

# Entrada/salida (java.io)

# Fundamentos

- La entrada y salida se refiere a operaciones de recuperación de datos desde una fuente externa (entrada) y envío de datos desde el programa al exterior (salida)
- Dos paquetes de clases para realizar esas operaciones:
  - `java.io`. Paquete tradicional, incluido desde las primeras versiones
  - `java.nio`. Nueva entrada y salida. Incorporado en Java 5 y mejorado en sucesivas versiones
- En esta lección se estudia `java.io`

# Salida con java.io

➤ Entre las principales clases para operaciones de salida de datos, están:

- **OutputStream.** Clase abstracta que representa un flujo de salida
- **PrintStream.** Subclase de **OutputStream** que proporciona métodos para enviar datos a cualquier flujo de salida
- **FileOutputStream.** Subclase de **OutputStream** que representa un flujo de salida asociado a un fichero
- **FileWriter.** Clase específica para escritura de texto en un fichero

# Escritura en consola

`System.out.println(dato)`

objeto  
PrintStream

Método

## ➤ Métodos de PrintStream:

- `print(tipo dato)`. Escritura de cualquier tipo de dato Java, sin incluir salto de línea al final
- `println(tipo dato)`. Escritura de cualquier tipo de dato Java, con salto de línea al final
- `printf(String format, Object...args)`. Escritura de datos con formato

# Escritura en un fichero


## ➤ Utilizando PrintStream:

```
String dir="/user/mydata.txt";  
try(PrintStream out=new PrintStream(dir)){  
    out.println("dato1");  
    ...  
}catch(IOException ex){...}
```

- Escritura con formato
- Graba los datos en modo sobrescritura
- Si el fichero no existe se crea

```
String dir="/user/mydata.txt";  
try(FileOutputStream fos=new FileOutputStream(dir, true);  
    PrintStream out=new PrintStream(fos)){  
    out.println("dato1");  
    ...  
}catch(IOException ex){...}
```

Permite realizar la escritura en modo append



# Escritura en un fichero

## ➤ Utilizando FileWriter:

```
String dir="/user/mydata.txt";  
try(FileWriter out=new FileWriter(dir)){  
    out.write("dato1");  
    ...  
}catch(IOException ex){...}
```

- Graba los datos en modo sobrescritura
- Si el fichero no existe se crea

```
String dir="/user/mydata.txt";  
try(FileWriter out=new FileWriter(dir, true)){  
    out.write("dato1");  
    ...  
}catch(IOException ex){...}
```

- Graba los datos en modo append
- Si el fichero no existe se crea

```
String dir="/user/mydata.txt";  
try(FileWriter out=new FileWriter(dir, true);  
    BufferedWriter bw=new BufferedWriter(out)){  
    bw.write("dato1");  
    bw.newLine();  
    ...  
}catch(IOException ex){...}
```

- Escritura de datos a través de un `BufferedWriter` que mejora el rendimiento

# Revisión conceptos



**Dado el siguiente código, indica la respuesta correcta**

```
var out = new FileWriter("text.txt"); //1  
out.write("hello"); //2  
out.close(); //3
```

- A. Si el fichero no existe, se producirá una excepción en la línea 1
- B. Si el fichero no existe, se producirá una excepción en la línea 2
- C. El texto "hello" reemplazará al contenido existente

**Respuesta**

- A. Incorrecta. En la escritura si el fichero no existe se crea
- B. Incorrecta.
- C. Correcta. Por defecto se abre en modo sobrescritura

# Entrada con java.io

➤ Entre las principales clases para operaciones de entrada de datos, están:

- **InputStream.** Clase abstracta que representa un flujo de entrada de bytes
- **FileInputStream.** Subclase de **InputStream** que representa un flujo de entrada asociado a un fichero
- **FileReader.** Clase específica para lectura de texto en un fichero
- **BufferedReader.** Proporciona un mecanismo eficiente para la lectura de cadenas de texto de una fuente externa



# Lectura por teclado

## ➤ Lectura utilizando BufferedReader:

```
try(InputStreamReader reader=new  
    InputStreamReader(System.in);  
    BufferedReader br=new BufferedReader(reader)){  
    System.out.println("Introduce tu nombre: ");  
    String nombre=br.readLine();  
}catch(IOException ex){..}
```

- Se debe crear un `InputStreamReader` asociado a la entrada estándar (`System.in`)

## ➤ Lectura mediante Scanner:

```
Scanner sc=new Scanner(System.in);  
System.out.println("Introduce tu nombre: ");  
String nombre=sc.nextLine();
```

- `Scanner` se encuentra en `java.util`
- Dispone de otros métodos para leer datos como tipos primitivos (`nextInt`, `nextDouble`, ..)

# Lectura de un fichero

## ➤ Lectura de texto utilizando BufferedReader:

```
String dir="/user/mydata.txt";
try(FileReader fr=new FileReader(dir);
BufferedReader br=new BufferedReader(fr)){
    String line;
    while((line=br.readLine())!=null){
        System.out.println(line);
    }
}catch(IOException ex){...}
```

- Lectura de todas las líneas del fichero
- Si el fichero no existe se produce una excepción

## ➤ Lectura de bytes mediante FileInputStream:

```
String dir="/user/mydata.txt";
File file=new File(dir);
try(FileInputStream fis=new FileInputStream(file)){
    byte[] res=new byte[file.length()];
    fis.read(res);
}catch(IOException ex){...}
```

- Utilizado para lectura de ficheros binarios

# Revisión conceptos



Tenemos un fichero "temperaturas.txt", que en cada línea contiene el registro de temperatura de una ciudad. Escribe un bloque de código que calcule la temperatura media registrada

**Respuesta**

```
String dir="temperaturas.txt";
try(FileReader fr=new FileReader(dir);
BufferedReader br=new BufferedReader(fr)){
    int contador=0;
    double media=0.0;
    while((line=br.readLine())!=null){
        media+=Double.parseDouble(linea);
        contador++;
    }
    System.out.println(media/contador);
}catch(IOException ex){...}
```

# La clase File

## ➤ Representa una ruta a un fichero o directorio

```
File file=new File("/user/mydata.txt");
```

## ➤ Proporciona métodos para obtener información sobre el elemento:

- `boolean exists()`. Devuelve true si existe
- `boolean isFile()`. Devuelve true si es un fichero
- `boolean isDirectory()`. Devuelve true si es un directorio
- `boolean delete()`. Elimina el elemento. Devuelve true si ha conseguido eliminarlo