

UNIDAD IO Y II:

FICHEROS

CLASE I: EXCEPCIONES Y FICHEROS

FLUJOS O STREAMS

**Comunicación entre dos fuentes
de información**

- Un fichero
- La memoria
- La red
- Otro programa

**SIEMPRE QUE SE VAYA A REALIZAR
UN INTERCAMBIO DE INFORMACIÓN,
TENEMOS QUE ABRIR UN STREAM**

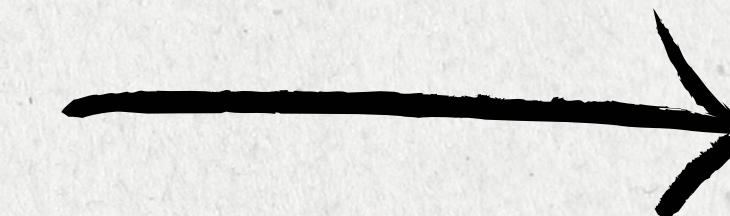
FLUJOS DE DATOS

- Flujos pueden ser de entrada o salida
 - Carácter: Asociados a fuentes de texto
 - Binarios: Trasmiten bytes (0 a 255) aunque permite transmitir cualquier tipo de dato.

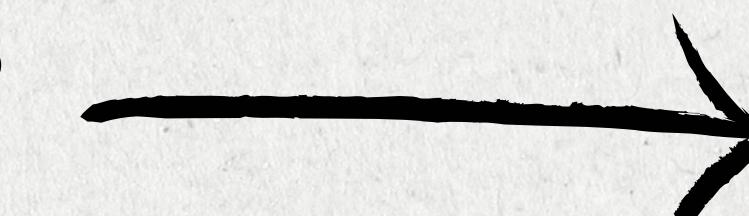
FLUJOS O STREAMS

TIPOS DE FLUJOS

- Flujo de bytes
(8 bits)
- Flujo de
caracteres
(16 bits)



InputStream y
OutputStream



Reader y Writer

EXCEPCIONES

Se pueden producir errores en la trasmisión de información.

- Acceso a ficheros inexistentes
- Nombres mal escritos
- Fin de fichero alcanzado

Son situaciones anormales que interrumpen el flujo normal de ejecución.

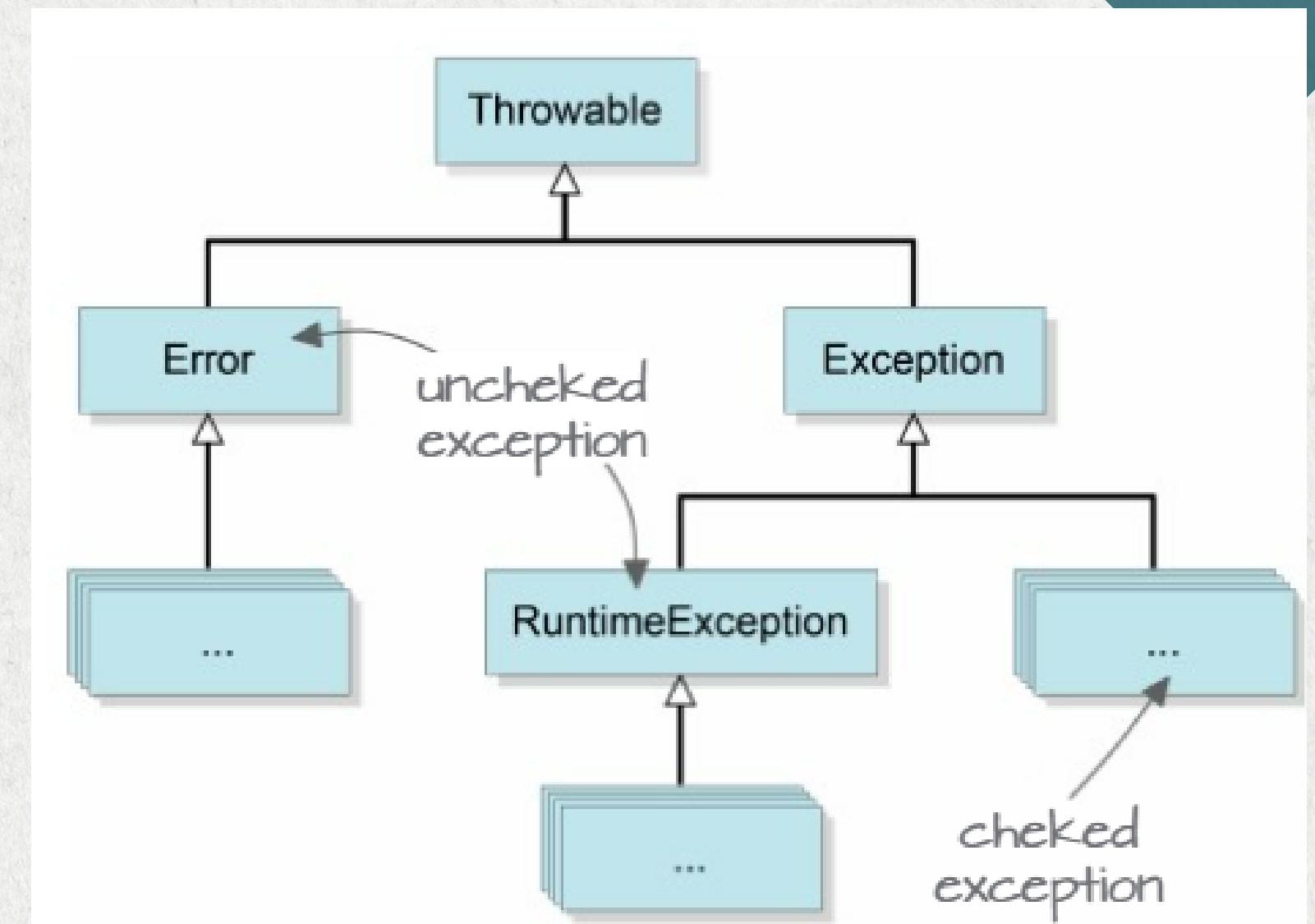
EXCEPCIONES

Tras cada error se lanza un objeto de la clase Throwable que contiene:

- Información del error
- Su causa
- Contexto en el que sucede

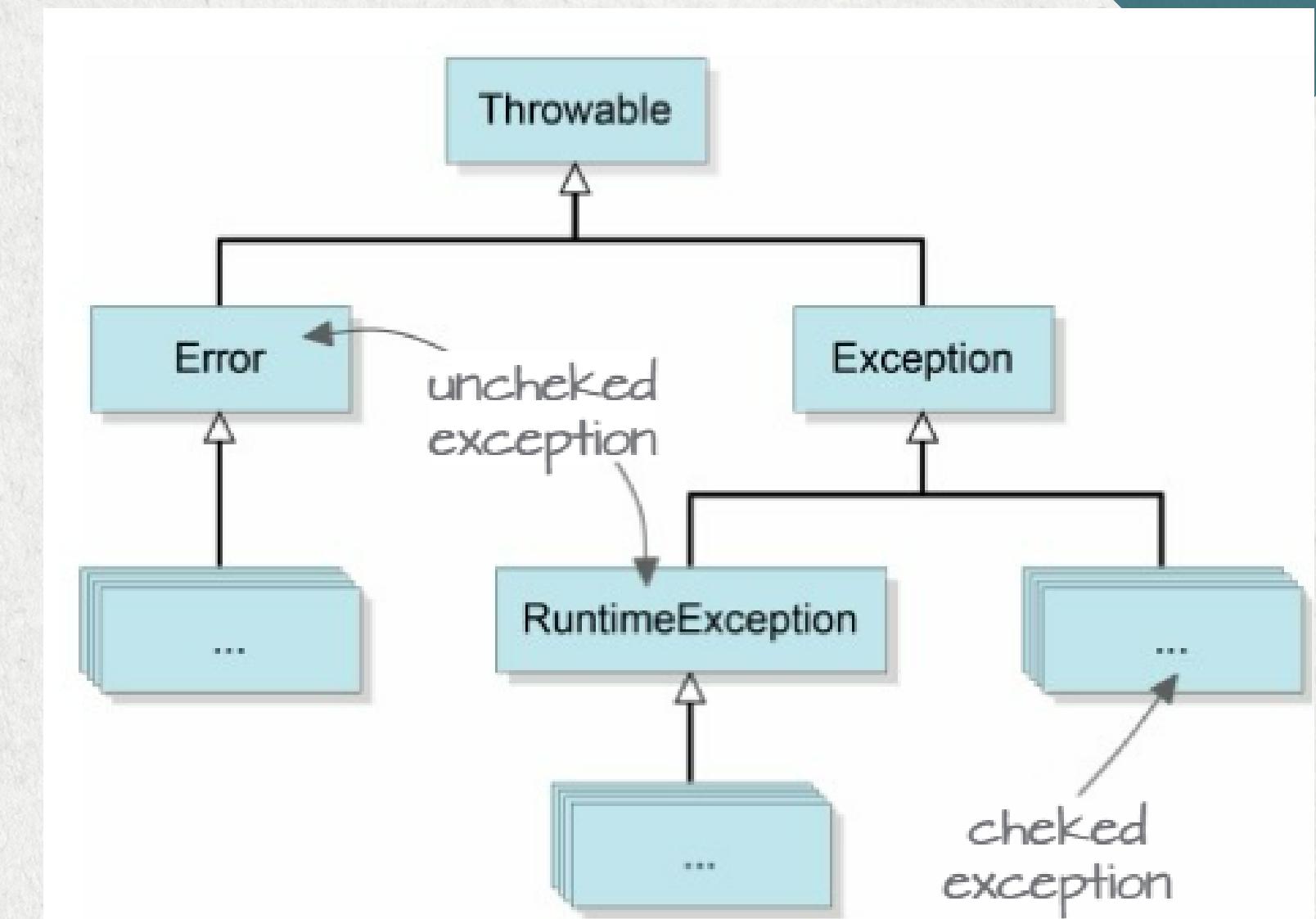
EXCEPCIONES

Los errores no vinculados con nuestro código serán de tipo Error (por ejemplo, hardware)



EXCEPCIONES

RuntimeExceptions son lanzadas cuando parte de nuestro código es erróneo



EXCEPCIONES

Las excepciones pueden ser:

- Capturados para que continúe la ejecución (try / catch / finally)
- Enviados a la máquina virtual y mostrados (throw)

EXCEPCIONES

- El bloque TRY almacena un conjunto de líneas de código que pueden producir errores.

```
try {  
    int a = 2;  
    int b = 0;  
    int c = a/b;  
}
```

EXCEPCIONES

- El bloque CATCH captura la excepción y en consecuencia actuará.

```
catch(Exception e) {  
    System.out.println(e.getMessage());  
    System.out.println(e);  
}
```

```
/ by zero  
java.lang.ArithmetricException: / by zero
```

EXCEPCIONES

- Dentro de los paréntesis podemos poner cualquier excepción y capturará la misma y sus hijas.

```
catch(ArithmetricException e) {  
    System.out.println(e);  
}  
catch(IOException e) {  
    System.out.println(e);  
}  
catch(Exception e) {  
    System.out.println(e);  
}
```

EXCEPCIONES

- Podemos indicar más de una en una misma línea

```
catch(ArithmetricException | IOException e) {  
    System.out.println(e);  
}
```

EXCEPCIONES

- Podemos indicar que podrá haber excepción, pero dejar que otros métodos manejen.

```
public static void main(String[] args){  
    try {  
        metodo(2,0);  
    }catch(Exception e) {  
        System.out.println(e);  
    }  
  
}  
public static void metodo(int a, int b) throws ArithmeticException {  
    int c = a/b;  
}
```

EXCEPCIONES

- El bloque FINALLY permite continuar con la ejecución una vez se ha solucionado la excepción.

```
}catch(Exception e) {  
    System.out.println(e);  
}finally {  
    System.out.println("Haremos otras cosas después");  
}
```

```
java.lang.ArithmetricException: / by zero  
Haremos otras cosas después
```

EXCEPCIONES

- Algunas excepciones son “predicibles” por Java y nos obligan a establecer un bloque try/catch o throws.

EXCEPCIONES

Excepciones comprobadas

- IOException
- FileNotFoundException
- NumberFormatException
- ClassCastException

Excepciones no comprobadas

- ArithmeticException
- ArrayIndexOutOfBoundsException

EXCEPCIONES PROPIAS

- Podemos implementar nuestras excepciones heredando de cualquier excepción.

```
public class ExcepcionNumeroNegativo extends Exception{  
    public String toString() {  
        return "Número invalido";  
    }  
}
```

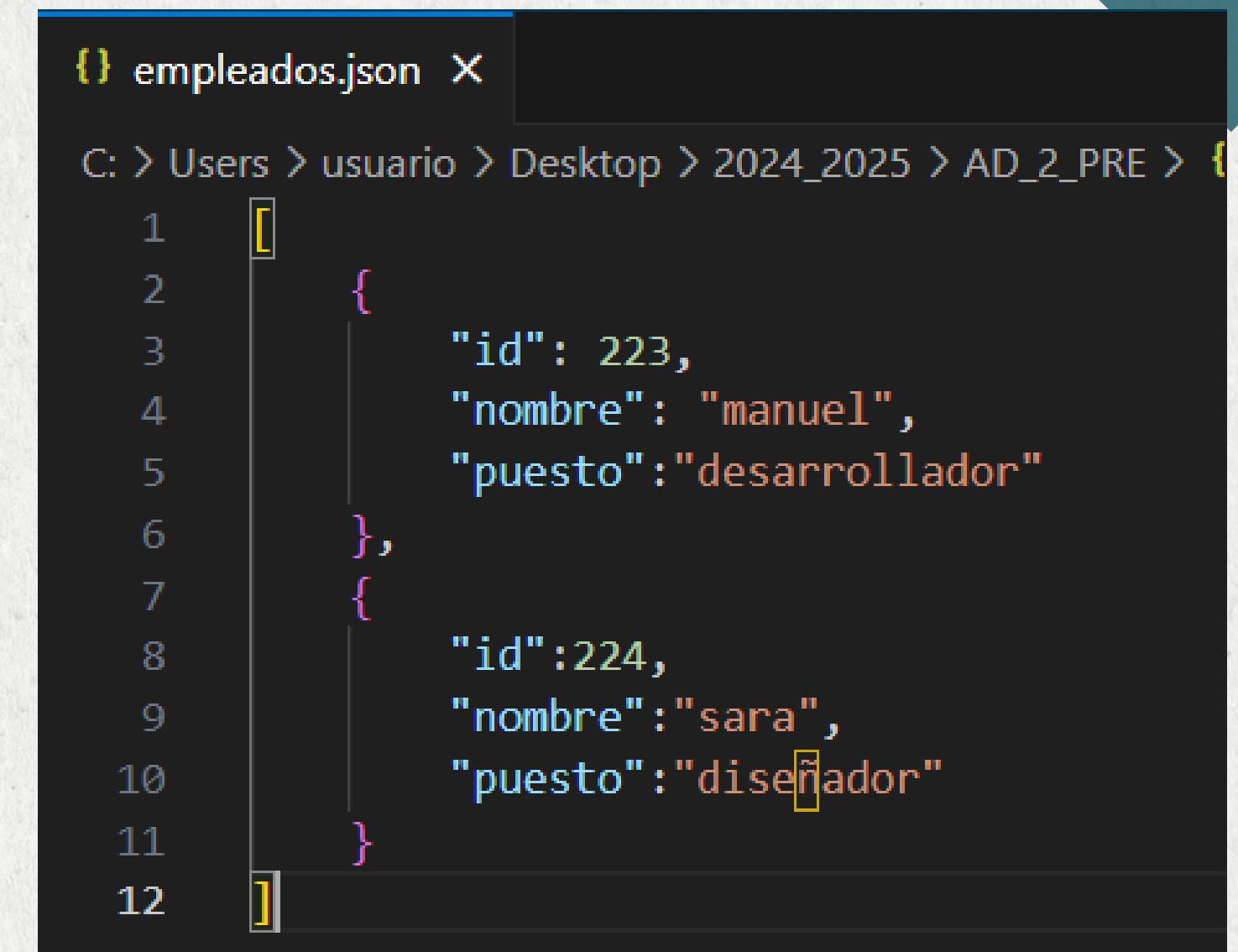
EXCEPCIONES PROPIAS

- Podemos implementar nuestras excepciones heredando de cualquier excepción.

```
try {  
    int numero = -17;  
    if(numero < 0) {  
        throw new ExpcionNumeroNegativo();  
  
    }  
}catch(ExpcionNumeroNegativo e) {  
    System.out.println(e);  
}
```

FICHEROS

- Permiten almacenar datos permanentes (no son volátiles)
- Generalmente se utilizarán para almacenar registros



```
{} empleados.json X  
C: > Users > usuario > Desktop > 2024_2025 > AD_2_PRE > [  
1   [  
2     {  
3       "id": 223,  
4       "nombre": "manuel",  
5       "puesto": "desarrollador"  
6     },  
7     {  
8       "id": 224,  
9       "nombre": "sara",  
10      "puesto": "diseñador"  
11    }  
12  ]
```

JAVA.IO Y CLASE FILE

El paquete java.io maneja la entrada y salida de información

Concretamente, la clase File permite obtener información sobre los ficheros (dirección, nombre, atributos...)

Constructores:

File (String directorioyfichero)

`new File("C:\\\\directorio\\\\fichero.txt");`

`new File("/directorio/fichero.txt");`

CLASE FILE

Método	Función
<code>String[] list()</code>	Devuelve un array de String con los nombres de ficheros y directorios asociados al objeto File
<code>File[] listFiles()</code>	Devuelve un array de objetos File conteniendo los ficheros que estén dentro del directorio representado por el objeto File
<code>String getName()</code>	Devuelve el nombre del fichero o directorio
<code>String getPath()</code>	Devuelve el camino relativo
<code>String getAbsolutePath()</code>	Devuelve el camino absoluto del fichero/directorio
<code>boolean exists()</code>	Devuelve <i>true</i> si el fichero/directorio existe
<code>boolean canWrite()</code>	Devuelve <i>true</i> si el fichero se puede escribir
<code>boolean canRead()</code>	Devuelve <i>true</i> si el fichero se puede leer
<code>boolean isFile()</code>	Devuelve <i>true</i> si el objeto File corresponde a un fichero normal
<code>boolean isDirectory()</code>	Devuelve <i>true</i> si el objeto File corresponde a un directorio
<code>long length()</code>	Devuelve el tamaño del fichero en bytes
<code>boolean mkdir()</code>	Crea un directorio con el nombre indicado en la creación del objeto File . Solo se creará si no existe
<code>boolean renameTo(File nuevonombre);</code>	Renombra el fichero representado por el objeto File asignándole <i>nuevonombre</i>
<code>boolean delete()</code>	Borra el fichero o directorio asociado al objeto File
<code>boolean createNewFile()</code>	Crea un nuevo fichero, vacío, asociado a File si y solo si no existe un fichero con dicho nombre
<code>String getParent()</code>	Devuelve el nombre del directorio padre, o <i>null</i> si no existe

EJERCICIOS

Ejercicio 1:

Crea una excepción que maneje la obtención de números negativos y un método con un bloque try/catch que permita la lectura de enteros positivos por teclado utilizando esa excepción.

EJERCICIOS

Ejercicio 2:
Listar todos los
ficheros de la
carpeta
proporcionada y
mostrar la información
de (cada) uno de ellos

EJERCICIOS

Ejercicio 3:
Listar todos los
ficheros de una
carpeta
proporcionada
por el usuario por
teclado y que elija de que
documento quiere ver la
información