

DESARROLLO DE APLICACIONES WEB

Unidad 4

Entrada y Salida de la Información

1r DAW

IES La Mola de Novelda

Departament d'informàtica

ÍNDIX

1.- Lectura de datos des del teclado.....	3
1.1.- Lectura de texto.....	3
1.2.- Lectura de números.....	3
1.3.- La clase Scanner.....	4
1.3.1.- LecturaDadesScanner01.....	4
1.3.2.- LecturaDadesScanner02.....	4
1.3.3.- LecturaDadesScanner03.....	4
2.- Salida por Pantalla.....	6
2.1.- Salida por pantalla.....	6
2.2.- Salida por pantalla formateada.....	6
2.2.1.- EixidaPantallaFormatada01.....	6
2.2.2.- EixidaPantallaFormatada02.....	6
2.2.3.- EixidaPantallaFormatada03.....	6
2.2.4.- Sintaxis del printf.....	6
2.2.5.- Ejemplo completo.....	8
2.2.6.- ImprimirText.....	8
2.3.- La Clase DecimalFormat.....	9
2.3.1.- ExempleDecimalFormat01.....	9

Unidad 4: ENTRADA Y SALIDA DE LA INFORMACIÓN

1.- LECTURA DE DATOS DES DEL TECLADO

1.1.-LECTURA DE TEXTO

Mediante ***System.console().readLine()*** se recoge una línea de texto introducida por teclado.

```
/**
 * Lectura de dades des del teclat
 *
 * @author Joan Carbonell Picó
 */

public class DimeElTeuNom {

    public static void main(String args[]){

        String nom;
        System.out.print("Dime el teu nom: ");
        nom = System.console().readLine();
        System.out.println("Hola "+nom+", encantat de coneixer-te!!");
    }
}
```

```
lliurex@lliurex-ttlportatil:~/Música$ javac DimeElTeuNom.java
lliurex@lliurex-ttlportatil:~/Música$ java DimeElTeuNom
Dime el teu nom: Joan
Hola Joan, encantat de coneixer-te!!
lliurex@lliurex-ttlportatil:~/Música$
```

Siempre hay que especificar el dato que el usuario tiene que introducir. Los datos introducidos por teclado se leen como una cadena de caracteres (***String***).

1.2.-LECTURA DE NÚMEROS

En este caso, la cadena introducida por el usuario se tiene que convertir a un dato numérico por medio del método ***Integer.parseInt()***.

```
/**
 * Lectura de dades des del teclat
 *
 * @author Joan Carbonell Picó
 */

public class LecturaNumeros {

    public static void main(String args[]){

        String l;
        int primerNumero, segonNumero, total;

        System.out.print("Per favor, introdueix un número: ");
        l = System.console().readLine();
        primerNumero = Integer.parseInt(l);

        System.out.print("introdueix un altre, per favor: ");
        l = System.console().readLine();
        segonNumero = Integer.parseInt(l);

        total = primerNumero + segonNumero;

        System.out.println("El primer número introduït és: "+primerNumero);
        System.out.println("El segon número introduït és: "+segonNumero);
        System.out.println("La suma dels dos números és: "+total);
    }
}
```

```
lliurex@lliurex-ttlportatil:~/Música$ javac LecturaNumeros.java
lliurex@lliurex-ttlportatil:~/Música$ java LecturaNumeros
Per favor, introdueix un número: 23
introdueix un altre, per favor: -9
El primer número introduït és: 23
El segon número introduït és: -9
La suma dels dos números és: 14
lliurex@lliurex-ttlportatil:~/Música$
```

1.3.-LA CLASE SCANNER

El método **`System.console().readLine()`** funcionan muy bien en la **consola**, pero puede provocar **problemas** cuando trabajamos con algunos entornos integrados como NetBeans o Eclipse. Para evitarlo, utilizaremos la clase **`Scanner`**. La clase **`Scanner`** funciona tanto en un IDE como en un terminal o consola.

La clase **`Scanner`** provee métodos para leer datos de entrada de varios tipos y se encuentra localizada en el paquete **`java.util`**. Éste es uno de los principales paquetes que vamos a usar ya que incluyen funciones muy utilizadas. Si vamos a la [API de JAVA](#) (Application Programming Interface / Interfaz de Programación de Aplicaciones) podemos ver todas la herramientas que los creadores de Java nos ofrecen para poder crear nuestras aplicaciones. La API está organizada en paquetes los cuales incluyen funciones relacionadas entre sí. En verdad, un paquete contiene un conjunto de clases relacionadas semánticamente. Los valores de entrada pueden venir de varias fuentes, incluyendo valores que se entren por el teclado o datos almacenados en un archivo.

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

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

Se ha creado el objeto teclado asociado al teclado representado por **`System.in`**. Una vez hecho esto podemos leer datos por teclado. Veamos ejemplos de cómo utilizar la clase **`Scanner`** a partir de algunos ejercicios:

1.3.1.- LecturaDadesScanner01

Dado un alumno, queremos obtener el número de clase que se le ha asignado, el nombre del alumno y la nota que ha obtenido en un examen. A continuación aparecerá en pantalla la información introducida por el usuario

1.3.2.- LecturaDadesScanner02

Realiza el mismo ejercicio anterior pero introduciendo los tres datos (número de clase, nombre alumno y nota examen) en una misma línea.

1.3.3.- LecturaDadesScanner03

Calcular la media aritmética de tres números decimales

Principales constructores y métodos de la clase Scanner

public Scanner (InputStream source)	Crea un nuevo Scanner a partir de un flujo de entrada de datos como es el caso de System.in (para poder leer desde teclado).
public String next () public String next (String pattern)	Obtiene el siguiente elemento leído del teclado como un String (si coincide con el patrón especificado). Lanza NoSuchElementException si no quedan más elementos por leer.
public String nextLine ()	Se lee el resto de línea completa, descartando el salto de línea. Devuelve el resultado como un String. Lanza NoSuchElementException si no quedan más elementos por leer.
public int nextInt () public long nextLong () public short nextShort () public byte nextByte () public float nextFloat () public double nextDouble () public boolean nextBoolean ()	Devuelve el siguiente elemento como un int siempre que se trate de un int. Ídem para long, short, byte, float, double y boolean. Lanza InputMismatchException en caso de no poder obtener un valor del tipo apropiado. Lanza NoSuchElementException si no quedan más elementos por leer.
public boolean hasNext ()	Devuelve true si queda algún elemento por leer.
public boolean hasNextLine ()	Devuelve true si queda alguna línea por leer.
public boolean hasNextInt () public boolean hasNextLong () public boolean hasNextShort () public boolean hasNextByte () public boolean hasNextFloat () public boolean hasNextDouble () public boolean hasNextBoolean ()	Devuelve true si el siguiente elemento a obtener se puede interpretar como un int. Ídem para long, short, byte, float, double y boolean.
public Scanner useLocale (Locale l)	Establece la configuración local del Scanner a la configuración especificada por el Locale l.

Explicar la entrada de datos por teclado cuando apenas se ha empezado a aprender a programar usando Java tiene una serie de complicaciones, puesto que para entender exactamente la sintaxis de cada instrucción usada hay que conocer numerosos conceptos que todavía no se han explicado. Además, para acabarlo de complicar, también interviene el aspecto de lenguaje orientado a objetos de Java. Por este motivo, de momento basta con que simplemente aprendáis las instrucciones desde un punto de vista puramente práctico y funcional, pero sin tener que saber exactamente qué está pasando o qué sintaxis concreta se está usando. Siempre que os haga falta, podéis usar como plantilla el código fuente de los ejemplos anteriores.

2.- SALIDA POR PANTALLA

2.1.-SALIDA POR PANTALLA

Utilizada en los ejercicios anteriores: ***System.out.println*** / ***System.out.print***

2.2.-SALIDA POR PANTALLA FORMATEADA

Con ***System.out.printf*** le damos formato a los datos que se van a imprimir en pantalla.

2.2.1.- EixidaPantallaFormatada01

Sigue las instrucciones del profesor y genera el código fuente de las diferentes posibilidades que tenemos en Java para dar formato por salida en la pantalla.

2.2.2.- EixidaPantallaFormatada02

Mostrar el número 1.22 en un ancho de campo de 8 caracteres con dos decimales.

2.2.3.- EixidaPantallaFormatada03

Mostrar la cadena "Total" con un ancho de 15 caracteres. Primero alineado a la izquierda y luego a la derecha. Con el signo menos indicamos alineación a la izquierda. Por defecto es alineación a la derecha.

2.2.4.- Sintaxis del printf



La sintaxis para los especificadores de formato de **printf** es:

%[posición_dato\$][indicador_de_formato][ancho][.precision]carácter_de_conversión

Los elementos entre corchetes son opcionales.

- **posición_dato**: indica la posición del dato sobre el que se va aplicar el formato. El primero por la izquierda ocupa la posición 1.
- **indicador_de_formato**: es el conjunto de caracteres que determina el formato de salida. Los indicadores de formato de **printf** en Java son:

INDICADORES DE FORMATO			
Indicador	Significado	Indicador	Significado
-	Alineación a la izquierda	+	Mostrar signo + en números positivos
(Los números negativos se muestran entre paréntesis	0	Rellenar con ceros
,	Muestra el separador decimal		

- **ancho**: Indica el tamaño mínimo, medido en número de caracteres, que debe ocupar el dato en pantalla.
- **.precision**: Indica el número de decimales que serán representados. Solo aplicable a datos de tipo float o double.
- **carácter_de_conversión**: Carácter que indica cómo tiene que ser formateado el dato. Los más utilizados se muestran en la tabla.

CARACTERES DE CONVERSIÓN			
Carácter	Tipo	Carácter	Tipo
d	Número entero en base decimal	X, x	Número entero en base hexadecimal
f	Número real con punto fijo	s	String
E, e	Número real notación científica	S	String en mayúsculas
g	Número real. Se representará con notación científica si el número es muy grande o muy pequeño	C, c	Carácter Unicode. C: en mayúsculas

2.2.5.- Ejemplo completo

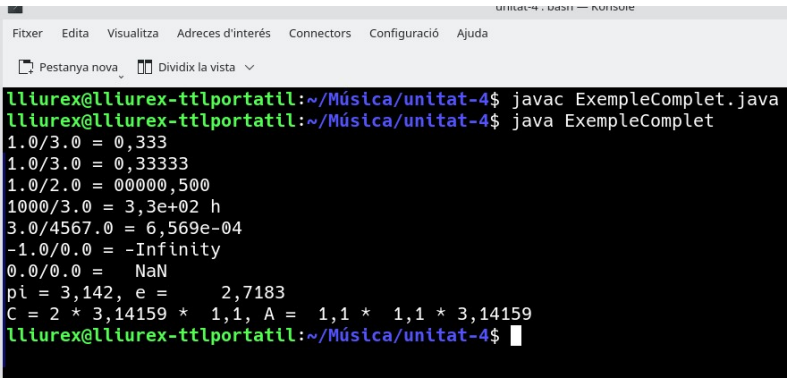
El paquete **java.lang** es un paquete que se carga automáticamente. Por ello podemos hacer uso de todas sus clases, como **String** o **Math**, sin necesidad de importarlo.

Prueba el siguiente ejemplo en el que se le da formato a diferentes tipos de datos. Estudia su resultado:

```
/**
 * Lectura de dades des del teclat
 *
 * @author Joan Carbonell Picó
 */
//import java.lang.Math;
public class ExempleCompleto {

    public static void main(String args[]){
        double q = 1.0/3.0;
        System.out.printf ("1.0/3.0 = %5.3f %n", q);
        System.out.printf ("1.0/3.0 = %7.5f %n", q);
        q = 1.0/2.0;
        System.out.printf ("1.0/2.0 = %09.3f %n", q);
        q = 1000.0/3.0;
        System.out.printf ("1000/3.0 = %7.1e h%n", q);
        q = 3.0/4567.0;
        System.out.printf ("3.0/4567.0 = %7.3e %n", q);
        q = -1.0/0.0;
        System.out.printf ("-1.0/0.0 = %7.2e %n", q);
        q = 0.0/0.0;
        System.out.printf ("0.0/0.0 = %5.2e %n", q);
        System.out.printf ("pi = %5.3f, e = %10.4f %n", Math.PI, Math.E);|
        double r = 1.1;
        System.out.printf("C = 2 * %1$5.5f * %2$4.1f, "+"A = %2$4.1f * %2$4.1f * %1$5.5f %n",Math.PI, r);

    }
}
```



```
lliurex@lliurex-ttlportatil:~/Música/unitat-4$ javac ExempleCompleto.java
lliurex@lliurex-ttlportatil:~/Música/unitat-4$ java ExempleCompleto
1.0/3.0 = 0,333
1.0/3.0 = 0,33333
1.0/2.0 = 00000,500
1000/3.0 = 3,3e+02 h
3.0/4567.0 = 6,569e-04
-1.0/0.0 = -Infinity
0.0/0.0 = NaN
pi = 3,142, e = 2,7183
C = 2 * 3,14159 * 1,1, A = 1,1 * 1,1 * 3,14159
lliurex@lliurex-ttlportatil:~/Música/unitat-4$
```

2.2.6.- ImprimirText

Declarar una variable con el contenido "Mayor". Imprimir en pantalla el contenido de esa variable con el contenido inicial y luego volver a imprimir el contenido de esa variable pero todas las letras en mayúscula.

2.3.-LA CLASE DECIMALFORMAT

La clase **DecimalFormat** de java nos permite mostrar los números en pantalla con el formato que queramos, por ejemplo, con dos decimales, con una coma para separar los decimales, etc.

Las # representan una cifra. Pero si usamos **ceros** en vez de #, los huecos se rellenarán con **ceros por delante**.

2.3.1.- ExempleDecimalFormat01

```
/**
 * Lectura de dades des del teclat
 *
 * @author Joan Carbonell Picó
 */

import java.text.DecimalFormat;

public class ExempleDecimalFormat01 {

    public static void main(String args[]){

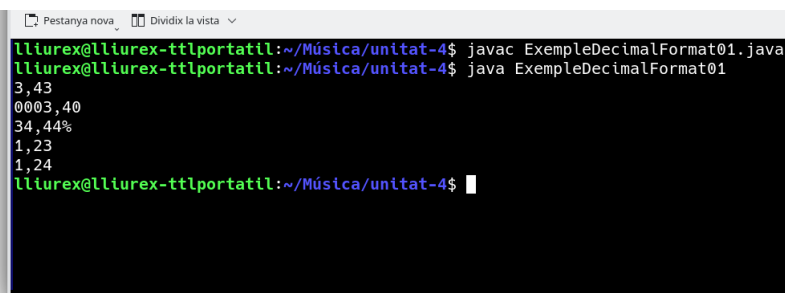
        DecimalFormat f = new DecimalFormat("###.##");
        System.out.println (f.format (3.43242383));

        f = new DecimalFormat("0000.00");
        System.out.println (f.format (3.4));

        f = new DecimalFormat("###.##%");
        System.out.println (f.format(0.3444));

        double a = 1.2345;
        double b = 1.2356;
        f = new DecimalFormat("#.##");
        System.out.println(f.format(a)); // L'eixida és 1,23
        System.out.println(f.format(b)); // L'eixida és 1,24

    }
}
```



```
lliurex@lliurex-ttlportatil:~/Música/unitat-4$ javac ExempleDecimalFormat01.java
lliurex@lliurex-ttlportatil:~/Música/unitat-4$ java ExempleDecimalFormat01
3,43
0003,40
34,44%
1,23
1,24
lliurex@lliurex-ttlportatil:~/Música/unitat-4$
```