

# Entrada/salida (java.io)

# Fundamentos

- La entrada y salida se refiere a operaciones de recuperación de datos desde una fuente externa (lectura) 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 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

# 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 salida 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 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

# 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