

3.F. Entrada y salida de datos por consola.

Sitio: [Formación Profesional a Distancia](#)
Curso: Programación
Libro: 3.F. Entrada y salida de datos por consola.
Imprimido por: Iván Jiménez Utiel
Día: jueves, 7 de noviembre de 2019, 15:25

Tabla de contenidos

[1. Programación de la consola: entrada y salida de la información.](#)

[1.1. Conceptos sobre la clase System.](#)

[1.2. Entrada por teclado. Clase System.](#)

[1.3. Entrada por teclado. Clase Scanner.](#)

[1.4. Salida por pantalla.](#)

[1.5. Salida de error.](#)

1. Programación de la consola: entrada y salida de la información.

Los programas a veces necesitan acceder a los recursos del sistema, como por ejemplo los dispositivos de entrada/[salida estándar](#), para recoger datos de teclado o mostrar datos por pantalla.

En Java, la entrada por teclado y la salida de información por pantalla se hace mediante la [clase](#) `System` del paquete `java.lang` de la Biblioteca de Clases de Java.

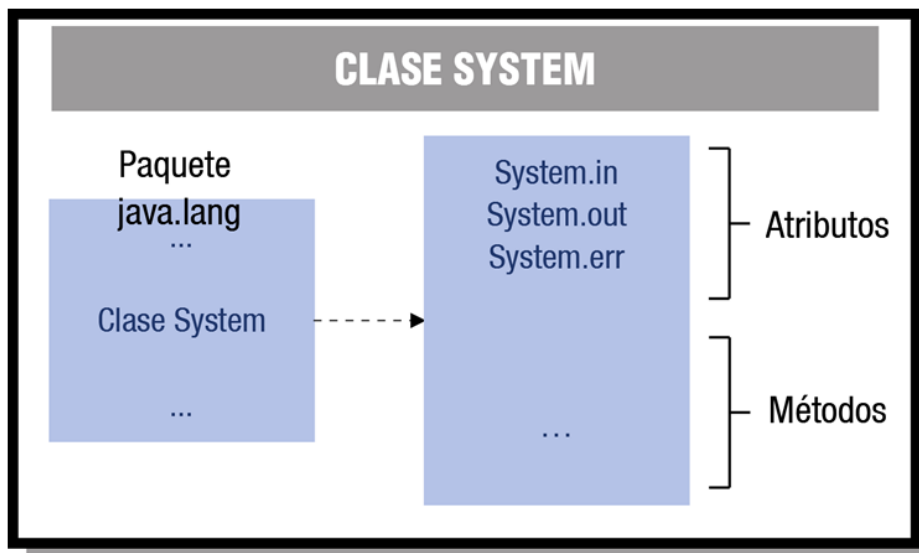


Imagen extraída de curso Programación del MECD.

Como cualquier otra [clase](#), está compuesta de métodos y atributos. Los atributos de la [clase](#) `System` son tres objetos que se utilizan para la entrada y [salida estándar](#). Estos objetos son los siguientes:

- `System.in`. Entrada estándar: teclado.
- `System.out`. [Salida estándar](#): pantalla.
- `System.err`. Salida de error estándar, se produce también por pantalla, pero se implementa como un fichero distinto al anterior para distinguir la salida normal del programa de los mensajes de error. Se utiliza para mostrar mensajes de error.

No se pueden crear objetos a partir de la [clase](#) `System`, sino que se utiliza directamente llamando a cualquiera de sus métodos con el operador de manipulación de objetos, es decir, el operador punto (`.`):

```
System.out.println("Bienvenido a Java");
```

Para saber más

En el siguiente enlace puedes consultar los atributos y métodos de la [clase](#) `System` del paquete `java.lang` perteneciente a la Biblioteca de Clases de Java:

[Clase System de Java.](#)

Autoevaluación

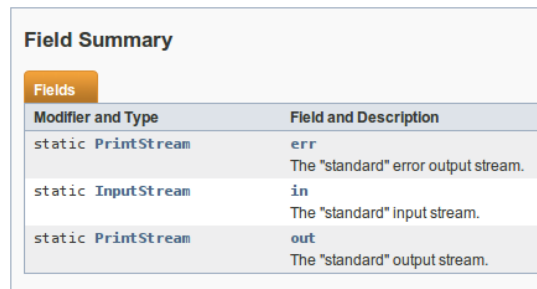
Texto de la pregunta tipo verdadero o falso:

La [clase](#) System del paquete java.io, como cualquier [clase](#), está formada por métodos y atributos, y además es una [clase](#) que no se puede instanciar, sino que se utiliza directamente.

- ☐ Verdadero
- ☐ Falso

1.1. Conceptos sobre la clase System.

La lectura por teclado es muy importante cuando empezamos a hacer nuestros primeros programas. Para entender mejor en qué consiste la [clase System](#), y en particular el [objeto System.in](#) vamos a describirlo más detenidamente.



Modifier and Type	Field and Description
static PrintStream	err The "standard" error output stream.
static InputStream	in The "standard" input stream.
static PrintStream	out The "standard" output stream.

Imagen extraída de curso Programación del MECD.

En el apartado anterior hemos dicho que [System.in](#) es un [atributo](#) de la [clase System](#), que está dentro del paquete [java.lang](#). Pero además, si consultamos la Biblioteca de Clases de Java, nos damos cuenta que es un [objeto](#), y como todos los objetos debe ser instanciado. En efecto, volviendo a consultar la biblioteca de clases nos damos cuenta que [System.in](#) es una [instancia](#) de una [clase](#) de java que se llama [InputStream](#).

En Java, [InputStream](#) nos permite leer en bytes, desde teclado, un archivo o cualquier otro dispositivo de entrada. Con esta [clase](#) podemos utilizar por ejemplo el [método read\(\)](#) que permite leer un byte de la entrada o [skip\(long n\)](#), que salta n bytes de la entrada. Pero lo que realmente nos interesa es poder leer texto o números, no bytes, para hacernos más cómoda la entrada de datos. Para ello se utilizan las clases:

- [InputStreamReader](#). Convierte los bytes leídos en caracteres. Particularmente, nos va a servir para convertir el [objeto System.in](#) en otro tipo de [objeto](#) que nos permita leer caracteres.
- [BufferedReader](#). Lee hasta un fin de línea. Esta es la [clase](#) que nos interesa utilizar, pues tiene un [método readLine\(\)](#) que nos va a permitir leer caracteres hasta el final de línea.

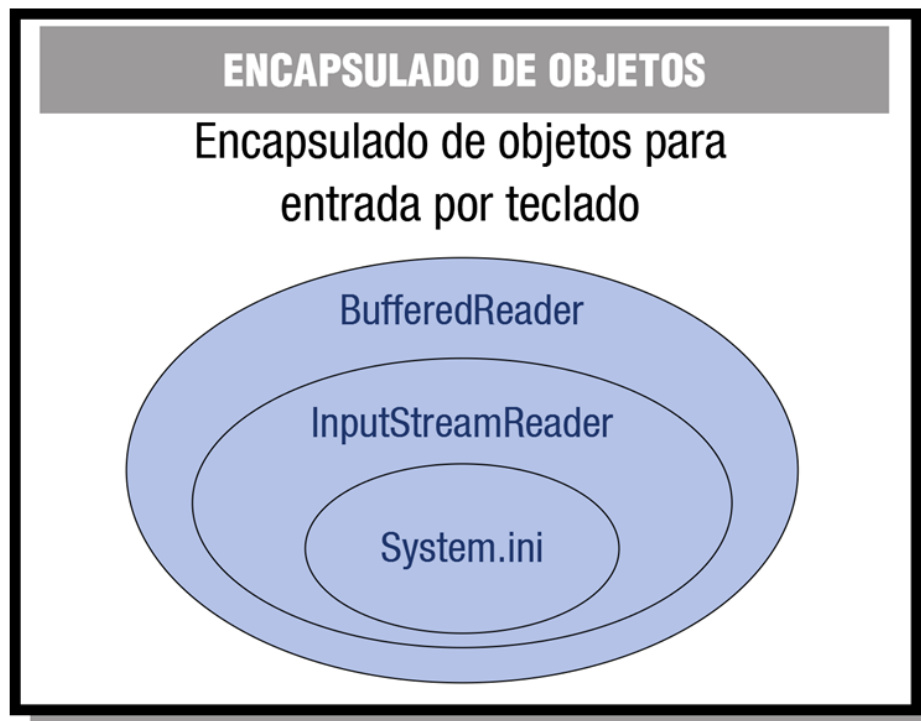


Imagen extraída de curso Programación del MECD.

La forma de instanciar estas clases para usarlas con System.in es la siguiente:

```
InputStreamReader isr = new InputStreamReader(System.in);
```

```
BufferedReader br = new BufferedReader (isr);
```

En el código anterior hemos creado un `InputStreamReader` a partir de `System.in` y pasamos dicho `InputStreamReader` al constructor de `BufferedReader`. El resultado es que las lecturas que hagamos con el [objeto](#) `br` son en realidad realizadas sobre `System.in`, pero con la ventaja de que podemos leer una línea completa. Así, por ejemplo, si escribimos una `A`, con:

```
String cadena = br.readLine();
```

Obtendremos en `cadena` una `"A"`.

Sin embargo, seguimos necesitando hacer la conversión si queremos leer números. Por ejemplo, si escribimos un entero `32`, en `cadena` obtendremos `"32"`. Si recordamos, para convertir cadenas de texto a enteros se utiliza el [método estático](#) `parseInt()` de la [clase](#) `Integer`, con lo cual la lectura la haríamos así:

```
int numero = Integer.parseInt (br.readLine());
```

1.2. Entrada por teclado. Clase System.

A continuación vamos a ver un ejemplo de cómo utilizar la [clase System](#) para la entrada de datos por teclado en Java.



Como ya hemos visto en unidades anteriores, para compilar y ejecutar el ejemplo puedes utilizar las órdenes `javac` y `java`, o bien crear un nuevo proyecto en Eclipse o Netbeans y copiar el código que se proporciona en el archivo anterior.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

/*
 * Ejemplo de entrada por teclado con la clase System
 */

/**
 * @author FMA
 */
public class entradateclado {
    public static void main(String[] args) {
        try
        {
            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(isr);

            System.out.print("Introduce el texto: ");
            String cad = br.readLine();

            //salida por pantalla del texto introducido
            System.out.println(cad);

            System.out.print("Introduce un numero: ");
            int num = Integer.parseInt(br.readLine());

            // salida por pantalla del numero introducido
            System.out.println(num);

        } catch (Exception e) {
            // System.out.println("Error al leer datos");
            e.printStackTrace();
        }
    }
}
```

El mismo código copiable:

```
import java.io.BufferedReader;

import java.io.InputStreamReader;

/*
 * Ejemplo de entrada por teclado con la clase System
 */

/**
 *
 * @author FMA
 */

public class EntradaTecladoSystem {

    public static void main(String[] args) {

        try

        {

            InputStreamReader isr = new InputStreamReader(System.in);

            BufferedReader br = new BufferedReader(isr);

            System.out.print("Introduce el texto: ");

            String cad = br.readLine();

            //salida por pantalla del texto introducido

            System.out.println(cad);

            System.out.print("Introduce un numero: ");

            int num = Integer.parseInt(br.readLine());

            // salida por pantalla del numero introducido

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



```

} catch (Exception e) {
    // System.out.println("Error al leer datos");
    e.printStackTrace();
}
}
}
}
}
```

Observa que hemos metido el código entre excepciones **try-catch**. Cuando en nuestro programa falla algo, por ejemplo la conversión de un **String** a **int**, Java nos avisa lanzando excepciones. Si "capturamos" esa excepción en nuestro programa, podemos avisar al usuario de qué ha pasado. Esto es conveniente porque si no tratamos la **System** excepción seguramente el programa se pare y no siga ejecutándose. El control de excepciones lo veremos en unidades posteriores, ahora sólo nos basta saber que en las llaves del **try** colocamos el código que puede fallar y en las llaves del **catch** el tratamiento de la excepción.

1.3. Entrada por teclado. Clase Scanner.

La entrada por teclado que hemos visto en el apartado anterior tiene el inconveniente de que sólo podemos leer de manera fácil tipos de datos `String`. Si queremos leer otros tipos de datos deberemos convertir la cadena de texto leída en esos tipos de datos.



Imagen extraída de curso Programación del MECD.

El kit de Desarrollo de Java, a partir de su versión 1.5, incorpora la [clase `java.util.Scanner`](#), la cual permite leer tipos de datos `String`, `int`, `long`, etc., a través de la consola de la aplicación. Por ejemplo para leer un tipo de datos entero por teclado sería:

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

```
int i = teclado.nextInt ();
```

O bien esta otra instrucción para leer una línea completa, incluido texto, números o lo que sea:

```
String cadena = teclado.nextLine();
```

En las instrucciones anteriores hemos creado un [objeto](#) de la [clase `Scanner`](#) llamado `teclado` utilizando el constructor de la [clase](#), al cual le hemos pasado como parámetro la entrada básica del sistema `System.in` que por defecto está asociada al teclado.

Para conocer cómo funciona un [objeto](#) de la [clase `Scanner`](#) te proporcionamos el siguiente ejemplo:

```

import java.util.Scanner;
/*
 * Ejemplo de entrada de teclado con la clase Scanner
 */

/**
 *
 * @author FMA
 */
public class EntradaTecladoScanner {

    public static void main(String[] args) {

        // Creamos objeto teclado
        Scanner teclado = new Scanner(System.in);

        // Declaramos variables a utilizar
        String nombre;
        int edad;
        boolean estudias;
        float salario;

        // Entrada de datos
        System.out.println("Nombre: ");
        nombre=teclado.nextLine();
        System.out.println("Edad: ");
        edad=teclado.nextInt();
        System.out.println("Estudias: ");
        estudias=teclado.nextBoolean();
        System.out.println("Salario: ");
        salario=teclado.nextFloat();

        // Salida de datos
        System.out.println("Bienvenido: " + nombre);
        System.out.println("Tienes: " + edad + " años");
        System.out.println("Estudias: " + estudias);
        System.out.println("Tu salario es: " + salario + " euros");
    }
}

```

El mismo código copiable:

import java.util.Scanner;
/*
* Ejemplo de entrada de teclado con la <u>clase</u> Scanner
*/
/**
*
* @author FMA
*/
public class EntradaTecladoScanner {
public static void main(String[] args) {

```
// Creamos objeto teclado

Scanner teclado = new Scanner(System.in);

// Declaramos variables a utilizar

String nombre;

int edad;

boolean estudias;

float salario;

// Entrada de datos

System.out.println("Nombre: ");

nombre=teclado.nextLine();

System.out.println("Edad: ");

edad=teclado.nextInt();

System.out.println("Estudias: ");

estudias=teclado.nextBoolean();

System.out.println("Salario: ");

salario=teclado.nextFloat();

// Salida de datos

System.out.println("Bienvenido: " + nombre);

System.out.println("Tienes: " + edad + " años");

System.out.println("Estudias: " + estudias);

System.out.println("Tu salario es: " + salario + " euros");

}
```

Para saber más

Si quieres conocer algo más sobre la [clase Scanner](#) puedes consultar el siguiente enlace:

[Capturar datos desde teclado con Scanner](#).

1.4. Salida por pantalla.

La salida por pantalla en Java se hace con el [objeto](#) `System.out`. Este [objeto](#) es una [instancia](#) de la [clase](#) `PrintStream` del paquete `java.lang`.



Imagen extraída de curso Programación del MECD.

Si miramos la [API](#) de `PrintStream` obtendremos la variedad de métodos para mostrar datos por pantalla, algunos de estos son:

- `void print(String s)`: Escribe una cadena de texto.
- `void println(String x)`: Escribe una cadena de texto y termina la línea.
- `void printf(String format, Object... args)`: Escribe una cadena de texto utilizando formato.

En la orden `print` y `println`, cuando queramos escribir un mensaje y el valor de una [variable](#) debemos utilizar el operador de concatenación de cadenas (+), por ejemplo:

```
System.out.println("Bienvenido, " + nombre);
```

Escribe el mensaje de `"Bienvenido, Carlos"`, si el valor de la [variable](#) `nombre` es Carlos.

Las órdenes `print` y `println` todas las variables que escriben las consideran como cadenas de texto sin formato, por ejemplo, no sería posible indicar que escriba un número decimal con dos cifras decimales o redondear las cifras, o escribir los puntos de los miles, por ejemplo. Para ello se utiliza la orden `printf()`.

La orden `printf()` utiliza unos códigos de conversión para indicar si el contenido a mostrar de qué tipo es. Estos códigos se caracterizan porque llevan delante el símbolo `%`, algunos de ellos son:

- `%c`: Escribe un carácter.
- `%s`: Escribe una cadena de texto.
- `%d`: Escribe un entero.
- `%f`: Escribe un número en punto flotante.
- `%e`: Escribe un número en punto flotante en notación científica.

Por ejemplo, si queremos escribir el número `float` 12345.1684 con el punto de los miles y sólo dos cifras decimales la orden sería:

```
System.out.printf("% ,.2f\n", 12345.1684);
```

Esta orden mostraría el número `12.345,17` por pantalla.

Estas órdenes pueden utilizar las secuencias de escape que vimos en unidades anteriores, como `"\n"` para

crear un salto de línea, `"\n"` para introducir un salto de tabulación en el texto, etc.

Para saber más

Si quieres conocer algo más sobre la orden `printf()` en el siguiente enlace tienes varios ejemplos de utilización:

[Salida de datos con la orden `printf\(\)`.](#)

1.5. Salida de error.

La salida de error está representada por el [objeto](#) `System.err`. Este [objeto](#) es también una [instancia](#) de la [clase](#) `PrintStream`, por lo que podemos utilizar los mismos métodos vistos anteriormente.



No parece muy útil utilizar `out` y `err` si su destino es la misma pantalla, o al menos en el caso de la consola del sistema donde las dos salidas son representadas con el mismo color y no notamos diferencia alguna. En cambio en la consola de varios entornos integrados de desarrollo como Eclipse o NetBeans la salida de `err` se ve en un color diferente. Teniendo el siguiente código:

```
System.out.println("Salida estándar por pantalla");
System.err.println("Salida de error por pantalla");
```

La salida de este ejemplo en Eclipse es:

```
Salida estándar por pantalla
Salida de error por pantalla
```

La salida de este ejemplo en Netbeans es:

```
run:
Salida estándar por pantalla
Salida de error por pantalla
BUILD SUCCESSFUL (total time: 1 second)
```

Como vemos en un entorno Eclipse o Netbeans, utilizar las dos salidas nos puede ayudar a una mejor depuración del código, pues la diferenciación de colores favorece la identificación más rápida de los mensajes.

Autoevaluación

Relaciona cada [clase](#) con su función, escribiendo el número asociado a la función en el hueco correspondiente.

Clase.

Scanner

PrintStream

InputStreamReader

Relación. Función.

1. Convierte los bytes leídos en caracteres.

2. Lee hasta un fin de línea.

3. Lee diferentes tipos de datos desde la consola de la aplicación.

BufferedReader

4. Contiene varios métodos para mostrar datos por pantalla.

Resolver