línea en línea con el método `nextLine()`**

Con este método la clase `Scanner` varía un poco su comportamiento general ya que ahora, no devuelve un token, si no que devuelve toda una línea que puede contener uno o muchos tokens, es decir, `nextLine()` recorre caracteres hacia adelante desde el punto de escaneo actual hasta encontrarse un carácter de nueva línea. Aunque se encuentre caracteres delimitadores como espacios en blanco o tabuladores, `nextLine()` no para hasta encontrar un enter. Para `nextLine()` el único separador es `\n`. Los blancos y tabuladores son caracteres normales para `nextLine()`

El ejemplo inicial en el que leíamos tres palabras, ahora lo modificamos para leer realmente una frase:

```
import java.util.Scanner;
class Unidad2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String miString;
        System.out.println("tres palabras: ");
        miString=sc.nextLine();
        System.out.println("palabras: "+miString);
    }
}
```

```
tres palabras:
uno dos y tres
palabras: uno dos y tres
BUILD SUCCESSFUL (total time: 7 seconds)
```

### **Un efecto inesperado al combinar nextLine() con otros métodos next().**

Como indicamos, la clase Scanner trabaja con el texto que le llega de dos maneras, por líneas y por tokens. Cuando ejecutamos nextLine() obtenemos toda la línea, pero cuando ejecutamos next(), nextByte(), nextInt(), etc... obtenemos el token siguiente. Los enter que introduce el usuario y el uso combinado de nextLine() con los otros métodos next produce efectos inesperados.

Ejemplo: el siguiente código tiene un comportamiento "normal"

```
import java.util.Scanner;
class Unidad2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int edad;
        String nombre;
        System.out.println("edad: ");
        edad=sc.nextInt();
        System.out.println("nombre: ");
        nombre=sc.next();
        System.out.println("edad:"+edad+" nombre: "+nombre);
    }
}
```

```
edad:
34
nombre:
Telma
edad:34 nombre: Telma
```

Si ahora sustituimos el next() por un nextLine() vemos que el programa no funciona bien



```

import java.util.Scanner;
class Unidad2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int edad;
        String nombre;
        System.out.println("edad: ");
        edad=sc.nextInt();
        System.out.println("nombre: ");
        nombre=sc.nextLine();
        System.out.println("edad:"+edad+" nombre: "+nombre);
    }
}

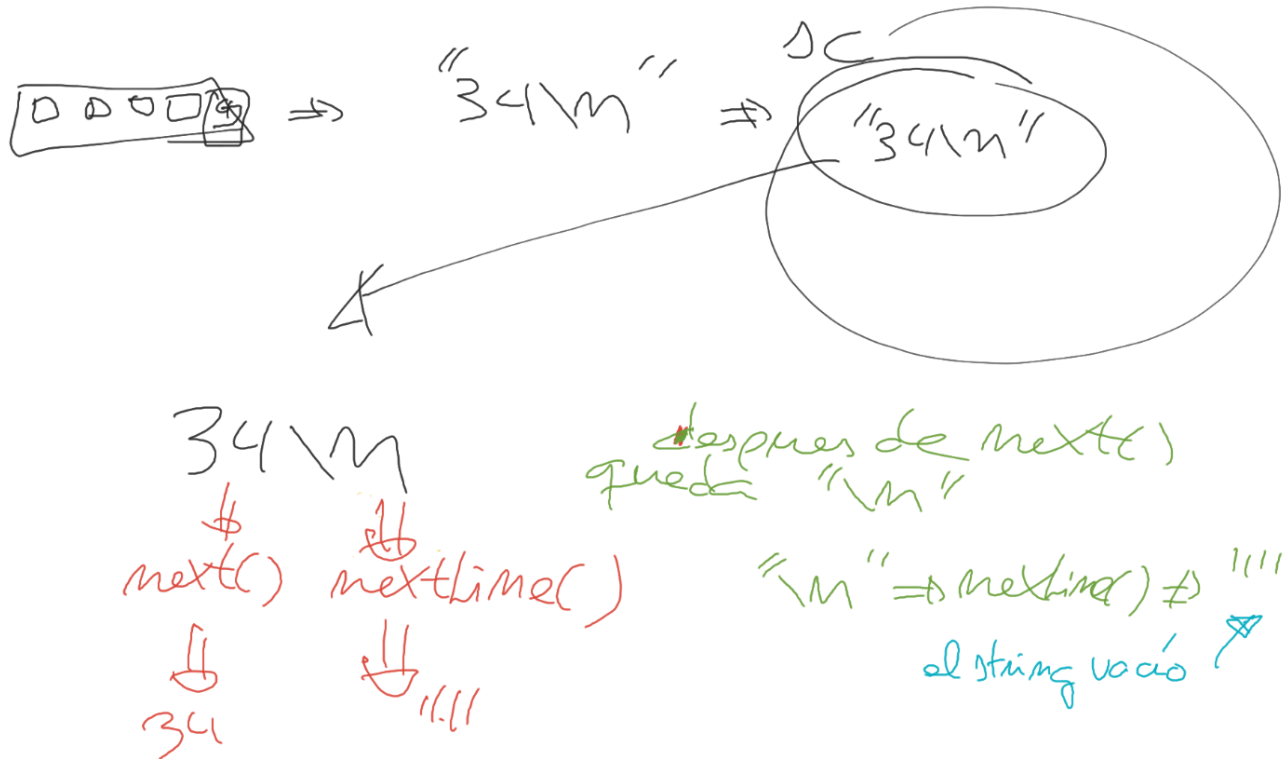
```

```

edad:
34
nombre:
edad:34 nombre:

```

1. tecleo "34\n". Digamos que la variable sc tiene ahora "34\n" y sobre esta cadena los métodos next() y nextLine irán extrayendo bloques de información.
2. nextInt() procesa la cadena anterior. Localiza token "34" ya que después del 4 hay un \n y lo considera carácter separador que le hace parar. Convierte "34" a entero y queda sin analizar la siguiente subcadena "\n".
3. nextLine() busca cualquier cadena que acabe con \n y justamente en el paso anterior nos quedó "\n" qué es un String que acaba en \n como otro cualquiera.
4. nextLine() devuelve al programa la cadena anterior pero sin el \n. De forma que nombre contiene un string vacío "". Por eso al imprimir la variable nombre no vemos nada.



### ¿Es una solución usar siempre next() en lugar de nextLine()?

No es una solución. Por ejemplo supongamos que el caso anterior el nombre se refiere a nombre con apellidos y entonces ya que describo el nombre completo a través de un String que contiene espacios en blanco, no me vale next()

```
import java.util.Scanner;
class Unidad2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int edad;
        String nombre;
        System.out.println("edad: ");
        edad=sc.nextInt();
        System.out.println("nombre: ");
        nombre=sc.next();
        System.out.println("edad:"+edad+" nombre: "+nombre);
    }
}
```

#### Ejecución:

```
edad:
29
nombre:
Pilar Lopez Rodriguez
edad:29 nombre: Pilar
```

### ¿Cómo puedo entonces usar nextLine() sin que se quede atascado con el \n del

## buffer?

haciendo dos `nextLine()`. Con el primero simplemente limpio el buffer de “\n” y con el segundo leo la línea

```
import java.util.Scanner;
class Unidad2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int edad;
        String nombre;
        System.out.println("edad: ");
        edad=sc.nextInt();
        System.out.println("nombre: ");
        sc.nextLine();//limpio enter de buffer
        nombre=sc.nextLine();
        System.out.println("edad:"+edad+" nombre: "+nombre);
    }
}
```

Ejecución, ahora OK

```
edad:
44
nombre:
Pilar Lopez Rodriguez
edad:44 nombre: Pilar Lopez Rodriguez
```

## ¿Por qué no usar siempre `nextLine()`?

Es posible, pero en muchas situaciones es mucho más incómodo e innecesariamente complejo ya que en muchas situaciones necesitaríamos tener que usar un método `split()` que devuelve un array de forma similar a como vimos en Kotlin. Además, nosotros no sabemos por el momento manejar `split()` ni arrays en Java.

## EJERCICIO: U2\_B6\_E1

En la unidad 1 hicimos un ejercicio que indicaba si un número es par o impar

```
class Unidad1{
    public static void main(String[] args){
        int x=8;
        System.out.println(x%2==0?"x es par": x + " es impar");
    }
}
```

SE PIDE: mejóralo introduciendo un número por teclado

## EJERCICIO: U2\_B6\_E2

En la unidad 1 hicimos un ejercicio que calculaba el IMC

```
class Unidad1{
    public static void main(String[] args){
        float peso=71.5f;
        float altura = 1.67f;
        float imc=peso/(altura*altura);
    }
}
```

```

        System.out.println("peso: " + peso);
        System.out.println("altura: " + altura);
        System.out.println("IMC: " + imc);
        System.out.println("\n\nTABLA IMC");
        System.out.println("Delgado: <18.5");
        System.out.println("Normal: entre 18.5 y 24.9");
        System.out.println("Sobrepeso: entre 25 y 29.9");
        System.out.println("Obeso: 30 o más");
    }
}

```

SE PIDE: mejóralo introduciendo los datos por teclado

## ENTRADA POR TECLADO CON LA CLASE CONSOLE

(no la vamos a usar, leer por encima para simplemente percatarse de que hay otros objetos que se pueden usar para leer del teclado).

Respecto a Scanner tiene como ventajas:

- que es más sencilla
- que tiene un método readPassword() para ocultar el texto tecleado.
- Que funciona mejor con la consola windows (acentos y eñes)

Como inconvenientes:

- que sólo vale para trabajar en consola(no funciona en las consolas virtuales de los IDE)
- y que precisamente por ser muy sencilla permite hacer muchas menos cosas que Scanner, de hecho, leer de teclado es sólo una de las cosas que puede hacer Scanner

Funcionamiento básico de Console:

### 1. Importar la clase

```
import java.io.Console;
```

### 2. Crear un objeto console con el método System.console()

Ahora mismo te puede resultar chocante que para crea un objeto Console no se haga

```
new Console()
```

A veces no se puede invocar directamente al constructor de una clase y para conseguir una instancia hay que invocar a métodos intermedios. En este caso System.console() crea un objeto Console y devuelve una referencia al objeto creado

### 3. Leer líneas de teclado con readLine()

### 4. Leer líneas pero con texto enmascarado con readPassword

Hay que observar que readLine() devuelve un String en cambio readPassword() devuelve un char[]. Por el momento no sabemos trabajar con arrays y basta saber que puedo

convertir un char[] en un String en el new de la clase String

Observa en el ejemplo, que nos libramos de los println() para indicar al usuario que teclee

```
import java.io.Console;
```

```
class Unidad2{  
    public static void main(String[] args){  
        Console miconsola=System.console();  
        String nombre= miconsola.readLine("Tu nombre: ");  
        char[] pass =miconsola.readPassword("contraseña: ");  
        String passString= new String(pass);  
        System.out.println("Tecleaste nombre: "+ nombre+ " y contraseña: "+ passString);  
    }  
}
```

### **EJERCICIO: U2\_B6\_E3**

Repite el EJERCICIO: U2\_B6\_E1 pero ahora con Console. Consulta en el API Integer.parseInt() para resolver el ejercicio.

### **EJERCICIO: U2\_B6\_E4**

Repite el EJERCICIO: U2\_B6\_E2 pero ahora con Console. Consulta en el API Integer.parseInt() para resolver el ejercicio.