



# CAPTURA Y SALIDA DE DATOS EN JAVA (TECLADO Y PANTALLA)

Agosto / 2019

INTRODUCCIÓN A LA PROGRAMACIÓN  
INGENIERÍA EN INFORMÁTICA

# Objetivos

Caracterizar la captura de datos como parte de la construcción de un programa computacional.

Caracterizar la salida de datos como parte de la construcción de un programa computacional.

# Sumario

Captura de datos por teclado con Java.  
Salida de datos por pantalla con Java.

# Bibliografía

Como programar en Java.  
Aprenda Java como si estuviera en  
primero.

# Flujo

Por lo general, las operaciones de entrada y salida de datos se llevan a cabo con flujos, los cuales son secuencias de bytes.

# Flujo

En las operaciones de entrada, los bytes fluyen de un dispositivo (como un teclado, una unidad de disco, una conexión de red) a la memoria principal.

# Flujo

En las operaciones de salida, los bytes fluyen de la memoria principal a un dispositivo (como una pantalla, una impresora, una unidad de disco, una conexión de red).

# Flujo

Cuando empieza la ejecución de un programa, tres flujos se conectan a éste de manera automática.



# Flujo

Por lo común, el flujo de entrada estándar se conecta al teclado, y el flujo de salida estándar se conecta a la pantalla. Un tercer flujo, el flujo de error estándar (*System.err*), se conecta generalmente a la pantalla.

# Clases de Java para lectura y escritura de datos

El package *java.io* contiene las clases necesarias para la comunicación del programa con el exterior. Dentro de este package existen dos familias de jerarquías distintas para la entrada/salida de datos.

```
import java.io.* ;
```

# Captura de datos

En Java, la entrada desde teclado está regulado a través de la clase *System*.

**System.in:** Objeto de la clase *InputStream* preparado para recibir datos desde la entrada estándar del sistema (habitualmente el teclado).

# Captura de datos

Para leer desde teclado se puede utilizar el método **`System.in.read()`** de la clase **`InputStream`**.

# Captura de datos

Este método lee un carácter por cada llamada. Su valor de retorno es un **int**. Si se espera cualquier otro tipo hay que hacer una conversión explícita mediante un **cast**.

```
char c;  
c=(char)System.in.read();
```

# Captura de datos

Para leer datos más largos que un simple carácter es necesario emplear un bucle **while** o **for** y unir los caracteres.

```
char c;  
String frase = new String("");  
while((c=System.in.read()) != '\n')  
    frase = frase + c; // frase.append(c);
```

# Captura de datos

Para facilitar la lectura de teclado se puede conseguir que se lea una línea entera con una sola orden si se utiliza un objeto **BufferedReader**.

# Captura de datos

El método String **readLine()** perteneciente a **BufferedReader** lee todos los caracteres hasta encontrar un `'\n'` o `'\r'` y los devuelve como un String (sin incluir `'\n'` ni `'\r'`).



# Captura de datos

```
InputStreamReader isr = new InputStreamReader(  
    System.in);  
BufferedReader br = new BufferedReader(isr);  
/* o en una linea:  
    BufferedReader br2 = new BufferedReader(new  
        InputStreamReader(System.in));*/  
  
String frase = br2.readLine();  
// Se lee la linea con una llamada
```

# Captura de datos

¿Y qué hacer con una línea entera? La clase **java.util.StringTokenizer** da la posibilidad de separar una cadena de caracteres en las “palabras” (tokens) que la forman. Cuando sea preciso se pueden convertir las “palabras” en números.

# Captura de datos

En el libro **Aprenda Java como si estuviera en primero** página 154 están los métodos más prácticos de la clase **StringTokenizer**.

# Captura de datos

Otra variante para la lectura de datos en Java es utilizando la clase **Scanner**. La clase Scanner está disponible a partir de Java 5 y facilita la lectura de datos en los programas Java.

# Captura de datos

Para utilizar Scanner en el programa tendremos que hacer lo siguiente. Como la clase **Scanner** se encuentra en el paquete `java.util` por lo tanto se debe incluir al inicio del programa la instrucción:

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

# Captura de datos

Para utilizar Scanner en el programa tendremos que hacer lo siguiente. Como la clase **Scanner** se encuentra en el paquete `java.util` por lo tanto se debe incluir al inicio del programa la instrucción:

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

# Captura de datos

Tenemos que crear un objeto de la clase **Scanner** asociado al dispositivo de entrada. Si el dispositivo de entrada es el teclado escribiremos:

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

# Captura de datos

Una vez hecho esto podemos leer datos por teclado. Para leer podemos usar el método `nextXxx()` donde `Xxx` indica en tipo, por ejemplo `nextInt()` para leer un entero, `nextDouble()` para leer un double, etc.



# Salida de datos

En Java, la salida a pantalla está regulado a través de la clase *System*.

**System.out:** Objeto de la clase *PrintStream* que imprimirá los datos en la salida estándar del sistema (normalmente asociado con la pantalla).

# Salida de datos

## System.out

Con la función **print()** imprime en pantalla el argumento que se le pase. Puede recibir cualquier tipo primitivo de variable de Java.

```
System.out.print("Hola, Mundo!");
```

# Salida de datos

## System.out

Con la función **println()** imprime en pantalla el argumento que se le pase. Puede recibir cualquier tipo primitivo de variable de Java. Pero añadiendo '**\n**'(nueva línea) al final.

```
System.out.println("Hola, Mundo!");
```

# Salida de datos

## System.out

Aunque imprime las variables de un modo legible, no permiten dar a la salida un formato ( un valor real imprimir una cantidad determinada de lugares después de la coma) a medida. El programador no puede especificar un formato distinto al disponible por defecto.

# Salida de datos

## System.out

Con **printf** podemos lograr un formato preciso en la salida. El método **printf** cuenta con las siguientes herramientas de formato, cada una de las cuales se verán a continuación:

# Salida de datos

## System.out

La función **printf()** imprime en la unidad de salida (el monitor, por defecto), el texto, y las constantes y variables que se indiquen. La forma general de esta función se puede estudiar viendo su prototipo:

```
System.out.printf("cadena_de_control", arg1, arg2,  
    ..., argn);
```

# Salida de datos

## System.out

La función **printf()** imprime el texto contenido en **cadena\_de\_control** junto con el valor de los otros argumentos, de acuerdo con los formatos incluidos en **cadena\_de\_control**

# Salida de datos

## System.out

Cada formato comienza con el carácter (%) y termina con un carácter de conversión.  
Considérese el ejemplo siguiente:

```
int i=1;
double tiempo=2.4;
float masa=100.56;
System.out.printf("Resultado no: %d. En el instante
    %lf la masa vale %.4f\n",i, tiempo, masa);
```



# Salida de datos

## System.out

Para conocer acerca de este aspecto puede consultar el libro **Cómo programar en Java** *página 1276* disponible en la bibliografía de la asignatura en su espacio virtual.

# Conclusiones

- ◇ Captura de datos
  1. **System.in**
  2. **BufferedReader** + **StringTokenizer**
  3. **Scanner**

# Conclusiones

- ◇ Salida de datos

## **System.out**

- ◇ `print()`
- ◇ `println()`
- ◇ `printf()`

Existen otras variantes pero con lo anterior es suficiente por ahora.

# Estudio independiente

Realizar los siguientes ejercicios publicados en *Athens Online Judge* ([aoj.umcc.cu](http://aoj.umcc.cu)) y enviar sus soluciones al jurado para su calificación:

- ◇  $A+B$
- ◇ Harder  $A+B$

# UNIVERSIDAD DE MATANZAS

cosechando el saber

FIN