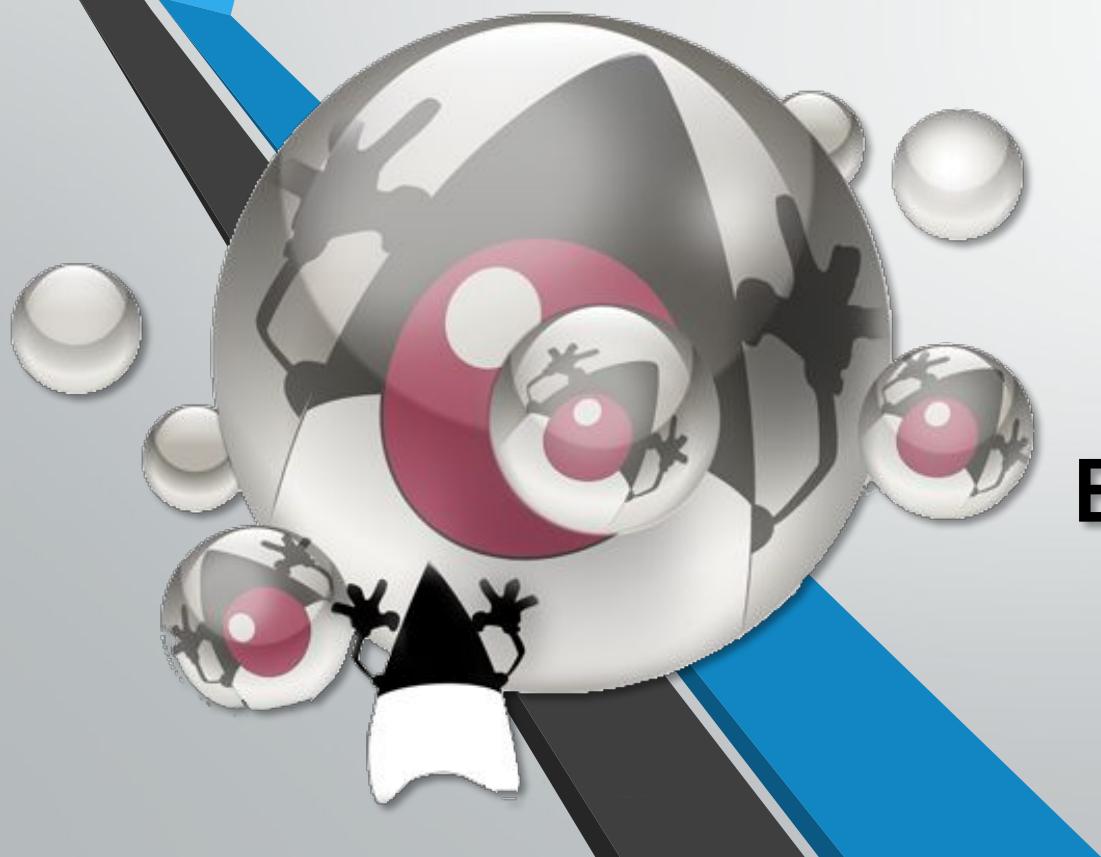


JAVA



TEMA 12: **ENTRADA/SALIDA EN JAVA**

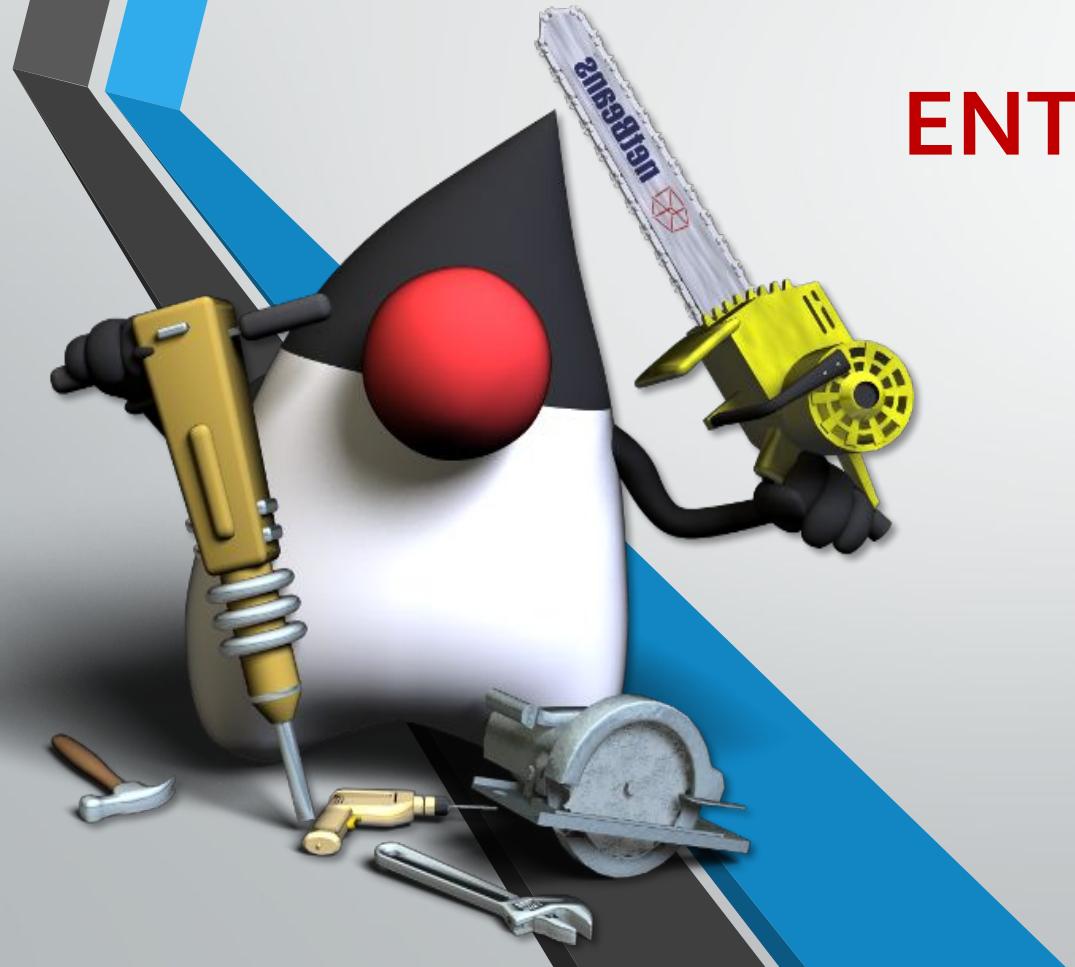
ÍNDICE

- 1.** Excepciones.
- 2.** La clase File.
- 3.** Entrada / Salida estándar.
- 4.** Entrada / Salida de Ficheros de Texto.
- 5.** Entrada / Salida de Ficheros Binarios.
- 6.** Serialización.
- 7.** Ficheros de acceso aleatorio.
- 8.** Ejercicios de consolidación



JAVA

ENTRADA Y SALIDA EN JAVA



1. Excepciones

1.- Excepciones

- En el tema 3 vimos que las excepciones eran un mecanismo de control y gestión de errores de ejecución.
- Las posibilidades de gestión de excepciones son 2:
 - Gestión en el propio método
 - try – catch
 - Paso de la excepción a otro método
 - throws

1.- Excepciones

```
try {  
    // Bloque de código peligroso  
} catch (TipoExpcion1 objetoquecapturaEx1) {  
    //Tratamiento excepciones TipoExpcion1  
} catch (TipoExpcion2 objetoquecapturaEx2) {  
    //Tratamiento excepciones TipoExpcion2  
} finally {  
    //Sentencias comunes (haya o no excepción)  
    //Se ejecutan SIEMPRE  
    //Usos típicos: cierre de ficheros, conexiones a BD's, etc  
}
```

1.- Excepciones

```
package excepciones;
/**
 * @author Oscar Laguna Garcia
 */
public class Excepciones {

    public static int calculoAleatorio() {
        int aleatorio;
        aleatorio = (int) (Math.round(Math.random()*9));
        return aleatorio;
    }

    public static void mostrarPosicion(String texto[]) {
        boolean indiceNoValido=true;
        int i; //Entero que tomará numeros aleatorios de 0 a 9
        while(indiceNoValido == true){
            try {
                i=calculoAleatorio(); //Aleatorio del 0 al 9
                System.out.println(texto[i]);
                indiceNoValido=false;
            } catch(ArrayIndexOutOfBoundsException exc) {
                System.out.println("Fallo en el indice");
            }
        }
    }

    public static void main(String[] args) {
        String texto[]={ "Uno", "Dos", "Tres", "Cuatro", "Cinco"}; //son 5 elementos
        mostrarPosicion(texto);
    }
}
```



1.- Excepciones

- En Java hay muchos tipos de excepciones.
- El paquete `java.lang.Exception` y sus subpaquetes contienen todos los tipos de excepciones.
- Cuando se produce una excepción, se genera un objeto asociado a la misma. Este objeto es de clase `Exception` o de alguna de sus herederas.
- Hay una clase, la `java.lang.Error` y sus subclases que sirven para definir los errores irrecuperables.

1.- Excepciones

throws

- Hasta ahora, cuando un método producía una excepción, nos hemos encargado de que el propio método quien se encargue de manejar sus excepciones.
- Otra posibilidad es hacer que la excepción la maneje el código que hizo la llamada. Esto se hace añadiendo la palabra **throws** tras la primera línea de un método.
- Tras **throws** se indica que excepciones puede provocar el código del método. Si ocurre una excepción en el método, el código abandona ese método y regresa al código desde el que se llamó al método.

1.- Excepciones

- Ejemplo:

```
void usarArchivo (String archivo) throws IOException , InterruptedException {  
    ...  
}
```

- En este caso se está indicando que el método `usarArchivo` puede provocar excepciones del tipo `IOException` y `InterruptedException`.
- Esto obliga a que el código que invoque a este método deba preparar el (o los) `catch` correspondientes:

1.- Excepciones

```
try{  
    ...  
    objeto.usarArchivo("C:\texto.txt"); //puede haber excepción  
    ...  
}  
catch(IOException ioe){ //Si ocurre un error en la apertura  
    ...  
}  
catch(InterruptedException ie){ //Si hay un error en el contenido del archivo  
    ...  
}
```

1.- Excepciones

```
package excepciones;
/**
 * @author OLG
 */
public class Excepciones {

    public static int calculoAleatorio(){
        int aleatorio;
        double doubleAleatorio=Math.random()*9;
        doubleAleatorio=Math.floor(doubleAleatorio);
        aleatorio=(int)doubleAleatorio;
        return aleatorio;
    }

    public static void mostrarPosicion (String[] array) throws ArrayIndexOutOfBoundsException {
        int i;
        i=calculoAleatorio();
        System.out.println(array[i]);
    }

    public static void main(String[] args) {
        String [] array={"Cero","Uno","Dos","Tres","Cuatro"}; //Son 5 elementos
        boolean generarOtroAleatorio=true;
        int i; //Entero que tomará números aleatorios entre 0 y 9
        while(generarOtroAleatorio==true){
            try{
                mostrarPosicion(array);
                generarOtroAleatorio=false; //Para que termine el programa
            }
            catch(ArrayIndexOutOfBoundsException e){
                System.out.println("Fallo al generar el indice");
            }
        }
    }
}
```

EJERCICIOS

- **Ejercicio 01.- (OPTATIVO)** Ayudándote de la cláusula **throws**, realiza el siguiente programa con listas y capturando todas las excepciones que te sean posibles:
- Implementa una aplicación en Java que escriba una agenda telefónica con los siguientes atributos para cada contacto:
 - Nombre
 - Edad
 - Número de móvil
- Además, crearás un menú que le permita al usuario las siguientes opciones:
 1. Añadir contactos a la agenda: Pediremos los datos de un contacto al usuario y lo introducimos en la lista.
 2. Visualizar la lista de contactos.
 3. Eliminar contactos de la lista: Pediremos el número de teléfono y eliminamos el contacto.
 4. Mostrar todos los contactos ordenados por su edad.
 5. Salir del programa.

Hasta que el usuario no pulse 5 no saldremos del programa y se volverá a mostrar el menú.

JAVA

ENTRADA / SALIDA EN JAVA



2. La clase File

2.- La clase File

- Antes de proceder al estudio de las clases que describen la entrada/salida vamos a estudiar la clase File.
- Esta clase nos proporciona información acerca de los archivos, de sus atributos, de los directorios, etc. También nos permite, entre otras cosas:
 - Crear y eliminar archivos
 - Crear y eliminar directorios.

2.- La clase File

- La clase nos proporciona los siguientes constructores para crear objetos File:

- ***File(String directorioyfichero):***

en Linux: new File("/directorio/fichero.txt");

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

- ***File(String directorio, String nombrefichero):***

new File("directorio", "fichero.txt");

- ***File(File directorio, String fichero)***

new File(new File("directorio"), "fichero.txt")

2.- La clase File

- Ejemplos:

- File fichero = new File("personas.dat"); //Crea el objeto fichero de la clase File asociado al fichero personas.dat que se encuentre en el directorio del proyecto
- File fichero = **new** File ("C:\\UT1\\ejercicios\\ejemplo1.txt");//Windows
- File fichero = **new** File ("/home/UT1/ejercicios/ejemplo1.txt");//Linux
- String directorio= "C:/UT1/ejercicios";
- File fichero2 = **new** File(directorio,"ejemplo2.txt");
- File direc = **new** File(directorio); //Creamos el directorio

File fichero3 = **new** File(direc,"ejemplo3.txt"); //Luego creamos el archivo

2.- La clase File

- La clase File también nos ofrece varios métodos. Veamos los más utilizados:

MÉTODO	DESCRIPCIÓN
boolean canRead()	Devuelve true si se puede leer el fichero
boolean canWrite()	Devuelve true si se puede escribir en el fichero
boolean createNewFile()	Crea el fichero asociado al objeto File. Devuelve true si se ha podido crear. Para poder crearlo el fichero no debe existir. Lanza una excepción del tipo IOException .
boolean delete()	Elimina el fichero o directorio. Si es un directorio debe estar vacío. Devuelve true si se ha podido eliminar.
boolean exists()	Devuelve true si el fichero o directorio existe
String getName()	Devuelve el nombre del fichero o directorio
String getAbsolutePath()	Devuelve la ruta absoluta asociada al objeto File.

2.- La clase File

<code>boolean isDirectory()</code>	Devuelve true si es un directorio válido
<code>boolean isFile()</code>	Devuelve true si es un fichero válido
<code>long lastModified()</code>	Devuelve un valor en milisegundos que representa la última vez que se ha modificado (medido desde las 00:00:00 GMT, del 1 de Enero de 1970). Devuelve 0 si el fichero no existe o ha ocurrido un error.
<code>long length()</code>	Devuelve el tamaño en bytes del fichero. Devuelve 0 si no existe. Devuelve un valor indeterminado si es un directorio.
<code>String[] list()</code>	Devuelve un array de String con el nombre de los archivos y directorios que contiene el directorio indicado en el objeto File. Si no es un directorio devuelve null. Si el directorio está vacío devuelve un array vacío.
<code>String[] list(FilenameFilter filtro)</code>	Similar al anterior. Devuelve un array de String con el nombre de los archivos y directorios que contiene el directorio indicado en el objeto File que cumplen con el filtro indicado.
<code>boolean mkdir()</code>	Crea el directorio. Devuelve true si se ha podido crear.
<code>boolean mkdirs()</code>	Crea el directorio incluyendo los directorios no existentes especificados en la ruta <i>padre</i> del directorio a crear. Devuelve true si se ha creado el directorio y los directorios no existentes de la ruta padre.
<code>boolean renameTo(File dest)</code>	Cambia el nombre del fichero por el indicado en el parámetro dest. Devuelve true si se ha realizado el cambio.

2.- La clase File

```
package otroejemplofile;
import java.io.File;
import java.io.IOException;
/**
 * @author OLG
 */
public class OtroEjemploFile2 {

    public static void main(String[] args) {

        File fich = new File("ficherol.txt"); //Asociamos el objeto fich al fichero ficherol.txt

        try { //Creo ficherol.txt
            if (fich.createNewFile()) {
                System.out.println("Fichero 1 creado correctamente...");
            } else {
                System.out.println("No se ha podido crear el fichero 1...");
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }

        if (fich.isFile())//el fichero existe físicamente y es un fichero (no directorio)
        {
            System.out.println("Nombre del fichero: " + fich.getName());
            System.out.println("Ruta relativa: " + fich.getPath());
            System.out.println("Ruta absoluta: " + fich.getAbsoluteFile());
            System.out.println("Se puede escribir: " + fich.canRead());
            System.out.println("Se puede leer: " + fich.canWrite());
            System.out.println("Tamaño: " + fich.length());
            System.out.println("Es un fichero: " + fich.isFile());
        }
    }
}
```

• Ejemplo 1:

```
run:
Fichero 1 creado correctamente...
Nombre del fichero: ficherol.txt
Ruta relativa: ficherol.txt
Ruta absoluta: C:\Users\OLG\Documents\NetBeansProjects\OtroEjemploFile\ficherol.txt
Se puede escribir: true
Se puede leer: true
Tamaño: 0
Es un fichero: true
BUILD SUCCESSFUL (total time: 0 seconds)
```

2.- La clase File

- Ejemplo 2:

```
package otroejemplofile;
import java.io.File;
import java.io.IOException;
/**
 * @author OLG
 */
public class OtroEjemploFile {

    public static void main(String[] args) {
        //Asociaciones
        File directorio = new File("Mis Documentos");
        File fich1 = new File(directorio, "fichero1.txt");
        File fich2 = new File(directorio, "fichero2.txt");

        directorio.mkdir(); //Creamos el directorio "Mis Documentos"

        if (directorio.exists()) { //Comprobamos que se ha creado
            System.out.println(" Mis Documentos existe");
        } else{
            System.out.println("Mis Documentos NO existe");
        }

        try {
            if (fich1.createNewFile()) {
                System.out.println("Fichero 1 creado correctamente...");
            } else {
                System.out.println("No se ha podido crear el fichero 1...");
            }
            if (fich2.createNewFile()) {
                System.out.println("Fichero 2 creado correctamente...");
            } else {
                System.out.println("No se ha podido crear el fichero 2...");
            }
        } catch (IOException e) { //Entra aquí si no encuentra el directorio
            System.out.println(e.getMessage());
        }
    }
}
```

```
//renombramos el fichero 1
if(fich1.renameTo(new File(directorio, "fichero1-nuevo"))){
    System.out.println("fichero1.txt renombrado a fichero1-nuevo");
} else {
    System.out.println("No se ha podido renombrar fichero1.txt");
}

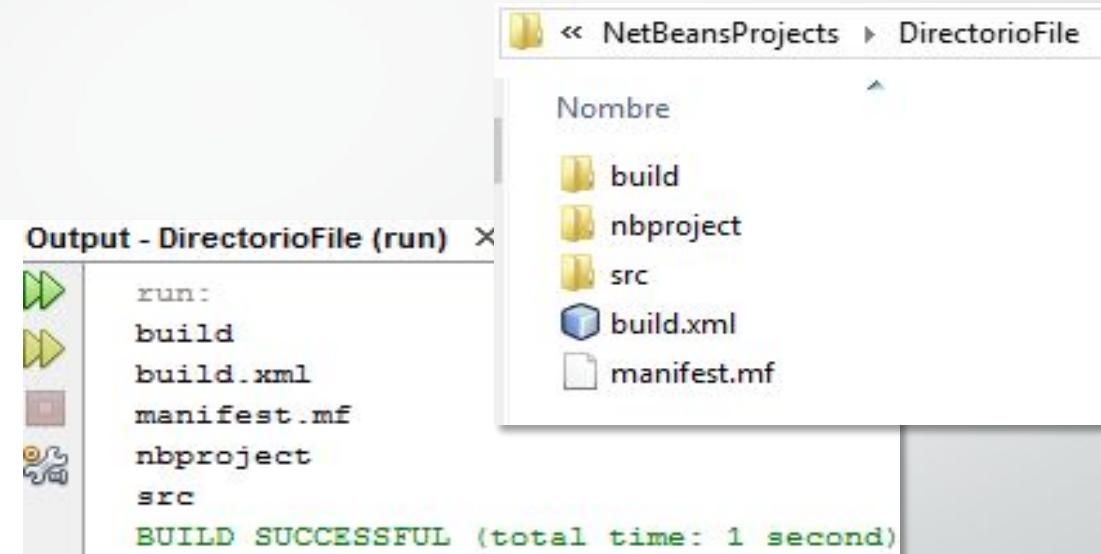
//borramos el fichero 2
if (fich2.delete()) {
    System.out.println("fichero2.txt borrado");
} else {
    System.out.println("No se ha podido borrar fichero2.txt");
}
```

2.- La clase File

- Ejemplo 3: uso de list()

```
package directoriofile;
import java.io.File;
/**
 * @author Oscar Laguna Garcia
 */
public class DirectorioFile {

    public static void main(String[] args) {
        File directorio = new File(".");
        String[] lista = directorio.list();
        for (int i = 0; i < lista.length; i++) {
            System.out.println(lista[i]);
        }
    }
}
```



EJERCICIOS

- **Ejercicio 02.- (OBLIGATORIO)** Utilizando la clase File, desarrolla una aplicación que realice las siguientes acciones:
 - Cree un directorio que se llame “ejemplo”.
 - Dentro de él crea dos ficheros llamados “practica1.txt” y “practica2.txt”
 - Renombra el fichero practica1.txt y llámalo “renombrado.txt”.
 - Elimina el fichero “practica2.txt”.
- Crea un método para cada acción, que vaya notificando al usuario de la acción que se está llevando a cabo.
- Controla las excepciones mediante la cláusula **throws**.
- Nota: En caso de duda, puedes consultar los ejemplos de la página Web: <http://puntocomnoesunlenguaje.blogspot.com.es/2013/05/clase-file-javascript.html>

EJERCICIOS

- **Ejercicio 03.- (OPTATIVO)** Crea un proyecto nuevo en NetBens y, dentro de él (desde el explorador de archivos de Windows), crea una carpeta llamada “DiasSemana”, que contenga 7 ficheros de texto, uno por cada día de la semana (lunes.txt, martes.txt, ...)
- Realiza un programa Java que muestre los ficheros del directorio “DiasSemana” y el **número** total (7) de ficheros que contiene.

EJERCICIOS

- **Ejercicio 04.- (OBLIGATORIO)** Crea un proyecto nuevo en NetBens y, dentro de él (desde el explorador de archivos de Windows), crea una carpeta llamada “DiasSemana”, que contenga 7 ficheros:lunes.txt, martes.dat,miércoles.bin, jueves.txt, viernes.mp3, sábado.txt y domingo.dat
- Escribir un programa donde le pidamos al usuario una extensión de archivo (txt, dat, bin...) y mostraremos los nombres de los archivos del directorio “DiasSemana”, cuya extensión coincida con la que nos introduzca el usuario.

JAVA

ENTRADA / SALIDA EN JAVA



3. E / S est\'andar

3.- E / S estándar

- JAVA, cuando lee un dato, no distingue si lo hace desde la entrada estándar (teclado), desde la memoria, desde un fichero, desde la red... Lo mismo ocurre con la salida, lo mismo le da mostrar un dato por pantalla, mostrarlo por impresora, volcarlo a un fichero...
- Para conseguir este nivel de abstracción en JAVA se trabaja con **FLUJOS** (en inglés, **Stream**), donde un flujo de información no es más que un objeto que hace de intermediario entre nuestro programa y el origen o el destino de la información.

3.- E / S estándar

- Entonces, para que un programa pueda obtener información desde un origen tiene que abrir un flujo y leer la información. Análogamente, para que un programa pueda enviar información a un destino tiene que abrir un flujo y escribir la información:

LEER	ESCRIBIR
Abrir un flujo desde un origen Mientras haya información { Leer información } Cerrar el flujo	Abrir un flujo hacia un destino Mientras haya información { Escribir información } Cerrar el flujo

3.- E / S estándar

- Todas las clases necesarias para leer y escribir datos en flujos, pertenecen al paquete `java.io`.
- Estas clases se pueden dividir en dos:
 - **Las que manejan Flujos de Bytes** (8 bits). Se utilizan en la E/S basada en datos, como por ejemplo lectura de los bits de una imagen, lectura de bits que llegan por una red...
 - **Las que manejan Flujos de Caracteres** (caracteres Unicode de 16 bits). Se utilizan en la E/S basada en texto, como por ejemplo lectura de caracteres por teclado o la lectura de un fichero de texto.

3.- E / S estándar

● Clases de flujo de entrada:

- Se utilizan para leer datos de una fuente de entrada (archivo, cadena o memoria)
- Flujo de bytes: **InputStream**, **BufferedInputStream**, **DataInputStream**, **FileInputStream**
- Flujo de caracteres: **Reader**, **BufferedReader**, **FileReader**

● Clases de flujo de salida:

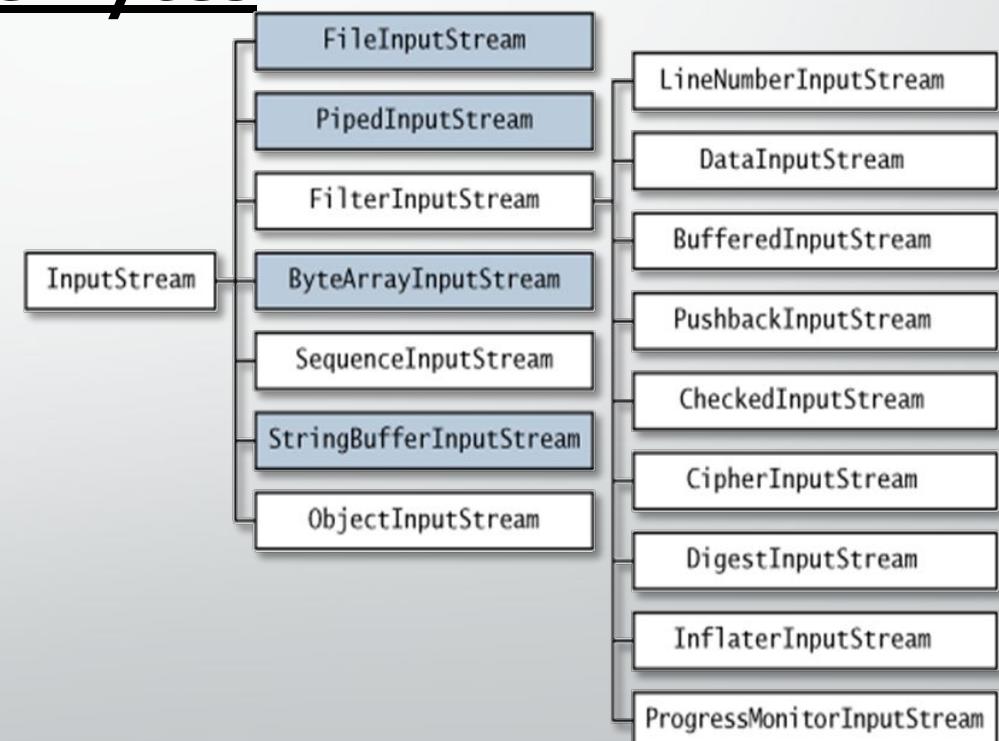
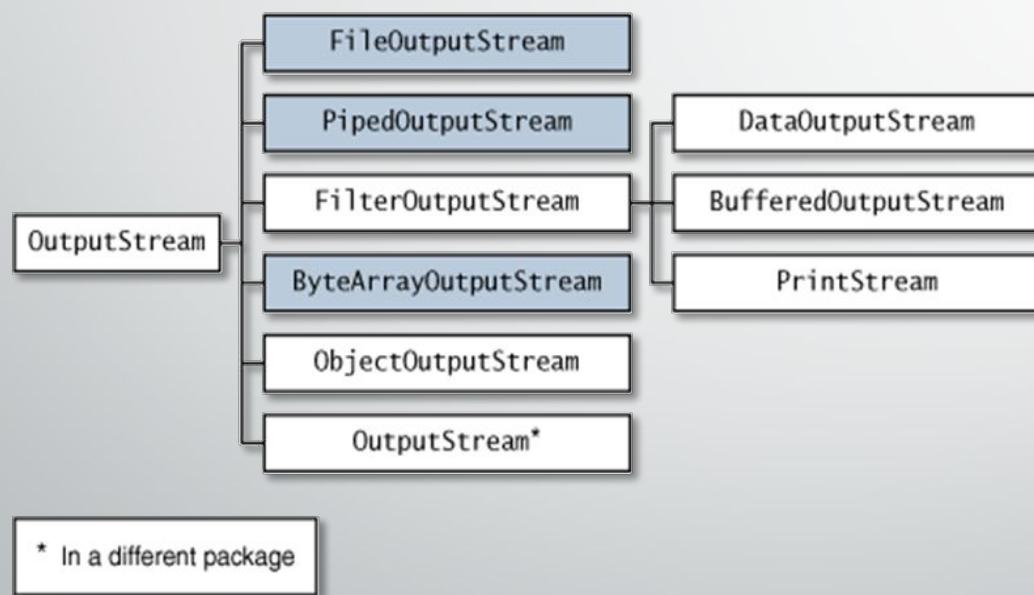
- Son las homólogas a las clases de flujo de entrada y se utilizan para enviar flujos de datos a dispositivos de salida
- Flujo de bytes: **OutputStream**, **PrintStream**, **BufferedOutputStream**, **DataOutputStream** y **FileOutputStream**
- Flujo de caracteres : **Writer**, **PrintWriter**, **FileWriter**

● Clases de archivo:

- **File** y **RandomAccessFile** (mayor control sobre los archivos)

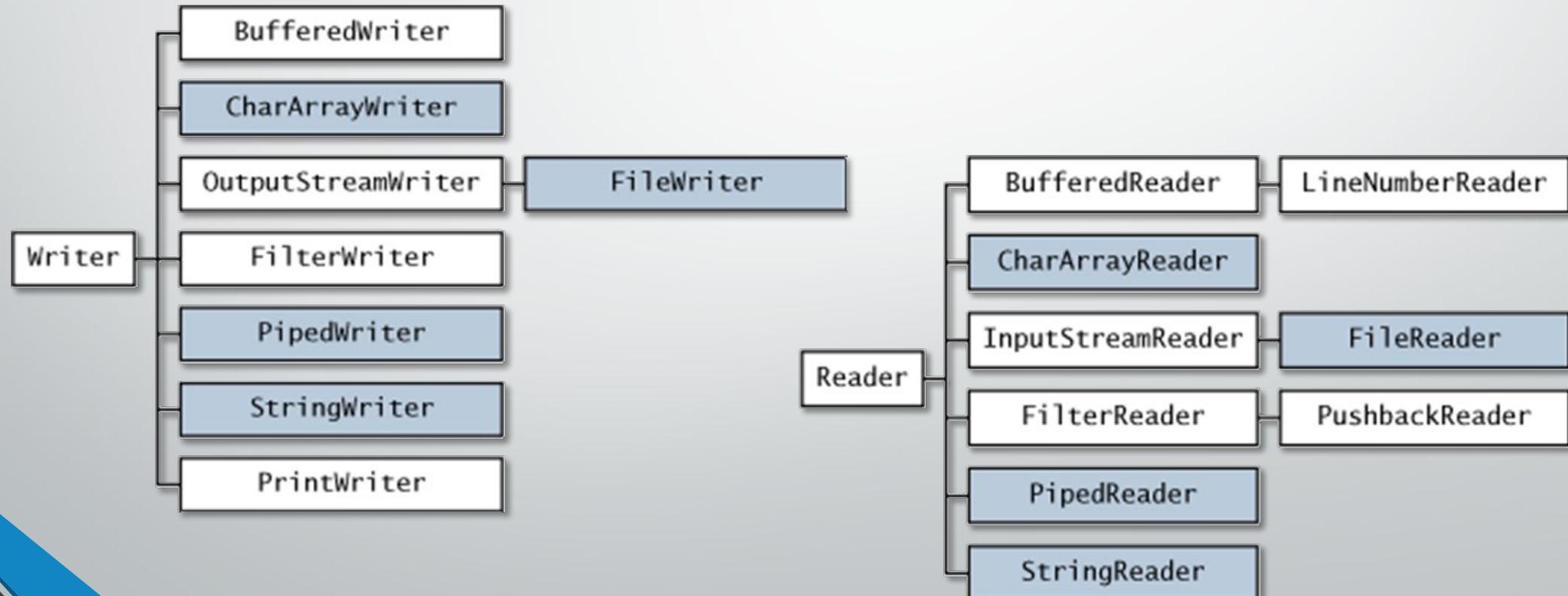
3.- E / S estándar

- Clases para manejar flujos de Bytes:



3.- E / S est ndar

- Clases para manejar flujos de caracteres:



JAVA

ENTRADA / SALIDA EN JAVA

4. E / S de Ficheros de Texto

4.- E / S de Ficheros de Texto

- **¿Qué es un fichero?**

- Secuencia de bytes que se encuentra guardado en un dispositivo de almacenamiento: Disco duro, CD, DVD, memoria USB, ...
- Se puede leer y/o escribir
- Se identifica mediante un nombre (*pathname*). Ejemplo: C:\ejemplo.txt

- **¿Qué tipos de ficheros hay?**

- Programas: contienen instrucciones en código máquina.
- Datos: contienen información legible, como números (enteros o reales), secuencias de caracteres, ...
- En algunos sistemas operativos (como Linux) también son ficheros los directorios, los dispositivos...

4.- E / S de Ficheros de Texto

• Los ficheros de texto y binarios

- Tipos de ficheros de datos:
 - De bytes (binarios): pensados para ser leídos por un programa.
 - De caracteres (de texto): están pensados para ser leídos y/o creados por una persona.

Fichero binario		Fichero de texto	
0	00000000	0	'1' (código ASCII 0x31)
1	00000000	1	'4' (código ASCII 0x34)
2	00000000	2	'h' (código ASCII 0x68)
3	00001110	3	'o' (código ASCII 0x6F)
4	00000000	4	'T' (código ASCII 0x6C)
5	00000000	5	'a' (código ASCII 0x61)
6	00000000
7	00100001		
...	...		

4.- E / S de Ficheros de Texto

• Lectura de un fichero de texto en Java

- Utilizaremos la clase `FileReader` para abrir un archivo de texto, y los métodos de esta clase para leer los caracteres del archivo.
- Sin embargo, si queremos leer líneas completas, utilizaremos la clase `BufferedReader`. Para ello podemos construir un `BufferedReader` a partir del `FileReader` de la siguiente forma:

`FileReader fr = new FileReader ("C:\\archivo.txt"); //Creo un flujo para leer los carácter del fichero`

`BufferedReader br = new BufferedReader(fr); //Creo un buffer para leer líneas del fichero`

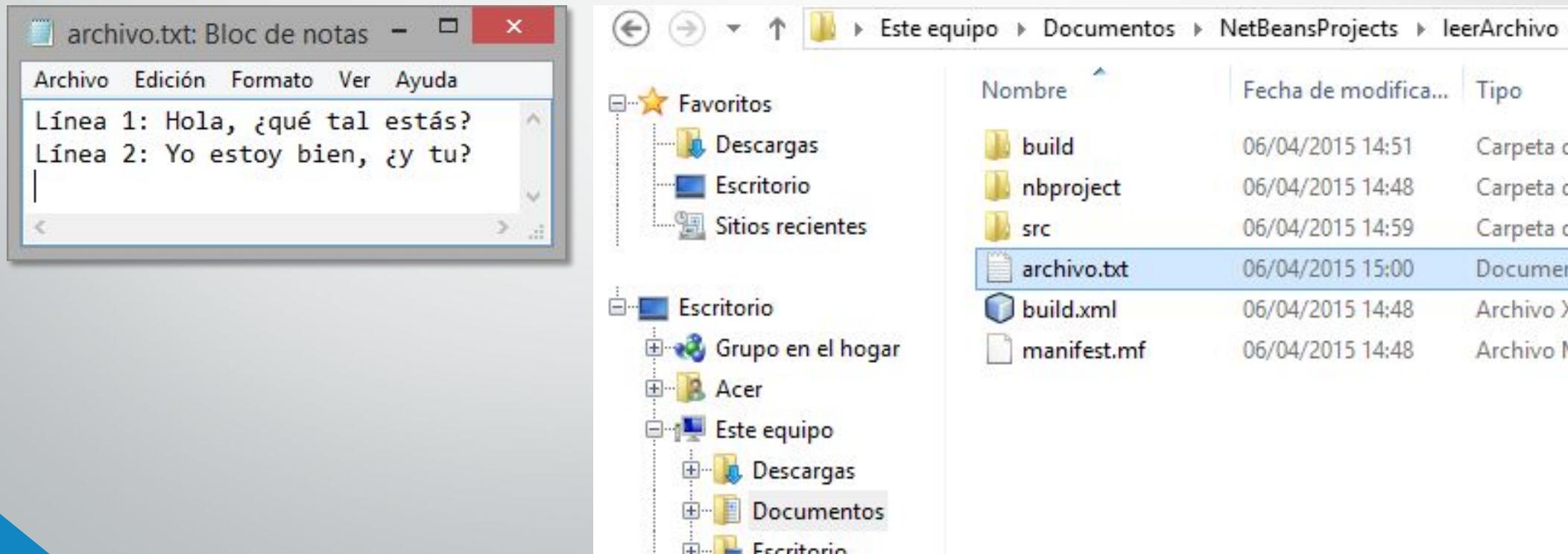
...

`String linea = br.readLine(); //readLine() es un método que me permite leer toda una línea.`

4.- E / S de Ficheros de Texto

- La apertura del fichero y su posterior lectura pueden lanzar excepciones que debemos capturar. Por ello, la apertura del fichero y la lectura debe meterse en un bloque **try-catch**.
- El fichero hay que cerrarlo cuando terminemos con él. Por ello, después del try-catch, se utiliza el bloque **finally** y dentro de él, los **close ()** de los streams.
- Ejemplo en el que leemos desde JAVA el fichero “archivo.txt” creado con el Bloc de Notas:

4.- E / S de Ficheros de Texto



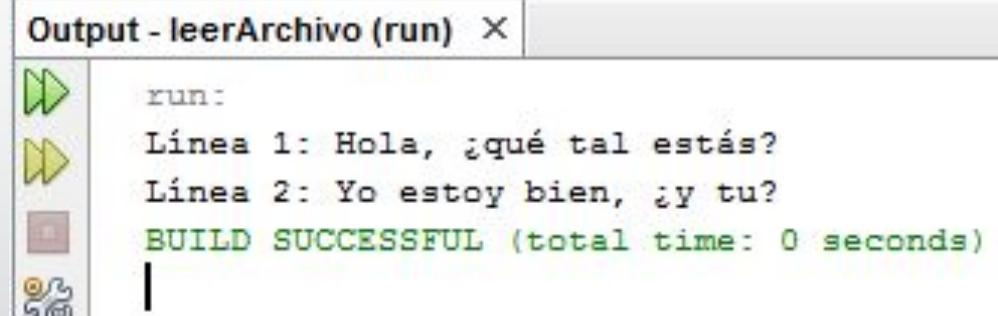
```
package lecturaficherotexto;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
/**
 * @author olagunag01
 */
public class LecturaFicheroTexto {

    public static void leerFichero(BufferedReader br) throws IOException {
        String linea;
        linea = br.readLine();
        while (linea != null) {
            System.out.println(linea);
            linea = br.readLine();
        }
    }

    public static void main(String[] args) {
        FileReader fr = null;
        BufferedReader br = null;
        try {
            fr = new FileReader("archivo.txt");
            br = new BufferedReader(fr);
            leerFichero(br);
        } catch (FileNotFoundException e) {
            System.out.println("No se ha encontrado el archivo");
            System.out.println(e.getMessage());
        } catch (IOException e) {
            System.out.println("Error de lectura");
            System.out.println(e.getMessage());
        } finally {
            if (br != null) {
                try {
                    br.close();
                } catch (IOException e) {
                    System.out.println(e.getMessage());
                }
            }
            if (fr != null) {
                try {
                    fr.close();
                } catch (IOException e) {
                    System.out.println(e.getMessage());
                }
            }
        }
    }
}
```

4.- E / S de Ficheros de Texto

Output - leerArchivo (run) X

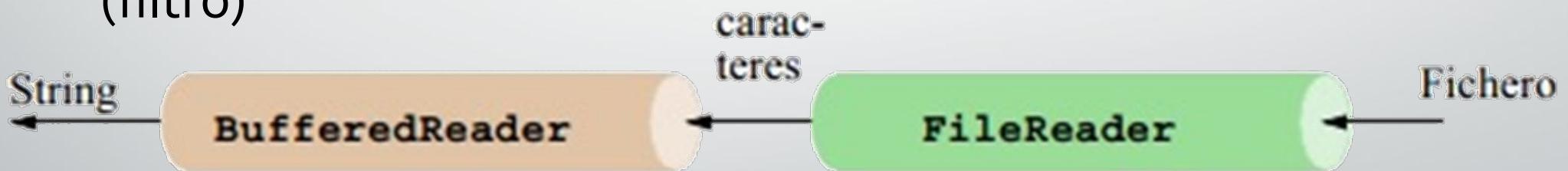


```
run:
Línea 1: Hola, ¿qué tal estás?
Línea 2: Yo estoy bien, ¿y tu?
BUILD SUCCESSFUL (total time: 0 seconds)
```

4.- E / S de Ficheros de Texto

- En resumen:

- Para la lectura de un fichero de texto en Java se realiza con la pareja de flujos **FileReader** (iniciador) y **BufferedReader** (filtro)



4.- E / S de Ficheros de Texto

- Operaciones con FileReader:

Descripción	Declaración
Constructor. Requiere el nombre del fichero. Si no existe lanza FileNotFoundException	<code>FileReader(String s)</code> <code>throws FileNotFoundException</code>

- Operaciones con BufferedReader:

Descripción	Declaración
Constructor. Requiere un Reader	<code>BufferedReader(Reader reader)</code>
Leer un string. Retorna null si se ha llegado al final Lanza IOException si se produce un error al acceder al Reader	<code>String readLine()</code> <code>throws IOException</code>
Cerrar. Lanza IOException si se produce un error al acceder al Reader	<code>void close()</code> <code>throws IOException</code>

La lectura se realiza línea a línea con `readLine()`

4.- E / S de Ficheros de Texto

- Escritura de un fichero de texto en Java
 - En JAVA es posible escribir en un fichero de texto tanto el contenido de variables como cadenas de texto (Strings)
 - Para ello se usa la pareja de flujos **FileWriter** (iniciador) y **PrintWriter** (filtro):



- Operaciones con **FileWriter**:

4.- E / S de Ficheros de Texto

Descripción	Declaración
Constructor. Requiere el nombre del fichero. Lo crea si no existe. Si existe se borran sus contenidos. Lanza <code>FileNotFoundException</code> si el fichero no se puede crear	<code>FileWriter(String s)</code> <code>throws IOException,</code> <code>FileNotFoundException</code>
Igual que el anterior, salvo en que cuando añade es true no se borran los contenidos, sino que los datos se añaden al final del fichero	<code>FileWriter(String s, boolean añade)</code> <code>throws IOException,</code> <code>FileNotFoundException</code>

Descripción	Declaración
Constructor. Requiere un Writer	<code>PrintWriter(Writer writer)</code>
Constructor. Crea el <code>FileWriter</code> internamente	<code>PrintWriter(String nomFich)</code>
Escribir un string	<code>void print(String str)</code>
Escribir un string con retorno de línea	<code>void println(String str)</code>
Escribe los argumentos con el formato deseado	<code>printf(String formato, Object... args)</code>
Sincroniza e informa si ha habido un error	<code>boolean checkError()</code>
Sincronizar	<code>void flush()</code>
Cerrar	<code>void close()</code>

```
package escrituraficherotexto;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
/**
 * @author olagunag01
 */
public class EscrituraFicheroTexto {

    public static void escribirFichero(PrintWriter pw) {
        for (int i = 0; i < 10; i++) {
            pw.println("Esto es la linea " + i);
        }
        System.out.println("Fichero guardado con éxito");
    }

    public static void main(String[] args) {
        FileWriter fw = null;
        PrintWriter pw = null;
        try {
            fw = new FileWriter("ejemplo.txt");
            pw = new PrintWriter(fw);
            escribirFichero(pw);
        } catch (IOException e) {
            System.out.println("Error de entrada/salida");
            System.out.println(e.getMessage());
        } finally {
            if (pw != null) {
                pw.close();
            }
            if (fw != null) {
                try {
                    fw.close();
                } catch (IOException e) {
                    System.out.println("Error de entrada/salida");
                    System.out.println(e.getMessage());
                }
            }
        }
    }
}
```

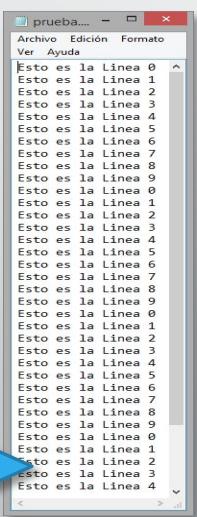
4.- E / S de Ficheros de Texto

The screenshot shows the NetBeans IDE interface. On the left, a file browser window titled 'Documentos' displays the project structure: NetBeansProjects/EscrituraFicheroTexto. It lists several files: build, nbproject, src, build.xml, ejemplo.txt (which is selected and highlighted in blue), and manifest.mf. Below the file browser is a code editor window showing Java code for writing to a file. To the right of the code editor is a terminal window displaying the output of the program, which is 10 lines of text: 'Esto es la linea 0' through 'Esto es la linea 9'.

```
ejemplo.txt: Bloc de notas
Archivo Edición Formato Ver
Esto es la linea 0
Esto es la linea 1
Esto es la linea 2
Esto es la linea 3
Esto es la linea 4
Esto es la linea 5
Esto es la linea 6
Esto es la linea 7
Esto es la linea 8
Esto es la linea 9
```

4.- E / S de Ficheros de Texto

- Si queremos añadir texto al final de un fichero ya existente, ponemos **"true"** como segundo parámetro del constructor de `FileWriter`
- Si ejecutamos varias veces el programa:



```
package escrituraficherotexto;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
/**
 * @author olagunag01
 */
public class EscrituraFicheroTexto {

    public static void escribirFichero(PrintWriter pw) {
        for (int i = 0; i < 10; i++) {
            pw.println("Esto es la linea " + i);
        }
        System.out.println("Fichero guardado con éxito");
    }

    public static void main(String[] args) {
        FileWriter fw = null;
        PrintWriter pw = null;
        try {
            fw = new FileWriter("ejemplo.txt",true);
            pw = new PrintWriter(fw);
            escribirFichero(pw);
        } catch (IOException e) {
```

EJERCICIOS

- **Ejercicio 05.- (OBLIGATORIO)** Realiza un programa en JAVA en el que muestres un menú que te permita 3 opciones:
 1. Añadir un contacto a un fichero de texto ("agenda.txt") en el que indiques en cada línea:
 - Un Nombre.
 - Una Edad.
 - Un número de teléfono.
 2. Mostrar por pantalla el contenido del fichero de texto creado.
 3. Salir del Programa.
 - Captura las excepciones que veas necesarias.

EJERCICIOS

- **Ejercicio 06.- (OPTATIVO)** Realiza un programa en JAVA en el que muestres un menú que te permita 3 opciones:
 1. Volcado de un array con los 100 primeros números pares a un fichero de texto. El nombre del fichero lo elegirá el usuario.
 2. Mostrar por pantalla el contenido del fichero de texto creado.
 3. Salir del Programa.
- Captura las excepciones que veas necesarias.

EJERCICIOS

• Ejercicio 07.- (OBLIGATORIO)

- Crea una aplicación donde pidamos el nombre de un fichero por teclado y un texto que queramos escribir en el fichero. Después de crear el fichero con la información introducida, deberás mostrar por pantalla el texto del fichero pero variando entre mayúsculas y minúsculas.

- Por ejemplo, si escribo:

"Bienvenidos a Plasencia" deberá devolver
"bIENVENIDOS A pLASENCIA".

- Ejemplo de ejecución:



```
Output - ArchivoIntercambia (run) X
run:
Introduzca el nombre del fichero
prueba.txt
Procedemos a llenar el fichero
Introduzca el contenido de la linea 1
Bienvenido, a mi Programa
Otra linea?
si
Introduzca el contenido de la linea 2
Esta es la ultima LINEA que EscRIBO
Otra linea?
no
Fichero creado y relleno con éxito
Procedemos a leer el fichero y a intercambiar las mayúsculas-minúsculas
Resultado:
bIENVENIDO, A MI pROGRAMA
eSTA ES LA ULTIMA linea QUE eSCRIBO
BUILD SUCCESSFUL (total time: 1 minute 49 seconds)
```

EJERCICIOS

- **Ejercicio 08.- (OBLIGATORIO)** Crea un fichero de texto, en el Bloc de Notas, con el nombre y contenido que tú quieras.
- Luego, crea una aplicación en JAVA que lea este fichero de texto (*¡OJO! □ Carácter a carácter, no línea a línea, para que en este ejercicio no uses BufferedReader*) y muestre su contenido por pantalla sin espacios.
- Por ejemplo, si un fichero contiene el texto “Esto es una prueba”, deberá mostrar por pantalla “Estoesunaprueba”.

EJERCICIOS

- **Ejercicio 09.- (OPTATIVO)** Realiza un programa en JAVA que lea un archivo creado en el bloc de notas llamado frase.txt que contiene una línea de texto. Luego, el programa creará un archivo llamado fraseinvertida.txt que contendrá el texto invertido del archivo frase.txt.
- Ejemplo: Si *frase.txt* contiene Hola, que tal *fraseinvertida.txt* contendrá lat euq ,aloH

EJERCICIOS

- **Ejercicio 10.- (OPTATIVO)** Realiza un programa en JAVA en el que le pidas al usuario su DNI y:
 - En caso de que el DNI sea correcto lo introducirás en un fichero llamado dni.txt.
 - En caso de que el DNI sea incorrecto avisarás al usuario del error y no lo meterás en el fichero.
 - Cuando vuelvas a ejecutar el programa no se sobrescribirá el fichero, sino que se seguirán añadiendo DNIs válidos al final del fichero dni.txt.

- Algoritmo para el cálculo del NIF:

- Calculo del Módulo de 23 (resto de la división) del número del DNI.
- La letra corresponde a la que haya para ese valor en la siguiente tabla:

CÓDIGO PARA LA LETRA DEL D.N.I. O DEL N.I.F.																							
RESTO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
LETRA	T	R	W	A	G	M	Y	F	P	D	X	B	N	J	Z	S	Q	V	H	L	C	K	E

Ejemplos de ejecución:

Introduzca un DNI: **12345678A**
DNI invalido
--FIN DEL PROGRAMA--

Introduzca un DNI: **12345678Z**
DNI valido e introducido en el fichero.
--FIN DEL PROGRAMA--

EJERCICIOS

- **Ejercicio 11.- (OPTATIVO)** Crea un programa en JAVA que lea un archivo creado en el bloc de notas llamado numeros.txt que contenga varias líneas, y en cada una de ellas un número. Luego, el programa te dará la suma total de todos los números del fichero.

EJERCICIOS

- **Ejercicio 12.- (OBLIGATORIO)** Realiza un programa en JAVA que lea dos archivos creados en el bloc de notas, llamados pares.txt e impares.txt.
- Estos archivos contienen varias líneas y en cada una de ellas un número par o impar respectivamente.
- Luego, el programa, te creará un fichero llamado resultados.txt en el que cada línea contendrá la suma de los números de los ficheros pares.txt e impares.txt de esa misma línea.

EJERCICIOS

- **Ejercicio 12.- (OPTATIVO)** Escriba un programa en JAVA que contenga un método, de nombre escribirCadenasEnArchivo, que reciba un array de 4 cadenas de caracteres (previamente relleno por el usuario) y vuelque su contenido a un archivo cuyo nombre también se recibirá por parámetro.
- Las cadenas quedarán separadas en el archivo por un asterisco.

EJERCICIOS

- **Ejercicio 12.- (OPTATIVO)** Realiza un programa en JAVA que lea un archivo creado en el bloc de notas llamado masnumeros.txt que contiene varias líneas y en cada una de ellas hay varios números separados por ; (punto y coma).
- El programa te mostrará por pantalla la suma de todos los números del fichero.

EJERCICIOS

- **Ejercicio 13.- (OPTATIVO)** Realizar un programa que lea de teclado (la entrada estándar) los siguientes datos:
 - Nombre y apellido de un supuesto becario.
 - Sexo (H-M)
 - Edad (20-60)
 - Número de suspensos del curso anterior (0-4).
 - Residencia familiar (SI-NO)
 - Ingresos anuales de la familia.
- Los datos se almacenan en un fichero llamado "DatosBeca.txt".
- Cuando vuelvas a ejecutar el programa no se sobrescribirá el fichero, sino que se seguirán añadiendo posibles becarios al final del fichero.

EJERCICIOS

- **Ejercicio 14.- (OPTATIVO)** Realizar un programa que partiendo del fichero “DatosBeca.txt” calcule la cuantía de la beca (en caso de que la haya). El total de la beca se calcula como sigue:
 - Base fija de 1500€
 - Si los ingresos anuales de la familia son menores o iguales a la media (12.000€), la beca se incrementa en 500€, en caso contrario no lleva complementos.
 - Si la edad de la persona es inferior a 23 años, 200€ de gratificación, si es mayor no hay gratificación.
 - Si no hay suspensos en el curso anterior, hay una gratificación de 500€, 1 suspenso 200€, si hay 2 suspensos o más no hay beca.
 - Si vive de alquiler (no residencia familiar), 1000€ más de gratificación.
- Visualizar el nombre de cada uno de los becarios y su cuantía total (sólo los que tienen beca).

JAVA

ENTRADA / SALIDA EN JAVA

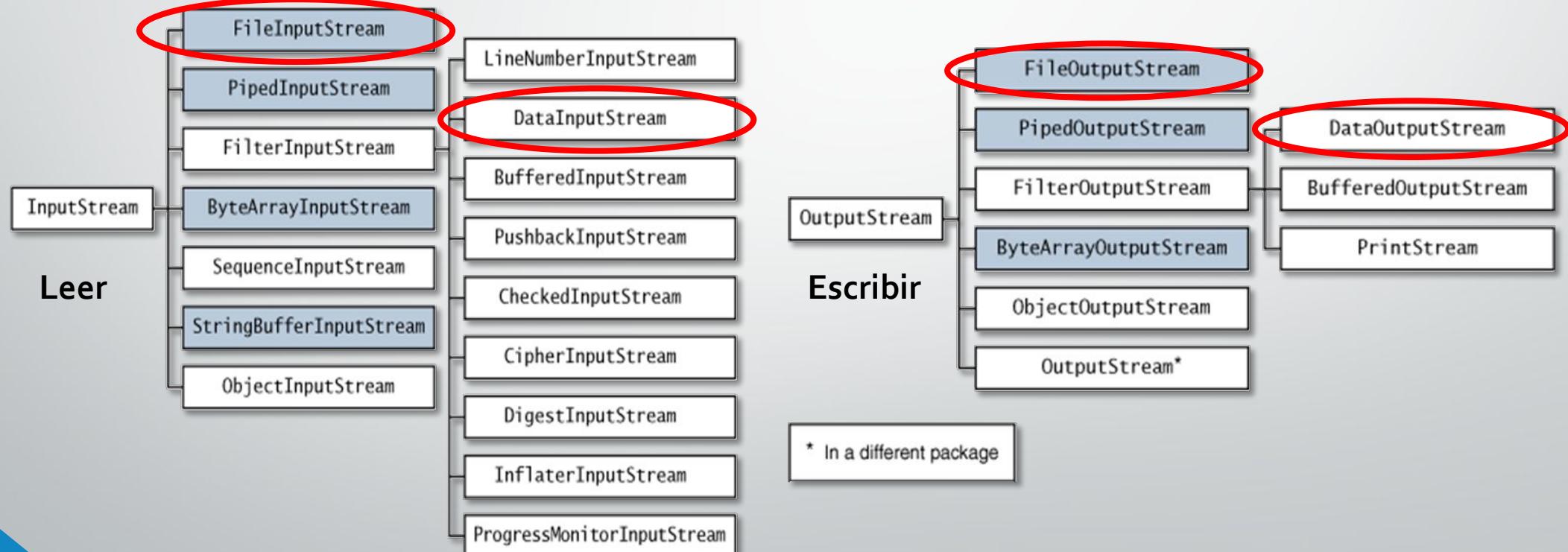
5. E / S de Ficheros Binarios

5.- E / S de Ficheros Binarios

- Para leer y escribir ficheros binarios se hace exactamente igual que con los de texto, pero en vez de usar los "**Reader**" y los "**Writer**", se usan los "**InputStream**" y los "**OutputStream**".
- Además, en lugar de los **readLine()** y **println()**, hay que usar los métodos **read()** y **write()** de array de bytes.

5.- E / S de Ficheros Binarios

- Clases para manejar flujos de Bytes:



5.- E / S de Ficheros Binarios

• **InputStream: lectura de ficheros binarios**

- FileInputStream (iniciador): lee bytes de un fichero.
- DataInputStream (filtro): nos permiten leer datos en un InputStream indicando un tipo.

• **OutputStream: Escritura de ficheros binarios**

- FileOutputStream (iniciador): escribe bytes en un fichero
- DataOutputStream (filtro): nos permiten escribir datos en un OutputStream indicando un tipo.

5.- E / S de Ficheros Binarios

- **Operaciones de la clase FileInputStream:**

Descripción	Declaración
Constructor. Requiere el nombre del fichero. Si el fichero no existe lanza FileNotFoundException	<code>FileInputStream(String s) throws FileNotFoundException</code>

- **Operaciones de la clase FileOutputStream:**

Descripción	Declaración
Constructor. Requiere el nombre del fichero. Lo crea si no existe. Si existe se borran sus contenidos. Lanza FileNotFoundException si el fichero no se puede crear	<code>FileOutputStream(String s) throws FileNotFoundException</code>
Igual que el anterior, salvo en que cuando añade es true no se borran los contenidos, sino que los datos se añaden al final del fichero	<code>FileOutputStream(String s, boolean añade) throws FileNotFoundException</code>
Sincronizar	<code>void flush()</code>
Cerrar	<code>void close()</code>

5.- E / S de Ficheros Binarios

- La clase *DataInputStream* define diversos métodos *readXXX* que son variaciones del método *read* de la superclase (*InputStream*) para leer datos de tipo primitivo:
 - *boolean readBoolean();*
 - *byte readByte();*
 - *int readUnsignedByte();*
 - *short readShort();*
 - *int readUnsignedShort();*
 - *char readChar();*
 - *int readInt();*
 - *String readUTF();*
 - *long readLong();*
 - *float readFloat();*
 - *double readDouble();*

5.- E / S de Ficheros Binarios

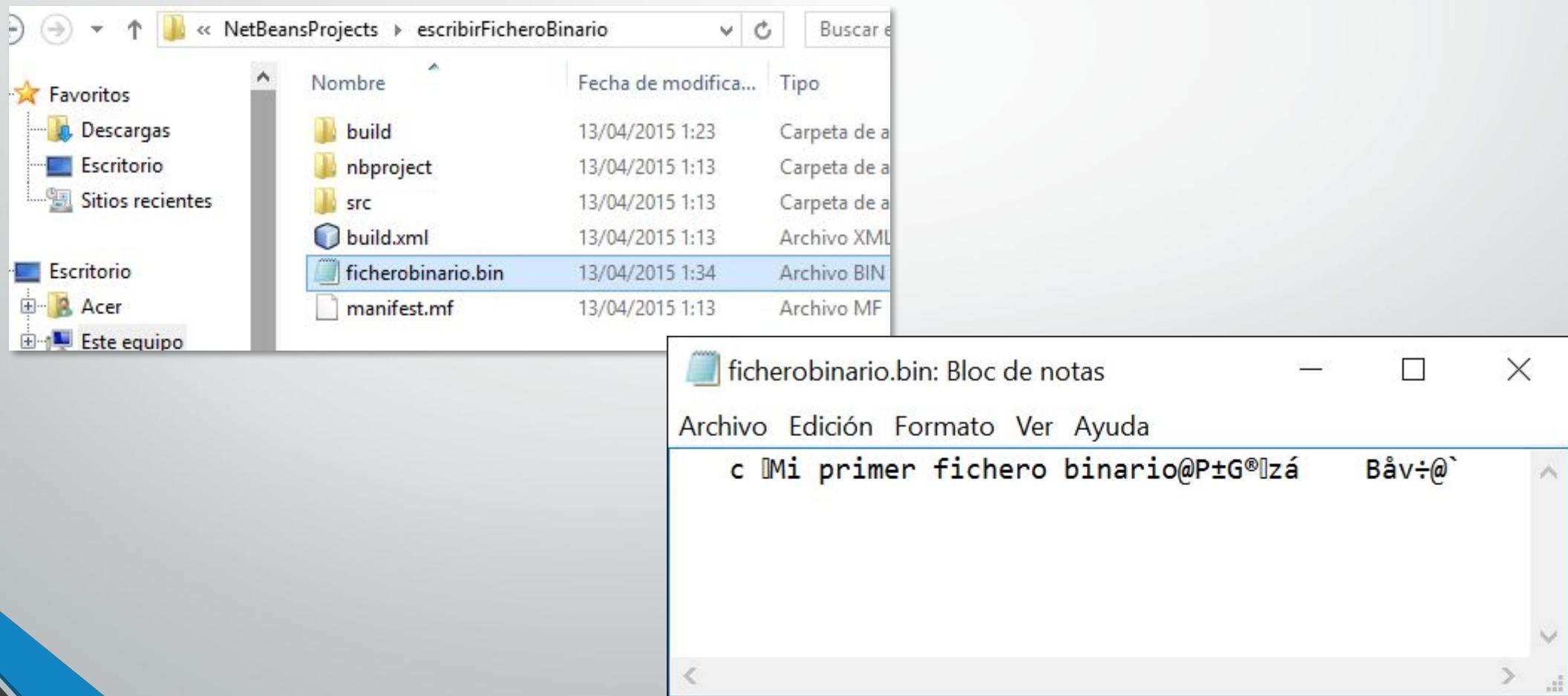
- La clase *DataOutputStream* define diversos métodos *writeXXX* que son variaciones del método *write* de la superclase (*OutputStream*) para escribir datos de tipo primitivo:
 - *void writeBoolean(boolean v);*
 - *void writeByte(int v);*
 - *void writeBytes(String s);*
 - *void writeShort(int v);*
 - *void writeChars(String s);*
 - *void writeUTF(String s);*
 - *void writeChar(int v);*
 - *void writeInt(int v);*
 - *void writeLong(long v);*
 - *void writeFloat(float v);*
 - *void writeDouble(double v);*

5.- E / S de Ficheros Binarios

```
public class EscrituraFicheroBinario {  
  
    public static void escribirFichero(DataOutputStream dos) throws IOException {  
        dos.writeInt(99);  
        dos.writeUTF("Mi primer fichero binario");  
        dos.writeDouble(66.77);  
        dos.writeLong(1122334455);  
        dos.writeFloat(3.5f);  
  
        System.out.println("Fichero guardado con éxito");  
    }  
  
    public static void main(String[] args) {  
        FileOutputStream fos = null;  
        DataOutputStream dos = null;  
  
        try {  
            fos = new FileOutputStream("ficherobinario.bin");  
            dos = new DataOutputStream(fos);  
            escribirFichero(dos);  
  
        } catch (FileNotFoundException ex) {  
            System.out.println("Error de acceso al fichero");  
            System.out.println(ex.getMessage());  
        } catch (IOException ex) {  
            System.out.println("Error al escribir en el fichero");  
            System.out.println(ex.getMessage());  
        } finally {  
            if (dos != null) {  
                try {  
                    dos.close();  
                } catch (IOException ex) {  
                    System.out.println(ex.getMessage());  
                }  
            }  
            if (fos != null) {  
                try {  
                    fos.close();  
                } catch (IOException ex) {  
                    System.out.println(ex.getMessage());  
                }  
            }  
        }  
    }  
}
```

- Esta vez, vamos a comenzar con la escritura de un archivo binario:

5.- E / S de Ficheros Binarios



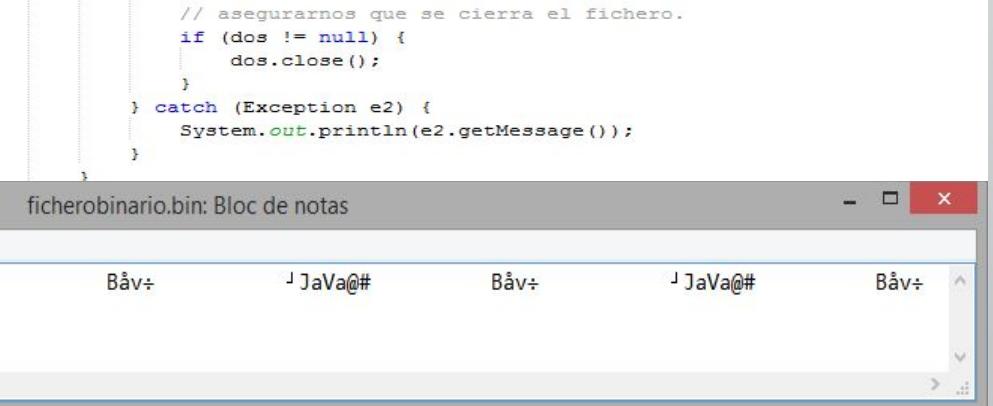
5.- E / S de Ficheros Binarios

- Si queremos añadir texto al final de un fichero ya existente, ponemos **"true"** como segundo parámetro del constructor de FileOutputStream
- Si ejecutamos varias veces el programa:

```
package escribirficherobinario;
import java.io.*;
/**
 * @author Oscar Laguna García
 */
public class EscribirFicheroBinario {

    public static void escribirFichero(DataOutputStream dos) throws Exception {
        dos.writeInt(9); //Escribimos un numero entero
        dos.writeUTF("Java"); //Escribimos una cadena
        dos.writeDouble(9.5); //Escribimos un numero double
        dos.writeLong(1122334455); //Escribimos un numero long
    }

    public static void main(String[] args) {
        DataOutputStream dos=null;
        try {
            FileOutputStream fos=new FileOutputStream("ficherobinario.bin",true);
            dos=new DataOutputStream(fos);
            escribirFichero(dos);
        }catch(Exception e){
            System.out.println(e.getMessage());
        } finally {
            try {
                // Nuevamente aprovechamos el finally para
                // asegurarnos que se cierra el fichero.
                if (dos != null) {
                    dos.close();
                }
            } catch (Exception e2) {
                System.out.println(e2.getMessage());
            }
        }
    }
}
```



```
package lecturaficherobinario;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
/**
 * @author olagunag01
 */
public class LecturaFicheroBinario {

    public static void leerFichero(DataInputStream dis) throws IOException {
        System.out.println(dis.readInt());
        System.out.println(dis.readUTF());
        System.out.println(dis.readDouble());
        System.out.println(dis.readLong());
        System.out.println(dis.readFloat());
    }

    public static void main(String[] args) {
        FileInputStream fis = null;
        DataInputStream dis = null;
        try {
            fis = new FileInputStream("ficherobinario.bin");
            dis = new DataInputStream(fis);
            leerFichero(dis);
        } catch (FileNotFoundException ex) {
            System.out.println("Fichero no encontrado");
            System.out.println(ex.getMessage());
        } catch (IOException ex) {
            System.out.println("Error de lectura");
            System.out.println(ex.getMessage());
        } finally {
            if (dis != null) {
                try {
                    dis.close();
                } catch (IOException ex) {
                    System.out.println(ex.getMessage());
                }
            }
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException ex) {
                    System.out.println(ex.getMessage());
                }
            }
        }
    }
}
```

5.- E / S de Ficheros Binarios

- Lectura de un archivo binario:

```
Output - leerFicheroBinario (run) ×
run:
9
Java
9.5
1122334455
BUILD SUCCESSFUL (total time: 0 seconds)
```

5.- E / S de Ficheros Binarios

- Para la lectura de un archivo binario del cual no sabemos la longitud, se utiliza un bucle while (true):

```
Output - LecturaFicheroBinario (run) ×
run:
99
Mi primer fichero binario
66.77
1122334455
3.5
Fin de lectura
null
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
package lecturaficheroBinario;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
/**
 * @author olagunag01
 */
public class LecturaFicheroBinario {

    public static void leerFichero(DataInputStream dis) throws IOException {
        while (true) { //Ocurre una EOFException y el bucle while(true) termina.
            System.out.println(dis.readInt());//Leemos un entero y lo mostramos
            System.out.println(dis.readUTF());//Leemos una cadena y la mostramos
            System.out.println(dis.readDouble());//Leemos un double y lo mostramos
            System.out.println(dis.readLong());//Leemos un long y lo mostramos
            System.out.println(dis.readFloat());//Leemos un float y lo mostramos
        }
    }

    public static void main(String[] args) {
        FileInputStream fis = null;
        DataInputStream dis = null;
        try {
            fis = new FileInputStream("ficherobinario.bin");
            dis = new DataInputStream(fis);
            leerFichero(dis);
        } catch (FileNotFoundException ex) {
            System.out.println("Fichero no encontrado");
            System.out.println(ex.getMessage());
        } catch (IOException ex) {
            System.out.println("Fin de lectura");
            System.out.println(ex.getMessage());
        } finally {
            if (dis != null) {
                try {
                    dis.close();
                } catch (IOException ex) {
                }
            }
        }
    }
}
```

EJERCICIOS

- **Ejercicio 15.- (OBLIGATORIO)** Realiza un programa en JAVA en el que muestres un menú que te permita 3 opciones:
 1. Añadir un contacto a un fichero binario ("agenda.bin") en el que indiques:
 - Un Nombre.
 - Una Edad.
 - Un número de teléfono.
 2. Mostrar por pantalla el contenido del fichero de texto creado.
 3. Salir del Programa.
 - Captura las excepciones que veas necesarias.

EJERCICIOS

- **Ejercicio 16.- (OPTATIVO)** Realiza un programa en JAVA en el que muestres un menú que te permita 3 opciones:
 1. Volcado de un array con los 100 primeros números impares a un fichero binario. El nombre del fichero lo elegirá el usuario.
 2. Mostrar por pantalla el contenido del fichero binario creado.
 3. Salir del Programa.
- Captura las excepciones que veas necesarias.

EJERCICIOS

- **Ejercicio 17.- (OPTATIVO)** Realiza un programa en JAVA en el que le pidas al usuario su DNI y:
 - En caso de que el DNI sea correcto lo introducirás en un fichero llamado dni.bin.
 - En caso de que el DNI sea incorrecto avisarás al usuario del error y no lo meterás en el fichero.
 - Cuando vuelvas a ejecutar el programa no se sobrescribirá el fichero, sino que se seguirán añadiendo DNIs válidos al final del fichero dni.bin.

- Algoritmo para el cálculo del NIF:

- Calculo del Módulo de 23 (resto de la división) del número del DNI.
- La letra corresponde a la que haya para ese valor en la siguiente tabla:

CÓDIGO PARA LA LETRA DEL D.N.I. O DEL N.I.F.																							
RESTO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
LETRA	T	R	W	A	G	M	Y	F	P	D	X	B	N	J	Z	S	Q	V	H	L	C	K	E

Ejemplos de ejecución:

Introduzca un DNI: **12345678A**
DNI invalido
--FIN DEL PROGRAMA--

Introduzca un DNI: **12345678Z**
DNI valido e introducido en el fichero.
--FIN DEL PROGRAMA--

EJERCICIOS

- **Ejercicio 18.- (OBLIGATORIO)** Realiza un programa en JAVA que lea un fichero llamado dni.bin y vuelque los dni correctos a un fichero llamado dni_correctos.bin, y los dni incorrectos los vuelque a un fichero llamado dni_incorrectos.bin.
- Un menú nos planteará las siguientes opciones:
 1. Añadir dnis al fichero dni.bin.
 2. Volcar los dnis a su archivo correspondiente.
 3. Visualizar archivo dni_correctos.
 4. Visualizar archivo dni_incorrectos.
 5. Salir.

EJERCICIOS

- **Ejercicio 19.- (OBLIGATORIO)** Realiza un programa que lea de teclado los siguientes datos:
 - Nombre y apellido de un supuesto becario.
 - Sexo (H-M)
 - Edad (20-60)
 - Número de suspensos del curso anterior (0-4).
 - Residencia familiar (SI-NO)
 - Ingresos anuales de la familia.
- Los datos se almacenan en un fichero binario llamado "datosbeca.bin".

EJERCICIOS

- **Ejercicio 20.- (OBLIGATORIO)** Realiza un programa que, partiendo del fichero binario “datosbeca.bin”, calcule la cuantía de la beca (en caso de que la haya). El total de la beca se calcula como sigue:
 - Base fija de 1500€
 - Si los ingresos anuales de la familia son menores o iguales a la media (12.000€), la beca se incrementa en 500€, en caso contrario no lleva complementos.
 - Si la edad de la persona es inferior a 23 años, 200€ de gratificación, si es mayor no hay gratificación.
 - Si no hay suspensos en el curso anterior, hay una gratificación de 500€, 1 suspenso 200€, si hay 2 suspensos o más no hay beca.
 - Si vive de alquiler (no residencia familiar), 1000€ más de gratificación.
- Visualizar el nombre de cada uno de los becarios y su cuantía total (sólo los que tienen beca).



JAVA

ENTRADA / SALIDA EN JAVA

6. Serialización

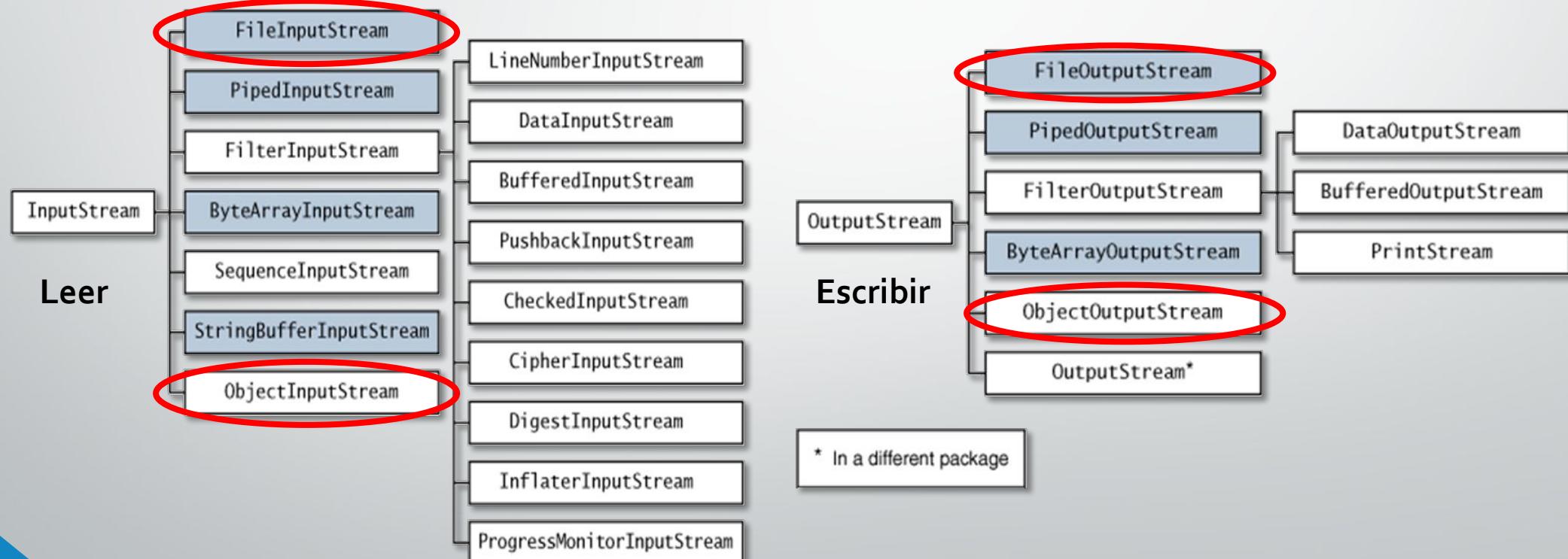
6.- Serialización

- Serialización consiste en convertir un objeto en un conjunto de bytes para así poder enviarlo a través de red o guardarlos en un fichero. Después, ese objeto, deberá ser reconstruido al otro lado de la red o leerlo del fichero.(deserialización)
- Para que un objeto sea serializable basta con que su clase implemente la interfaz **Serializable**.

Ejemplo: **public class Alumnos implements Serializable {**

6.- Serialización

- Clases para manejar flujos de Bytes:



6.- Serialización

- **FileInputStream** y **FileOutputStream**: (iniciadores)
Serán los flujos encargados de leer o escribir bytes en los ficheros.
- **ObjectInputStream** y **ObjectOutputStream**: (filtros)
Serán los flujos encargados de manejar la lectura y la escritura de los objetos a partir de los 2 flujos anteriores.

6.- Serialización

- Para escribir un objeto en el fichero (serialización) utilizaremos el método **writeObject(objeto)** de la clase ObjectOutputStream.
- Para leer un objeto de un fichero (deserialización) utilizaremos el método **readObject()** de la clase ObjectInputStream. Además, deberemos hacer un casting de los datos leídos antes de guardarlos en el objeto destino.

- **Ejemplo de Serialización**, con una clase Alumno serializable, y una clase Test donde manejamos los flujos:

```
package lecturaescrituraficheroobjetos;
import java.io.Serializable;
/**
 * @author olagunag01
 */
public class Alumno implements Serializable {
    private String nombre;
    private String dni;
    private int edad;

    public Alumno(String nombre, String dni, int edad) {
        this.nombre = nombre;
        this.dni = dni;
        this.edad = edad;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getDni() {
        return dni;
    }
    public void setDni(String dni) {
        this.dni = dni;
    }
    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

6.- Serialización

```
package lecturaescrituraficheroobjetos;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
/*
 * @author olagunag01
 */
public class Test {

    public static void rellenar(ObjectOutputStream oos) throws IOException {
        Alumno alumnoAuxiliar;
        alumnoAuxiliar = new Alumno("Pepe", "12345678A", 22);
        oos.writeObject(alumnoAuxiliar);
        alumnoAuxiliar = new Alumno("Ana", "23456789B", 11);
        oos.writeObject(alumnoAuxiliar);
        alumnoAuxiliar = new Alumno("Andres", "34567890C", 33);
        oos.writeObject(alumnoAuxiliar);
    }

    public static void escribirFichero() {
        FileOutputStream fos = null;
        ObjectOutputStream oos = null;
        try {
            fos = new FileOutputStream("alumnos.obj", true);
            oos = new ObjectOutputStream(fos);
            rellenar(oos);
        } catch (FileNotFoundException ex) {
            System.out.println("Fichero no encontrado");
            System.out.println(ex.getMessage());
        } catch (IOException ex) {
            System.out.println("Error de entrada-salida");
            System.out.println(ex.getMessage());
        } finally {
            if (oos != null) {
                try {
                    oos.close();
                } catch (IOException ex) {
                    System.out.println(ex.getMessage());
                }
            }
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException ex) {
                    System.out.println(ex.getMessage());
                }
            }
        }
    }
}
```

```
public static void mostrarFichero(ObjectInputStream ois) throws IOException, ClassNotFoundException {
    Alumno alumnoAuxiliar;
    while (true) {
        alumnoAuxiliar = (Alumno) ois.readObject();
        System.out.println("Nombre: " + alumnoAuxiliar.getNombre());
        System.out.println("DNI: " + alumnoAuxiliar.getDni());
        System.out.println("Edad: " + alumnoAuxiliar.getEdad());
        System.out.println("*****");
    }
}

public static void leerFichero() {
    FileInputStream fis = null;
    ObjectInputStream ois = null;
    try {
        fis = new FileInputStream("alumnos.obj");
        ois = new ObjectInputStream(fis);
        mostrarFichero(ois);
    } catch (FileNotFoundException ex) {
        System.out.println("Fichero no encontrado");
    } catch (ClassNotFoundException ex) {
        System.out.println("Clase no encontrada");
        System.out.println(ex.getMessage());
    } catch (IOException ex) {
        System.out.println("Fin de lectura");
    } finally {
        if (ois != null) {
            try {
                ois.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
        if (fis != null) {
            try {
                fis.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
    }
}

public static void main(String[] args) {
    escribirFichero();
    System.out.println("Datos guardados con éxito");
    System.out.println("Procedemos a visualizar el fichero");
    leerFichero();
}
```

EJERCICIOS

- **Ejercicio 21.- (OBLIGATORIO)** Realiza un programa en JAVA en el que le pidas al usuario las notas de las 6 asignaturas del Ciclo de DAM y las guarde en un fichero. Posteriormente leerá el fichero y te calculará la nota media del curso.
 - Cada una de las asignaturas serán un objeto cuyos atributos serán el nombre y la nota.
 - Con el constructor podrás asignar directamente el nombre de la asignatura al crear el objeto. En cambio, el atributo nota, será el usuario quien lo introduzca mediante un método que controle que la nota tenga un valor numérico (no letras) entre 0 y 10.
 - Ejemplo de ejecución:

Por favor, introduzca la nota de Programación: 6,5

Introduzca la nota de Lenguajes de Marcas: 7,5

Introduzca la nota de Bases de Datos: 7,5

Introduzca la nota de Entornos de Desarrollo: 8

Introduzca la nota de Sistemas Informáticos: 6,5

Por último, introduzca la nota de Formación y Orientación Laboral: 6

**** Notas almacenadas en el fichero ****

...Leyendo el fichero y calculando media....

Su nota media del curso es de: 7

EJERCICIOS

- **Ejercicio 22.- (OBLIGATORIO)** Realiza un programa en JAVA en el que le pidas al usuario las notas de las 6 asignaturas del Ciclo de DAM y las guarde en un fichero. Posteriormente leerá el fichero y te calculará la nota media del curso.
 - Cada una de las asignaturas serán un objeto **guardados en un array de 6 posiciones**, cuyos atributos serán el nombre y la nota que posteriormente volcarás al fichero.
 - Con el constructor podrás asignar directamente el nombre de la asignatura al crear el objeto. En cambio, el atributo nota, será el usuario quien lo introduzca mediante un método que controle que la nota tenga un valor numérico (no letras) entre 0 y 10.

Por favor, introduzca la nota de Programación: 6,5

Introduzca la nota de Lenguajes de Marcas: 7,5

Introduzca la nota de Bases de Datos: 7,5

Introduzca la nota de Entornos de Desarrollo: 8

Introduzca la nota de Sistemas Informáticos: 6,5

Por último, introduzca la nota de Formación y Orientación Laboral: 6

***** Notas almacenadas en el fichero *****

...Leyendo el fichero y calculando media....

Su nota media del curso es de: 7

EJERCICIOS

- **Ejercicio 23.- (OBLIGATORIO)** Realiza un programa en JAVA en el que le pidas al usuario las notas de las 6 asignaturas del Ciclo de DAM y las guarde en un fichero. Posteriormente leerá el fichero y te calculará la nota media del curso.
 - Cada una de las asignaturas serán un objeto **guardados en una lista**, cuyos atributos serán el nombre y la nota, que posteriormente volcarás al fichero.
 - Con el constructor podrás asignar directamente el nombre de la asignatura al crear el objeto. En cambio, el atributo nota, será el usuario quien lo introduzca mediante un método que controle que la nota tenga un valor numérico (no letras) entre 0 y 10.

Por favor, introduzca la nota de Programación: 6,5

Introduzca la nota de Lenguajes de Marcas: 7,5

Introduzca la nota de Bases de Datos: 7,5

Introduzca la nota de Entornos de Desarrollo: 8

Introduzca la nota de Sistemas Informáticos: 6,5

Por último, introduzca la nota de Formación y Orientación Laboral: 6

***** Notas almacenadas en el fichero *****

...Leyendo el fichero y calculando media....

Su nota media del curso es de: 7

6.- Serialización

- Si ejecutamos de nuevo nuestro programa, y queremos añadir nuevos objetos al fichero, veremos que se produce una excepción (**StreamCorruptedException**). Esto ocurre porque cada vez que abrimos el fichero para escribirlo, el `ObjectOutputStream`, escribe una nueva cabecera. Luego se irán añadiendo los objetos que vayamos escribiendo. Por ello, el fichero contendrá dos cabeceras, lo que origina la excepción.

Contenido del fichero									
Primera sesión con el fichero					Segunda sesión con el fichero. Le añadimos datos				
cabecera	Persona	Persona	Persona	Persona	cabecera	Persona	Persona	Persona	Persona

6.- Serialización

- Para solucionar esto tenemos 2 posibles soluciones:
 1. Nos creamos una clase que herede de la clase ObjectOutputStream y redefinir (@override) el método writeStreamHeader(), de tal forma que no haga nada.
 2. Volcar todos los objetos del fichero a una Lista, añadir los nuevos objetos a la Lista y por último volcar la Lista al fichero.

6.- Serialización

Solución 1:

```
package lecturaescrituraficheroobjetos;
import java.io.Serializable;
/**
 * @author Oscar Laguna García
 */
public class Alumno implements Serializable {

    private String nombre;
    private String dni;
    private int edad;

    public Alumno(String nombre, String dni, int edad) {
        this.nombre = nombre;
        this.dni = dni;
        this.edad = edad;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getDni() {
        return dni;
    }
    public void setDni(String dni) {
        this.dni = dni;
    }
    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

The screenshot shows an IDE interface with several windows:

- Projects**: Shows a project named "LecturaEscrituraFicheroObjetosMiObject" with a Source Packages folder containing "lecturaescrituraficheroobjetos" which has "Alumno.java", "MiObjectOutputStream.java", and "Test.java".
- Files**: Shows the "Alumno.java" file content.
- Services**: Shows the "MiObjectOutputStream.java" file content.
- Alumno.java**: The code for the Alumno class, identical to the one in the left panel.
- Test.java**: The code for the Test class.
- MiObjectOutputStream.java**: The code for the MiObjectOutputStream class, which extends ObjectOutputStream. It overrides the writeStreamHeader() method with a comment stating "//No hace nada".
- Navigator**: Shows the members of the MiObjectOutputStream class.

6.- Serialización

```
package lecturaescrituraficheroobjetos;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.Scanner;

/**
 * @author Oscar Laguna García
 */
public class Test {

    public static String pedirNombre() {
        Scanner entrada = new Scanner(System.in);
        System.out.println("Introduzca el nombre del alumno");
        return entrada.nextLine();
    }

    public static String pedirDNI() {
        Scanner entrada = new Scanner(System.in);
        System.out.println("Introduzca el DNI del alumno");
        return entrada.nextLine();
    }

    public static int pedirEdad() {
        Scanner entrada = new Scanner(System.in);
        System.out.println("Introduzca la edad del alumno");
        return entrada.nextInt();
    }
}
```

```
public static boolean pedirContinuar() {
    Scanner entrada = new Scanner(System.in);
    boolean respuesta = false;
    System.out.println("¿Desea introducir otro alumno? (si/no)");
    if (entrada.nextLine().equalsIgnoreCase("si")) {
        respuesta = true;
    }
    return respuesta;
}

public static void rellenar(ObjectOutputStream oos) throws IOException {
    Alumno alumnoAuxiliar;
    do {
        alumnoAuxiliar = new Alumno(pedirNombre(), pedirDNI(), pedirEdad());
        oos.writeObject(alumnoAuxiliar);
    } while (pedirContinuar());
}
```

6.- Serialización

```
public static void escribirFichero() {
    FileOutputStream fos = null;
    ObjectOutputStream oos = null;
    try {
        File fichero = new File("alumnos.obj");
        if (!fichero.exists()) { //Si el fichero no existe
            fos = new FileOutputStream("alumnos.obj");
            oos = new ObjectOutputStream(fos);
        }
        else {
            fos = new FileOutputStream("alumnos.obj", true);
            oos = new MiObjectOutputStream(fos);
        }
        llenar(oos);
    } catch (FileNotFoundException ex) {
        System.out.println("Fichero no encontrado");
        System.out.println(ex.getMessage());
    } catch (IOException ex) {
        System.out.println("Error de entrada-salida");
        System.out.println(ex.getMessage());
    } finally {
        if (oos != null) {
            try {
                oos.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
        if (fos != null) {
            try {
                fos.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
    }
}
```

```
public static void leerFichero() {
    FileInputStream fis = null;
    ObjectInputStream ois = null;
    try {
        fis = new FileInputStream("alumnos.obj");
        ois = new ObjectInputStream(fis);
        mostrarFichero(ois);
    } catch (FileNotFoundException ex) {
        System.out.println("Fichero no encontrado");
    } catch (ClassNotFoundException ex) {
        System.out.println("Clase no encontrada");
        System.out.println(ex.getMessage());
    } catch (IOException ex) {
        System.out.println("Fin de lectura");
    } finally {
        if (ois != null) {
            try {
                ois.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
        if (fis != null) {
            try {
                fis.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
    }
}
```

6.- Serialización

```
public static void mostrarFichero(ObjectInputStream ois) throws IOException, ClassNotFoundException {
    Alumno alumnoAuxiliar;
    while (true) {
        alumnoAuxiliar = (Alumno) ois.readObject();
        System.out.println("Nombre: " + alumnoAuxiliar.getNombre());
        System.out.println("DNI: " + alumnoAuxiliar.getDni());
        System.out.println("Edad: " + alumnoAuxiliar.getEdad());
        System.out.println("*****");
    }
}

public static void main(String[] args) {
    escribirFichero();
    System.out.println("Datos guardados con éxito");
    System.out.println("Procedemos a visualizar el fichero");
    leerFichero();
}
```

6.- Serialización

Solución 2:

```
package lecturaescrituraficheroobjetos;
import java.io.Serializable;
/**
 * @author Oscar Laguna García
 */
public class Alumno implements Serializable {

    private String nombre;
    private String dni;
    private int edad;

    public Alumno(String nombre, String dni, int edad) {
        this.nombre = nombre;
        this.dni = dni;
        this.edad = edad;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getDni() {
        return dni;
    }
    public void setDni(String dni) {
        this.dni = dni;
    }
    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

```
public static void main(String[] args) {
    ArrayList <Alumno> lista = new ArrayList <>();
    File fichero = new File("alumnos.obj");
    if (!fichero.exists()) { //Si el fichero no existe
        escribirFichero(fichero);
    }
    else{
        volcarFicheroALista(fichero,lista);
        añadirAlumnos(lista);
        escribirFichero(lista,fichero); //Volcamos de la lista al fichero
    }
    System.out.println("Datos guardados con éxito");
    System.out.println("Procedemos a visualizar el fichero");
    leerFichero(fichero);
}
```

6.- Serialización

```
package lecturaescrituraficheroobjetos;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.Scanner;

/**
 * @author Oscar Laguna García
 */
public class Test {

    public static String pedirNombre() {
        Scanner entrada = new Scanner(System.in);
        System.out.println("Introduzca el nombre del alumno");
        return entrada.nextLine();
    }

    public static String pedirDNI() {
        Scanner entrada = new Scanner(System.in);
        System.out.println("Introduzca el DNI del alumno");
        return entrada.nextLine();
    }

    public static int pedirEdad() {
        Scanner entrada = new Scanner(System.in);
        System.out.println("Introduzca la edad del alumno");
        return entrada.nextInt();
    }
}
```

```
public static boolean pedirContinuar() {
    Scanner entrada = new Scanner(System.in);
    boolean respuesta = false;
    System.out.println("¿Desea introducir otro alumno? (si/no)");
    if (entrada.nextLine().equalsIgnoreCase("si")) {
        respuesta = true;
    }
    return respuesta;
}

public static void rellenar(ObjectOutputStream oos) throws IOException {
    Alumno alumnoAuxiliar;
    do {
        alumnoAuxiliar = new Alumno(pedirNombre(), pedirDNI(), pedirEdad());
        oos.writeObject(alumnoAuxiliar);
    } while (pedirContinuar());
}
```

6.- Serialización

```
public static void escribirFichero(File fichero) {
    FileOutputStream fos = null;
    ObjectOutputStream oos = null;
    try {
        fos = new FileOutputStream(fichero);
        oos = new ObjectOutputStream(fos);
        llenar(oos);
    } catch (FileNotFoundException ex) {
        System.out.println("Fichero no encontrado");
        System.out.println(ex.getMessage());
    } catch (IOException ex) {
        System.out.println("Error de entrada-salida");
        System.out.println(ex.getMessage());
    } finally {
        if (oos != null) {
            try {
                oos.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
        if (fos != null) {
            try {
                fos.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
    }
}
```

```
public static void mostrarFichero(ObjectInputStream ois) throws IOException, ClassNotFoundException {
    Alumno alumnoAuxiliar;
    while (true) {
        alumnoAuxiliar = (Alumno) ois.readObject();
        System.out.println("Nombre: " + alumnoAuxiliar.getNombre());
        System.out.println("DNI: " + alumnoAuxiliar.getDni());
        System.out.println("Edad: " + alumnoAuxiliar.getEdad());
        System.out.println("*****");
    }
}

public static void leerFichero(File fichero) {
    FileInputStream fis = null;
    ObjectInputStream ois = null;
    try {
        fis = new FileInputStream(fichero);
        ois = new ObjectInputStream(fis);
        mostrarFichero(ois);
    } catch (FileNotFoundException ex) {
        System.out.println("Fichero no encontrado");
    } catch (ClassNotFoundException ex) {
        System.out.println("Clase no encontrada");
        System.out.println(ex.getMessage());
    } catch (IOException ex) {
        System.out.println("Fin de lectura");
    } finally {
        if (ois != null) {
            try {
                ois.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
        if (fis != null) {
            try {
                fis.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
    }
}
```

6.- Serialización

```
public static void volcarFicheroALista(File fichero, ArrayList<Alumno> lista) {
    FileInputStream fis = null;
    ObjectInputStream ois = null;
    try {
        fis = new FileInputStream(fichero);
        ois = new ObjectInputStream(fis);
        Alumno alumnoAuxiliar;
        while (true) {
            alumnoAuxiliar = (Alumno) ois.readObject();
            lista.add(alumnoAuxiliar);
        }
    } catch (FileNotFoundException ex) {
        System.out.println("Fichero no encontrado");
    } catch (ClassNotFoundException ex) {
        System.out.println("Clase no encontrada");
        System.out.println(ex.getMessage());
    } catch (IOException ex) {
        System.out.println("Fin de lectura");
    } finally {
        if (ois != null) {
            try {
                ois.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
        if (fis != null) {
            try {
                fis.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
    }
}
```

```
public static void añadirAlumnos(ArrayList<Alumno> lista) {
    Alumno alumnoAuxiliar;
    do {
        alumnoAuxiliar = new Alumno(pedirNombre(), pedirDNI(), pedirEdad());
        lista.add(alumnoAuxiliar);
    } while (pedirContinuar());
}

public static void llenar(ArrayList<Alumno> lista, ObjectOutputStream oos) throws IOException {
    for (int i = 0; i < lista.size(); i++) {
        oos.writeObject(lista.get(i));
    }
}

public static void escribirFichero(ArrayList<Alumno> lista, File fichero) {
    FileOutputStream fos = null;
    ObjectOutputStream oos = null;
    try {
        fos = new FileOutputStream(fichero);
        oos = new ObjectOutputStream(fos);
        llenar(lista,oos);
    } catch (FileNotFoundException ex) {
        System.out.println("Fichero no encontrado");
        System.out.println(ex.getMessage());
    } catch (IOException ex) {
        System.out.println("Error de entrada-salida");
        System.out.println(ex.getMessage());
    } finally {
        if (oos != null) {
            try {
                oos.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
        if (fos != null) {
            try {
                fos.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
    }
}
```

6.- Serialización

- Para finalizar este punto de Serialización tan solo añadir que si queremos que algún atributo de un objeto no se vuelque al fichero bastará con utilizar el modificador **transient**. Ejemplo:

```
public class Cliente implements Serializable{  
    private String nombre;  
    private transient String password;  
}
```

EJERCICIOS

- **Ejercicio 24.- (OBLIGATORIO)** Desarrolle una aplicación en Java que determine el sueldo bruto para cada uno de los empleados de una empresa. La empresa paga la tarifa normal en las primeras 40 horas de trabajo de cada empleado, y paga tarifa y media en todas las horas trabajadas que excedan de 40.
 - El programa creará los objetos que quiera el usuario (uno para cada empleado) los meterá en una lista y luego los volcará a un fichero. Ten en cuenta que cuando se vuelva a ejecutar el programa se podrán seguir añadiendo empleados.
 - Por cada empleado se almacenará su nombre, el número de horas que trabajó, y la tarifa que cobra por una hora de trabajo.
 - Para finalizar el programa debe leer el fichero y determinar y mostrar el sueldo bruto de cada empleado. Ejemplo: **Pepe trabajó 42 horas, cobra 20 euros la hora por lo que le corresponde un sueldo de 860 euros.**
 - Nota 1: Realiza un método para establecer el nombre de cada empleado, otro para establecer para las horas y otro para establecer la tarifa.
 - Nota 2: Crea otros 3 métodos para devolver el nombre de cada empleado, otro para devolver las horas y otro para devolver la tarifa.

Nota 3: Utiliza un método que lea el fichero y que calcule el sueldo que corresponde a cada empleado.

EJERCICIOS

- **Ejercicio 25.- (OBLIGATORIO)** Desarrolle una aplicación en Java que determine el sueldo bruto para cada uno de los empleados de una empresa. La empresa paga la tarifa normal en las primeras 40 horas de trabajo de cada empleado, y paga tarifa y media en todas las horas trabajadas que excedan de 40.
 - El programa creará **los objetos que quiera el usuario** (uno para cada empleado) y los irá metiendo a un fichero. Ten en cuenta que cuando se vuelva a ejecutar el programa se podrán seguir añadiendo empleados. (Necesitarás redefinir el método writeStreamHeader())
 - Por cada empleado se almacenará su nombre, el número de horas que trabajó, y la tarifa que cobra por una hora de trabajo.
 - Para finalizar el programa debe leer el fichero y determinar y mostrar el sueldo bruto de cada empleado. Ejemplo: **Pepe trabajó 42 horas, cobra 20 euros la hora por lo que le corresponde un sueldo de 860 euros.**
 - **Nota 1:** Realiza un método para establecer el nombre de cada empleado, otro para establecer para las horas y otro para establecer la tarifa.
 - **Nota 2:** Crea otros 3 métodos para devolver el nombre de cada empleado, otro para devolver las horas y otro para devolver la tarifa.

Nota 3: Utiliza un método que lea el fichero y que calcule el sueldo que corresponde a cada empleado.

EJERCICIOS

- **Ejercicio 26.- (OBLIGATORIO)** Realizar un programa en JAVA en el realices la gestión de una pequeña tienda de bebidas. Para ello manejarás objetos que tendrán 3 atributos:
 - Nombre de la bebida
 - Precio
 - Stock.
- Realiza un método en el que le muestre al usuario un menú con 2 opciones:
 - 1.- Introducción de bebidas:** Los productos de la tienda (objetos) estarán dentro de un fichero de objetos y serán los que desee el usuario, al que le iremos preguntando si desea introducir otro. Cuando vuelva a entrar en el programa podrá seguir añadiendo productos si lo desea.

EJERCICIOS

2.- Comprar productos: Donde le mostraremos al usuario un listado, el usuario elegirá que producto comprar y luego le preguntemos cuantas unidades desea de él. Luego se le preguntará si desea comprar otro producto o salir. Por último se le mostrará el importe total de la compra.

- Notas:
 - Necesitarás actualizar el valor del stock de un producto cuando el usuario lo compre. En caso de que el usuario pida más unidades de las que quedan se le avisará por pantalla del error, se le comunicarán las unidades restantes y le preguntará si desea comprar otro producto.
 - Para poder comprar productos se recomienda volcar el fichero a una lista o array, hacer las operaciones pertinentes y actualización de stock, y cuando mostremos el importe total volcar los datos modificados de la lista al fichero.

JAVA

ENTRADA / SALIDA EN JAVA



7. Ficheros de acceso aleatorio

8.- Ficheros de acceso aleatorio

- Los ficheros con los que hemos trabajado hasta ahora (ya sean ficheros de texto o ficheros binarios con objetos serializados) no resultan adecuados para aplicaciones en las que no se desea leer un fichero de principio a fin; sino acceder al fichero como una base de datos, donde se salta de un registro a otro; cada uno en diferentes partes del fichero.
- Java proporciona una clase **RandomAccessFile** para este tipo de entrada/salida, donde se ha de acceder a un registro concreto dentro de un fichero, por lo que los registros deben ser de tamaño fijo.

8.- Ficheros de acceso aleatorio

- Para crear un Fichero de Acceso Aleatorio tenemos 2 posibilidades:
 - Con el nombre del fichero:

```
fichero = new RandomAccessFile( String nombre, String modo );
```
 - Con un objeto File:

```
fichero = new RandomAccessFile( File fichero, String modo );
```
- El argumento modo determina si se tiene acceso de sólo lectura ("r") o de lectura/escritura ("rw")

8.- Ficheros de acceso aleatorio

- Para acceder a la información de un fichero de acceso aleatorio se tienen acceso a todas las operaciones `read()` y `write()` de las clases `DataInputStream` y `DataOutputStream`.
- Además tenemos 3 métodos para movernos dentro de un fichero:
 - **`long getFilePointer();`**
 - Devuelve la posición actual del puntero del fichero.

8.- Ficheros de acceso aleatorio

- **void seek (long pos);**

- Coloca el puntero del fichero en una posición determinada. La posición se da como un desplazamiento en bytes desde el comienzo del fichero. La posición 0 marca el comienzo de ese fichero.

- **long length ();**

- Devuelve la longitud del fichero. La posición length() marca el final de ese fichero

- **skipBytes ();**

- Mueve el puntero de fichero hacia adelante el número de bytes especificado.

8.- Ficheros de acceso aleatorio

```
package accesoaleatorio;
import java.io.*;
import java.util.Scanner;
/**
 * @author Oscar Laguna García
 */
public class AccesoAleatorio {

    public static void escribirFichero(RandomAccessFile raf) throws Exception{
        int numero;
        Scanner entrada = new Scanner(System.in);
        System.out.print("Introduce un número entero para añadir al final del fichero: ");
        numero = entrada.nextInt(); //se lee el entero a añadir en el fichero
        raf.seek(raf.length()); //nos situamos al final del fichero
        raf.writeInt(numero); //se escribe el entero
    }

    public static void crearFichero(){
        RandomAccessFile raf=null;
        try{
            File fichero = new File ("fichero.obj"); //Creo fichero
            raf = new RandomAccessFile(fichero,"rw"); //Se lo paso al flujo
            escribirFichero(raf);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        } finally {
            try {
                if (raf != null) {
                    raf.close();
                }
            } catch (Exception e2) {
                System.out.println(e2.getMessage());
            }
        }
    }
}
```

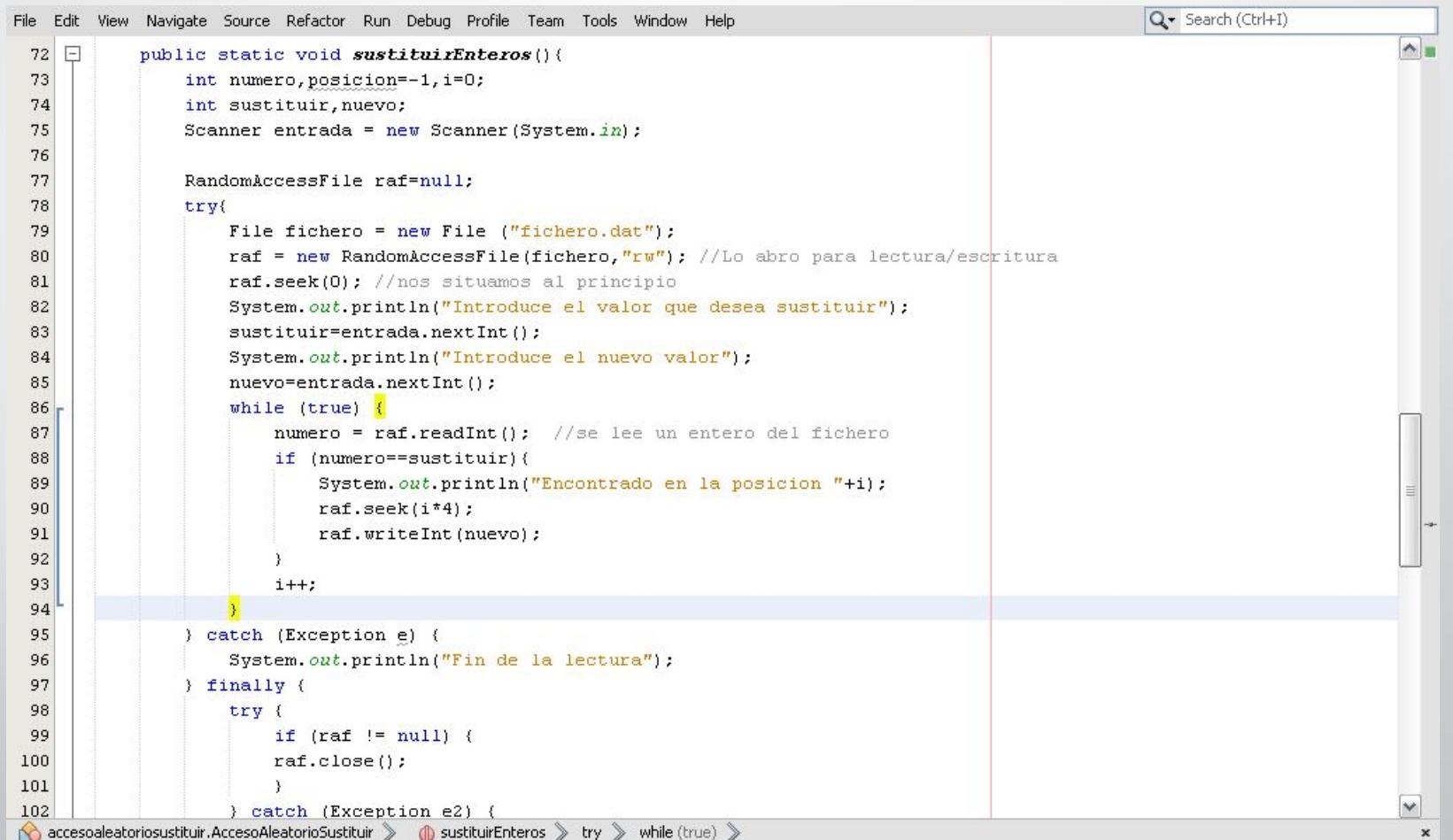
```
public static void mostrarFichero(RandomAccessFile raf) throws Exception{
    int numero;
    raf.seek(0); //nos situamos al principio
    while (true) {
        numero = raf.readInt(); //se lee un entero del fichero
        System.out.println(numero); //se muestra en pantalla
    }
}

public static void leerFichero(){
    RandomAccessFile raf=null;
    try{
        File fichero = new File ("fichero.obj"); //Creo fichero
        raf = new RandomAccessFile(fichero,"r"); //Se lo paso al flujo
        mostrarFichero(raf);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    } finally {
        try {
            if (raf != null) {
                raf.close();
            }
        } catch (Exception e2) {
            System.out.println(e2.getMessage());
        }
    }
}

public static void main(String[] args) {
    crearFichero();
    leerFichero();
}
```

8.- Ficheros de acceso aleatorio

- Ejemplo del método sustituir:



The screenshot shows a Java code editor with the following code:

```
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)

72 public static void sustituirEnteros() {
73     int numero, posicion=-1, i=0;
74     int sustituir, nuevo;
75     Scanner entrada = new Scanner(System.in);
76
77     RandomAccessFile raf=null;
78     try{
79         File fichero = new File ("fichero.dat");
80         raf = new RandomAccessFile(fichero,"rw"); //Lo abro para lectura/escritura
81         raf.seek(0); //nos situamos al principio
82         System.out.println("Introduce el valor que desea sustituir");
83         sustituir=entrada.nextInt();
84         System.out.println("Introduce el nuevo valor");
85         nuevo=entrada.nextInt();
86         while (true) {
87             numero = raf.readInt(); //se lee un entero del fichero
88             if (numero==sustituir){
89                 System.out.println("Encontrado en la posicion "+i);
90                 raf.seek(i*4);
91                 raf.writeInt(nuevo);
92             }
93             i++;
94         }
95     } catch (Exception e) {
96         System.out.println("Fin de la lectura");
97     } finally {
98         try {
99             if (raf != null) {
100                 raf.close();
101             }
102         } catch (Exception e2) {
103         }
104     }
105 }
```

The code implements a method named `sustituirEnteros()` that reads integers from a file named "fichero.dat" and replaces them with a specified value. It uses a `RandomAccessFile` to read and write integers at specific positions. The program prompts the user for the value to search and the new value. It then iterates through the file, reading each integer and writing it back if it matches the search value, effectively replacing it. Finally, it closes the file.

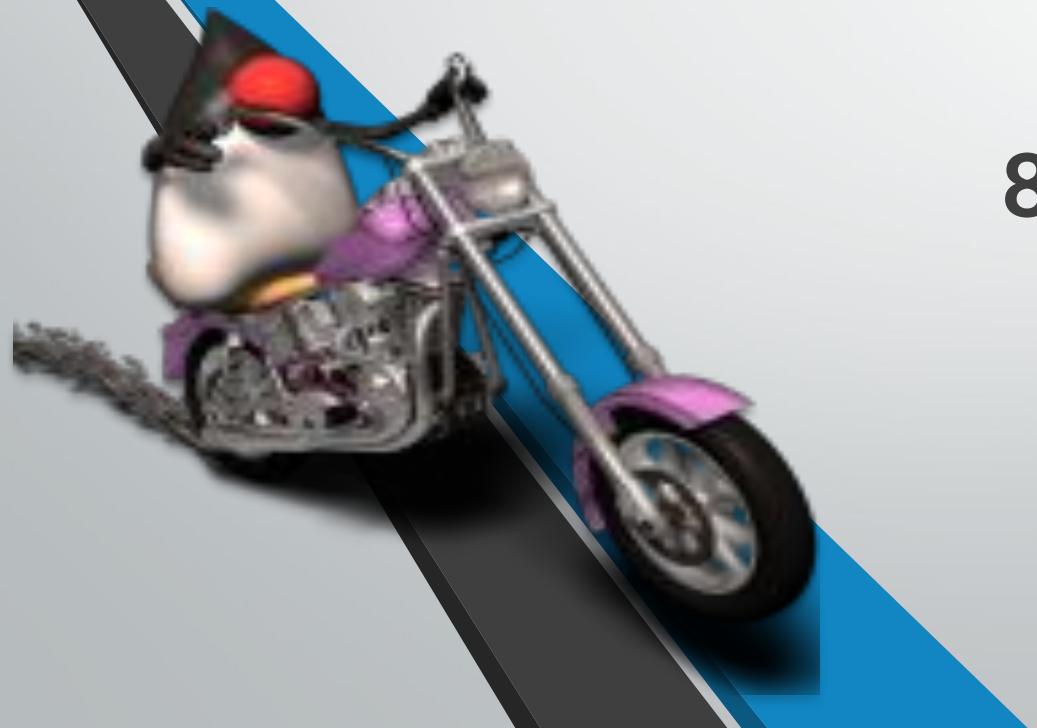
EJERCICIOS

- **Ejercicio 27.- (OBLIGATORIO)** Utilizando un fichero aleatorio, realiza un programa que le muestre al usuario un menú con las siguientes opciones:
 1. Añadir números de tipo double al principio del fichero.
 2. Añadir números de tipo double al final del fichero.
 3. Mostrar el fichero creado.
 4. Sustituir un número indicado por el usuario por otro número que también te indique el usuario.
- Nota: Un double en JAVA ocupa 8 bytes.

JAVA

ENTRADA / SALIDA EN JAVA

8. Ejercicios de consolidación



EJERCICIOS

- **Ejercicio 27.- (OPTATIVO)** Crea un programa que muestre un menú con las opciones "Nuevo cumpleaños", "Consultar cumpleaños" y "Salir".
 - La primera opción pedirá un nombre de una persona y una fecha, y las almacenará en un **fichero de texto** con el siguiente formato: en una línea, el nombre, y en la siguiente línea, la fecha, con el formato día/mes/año. El fichero no se sobrescribirá la siguiente vez que accedas al programa.
 - La segunda opción mostrará un listado con los nombres y fechas de cumpleaños almacenados en el fichero.
 - El menú se mostrará hasta que el usuario seleccione la opción de Salir.
- Implemente el programa haciendo uso de un método llamado `nuevoCumpleaños`, que reciba el nombre del fichero que sirve como agenda, y el cumpleaños a añadir.
- Implemente también un método `muestraCumpleaños`, que reciba como argumento el nombre del fichero que sirve como agenda, y muestre por pantalla todos los cumpleaños albergados en dicha agenda. Cada cumpleaños se mostrará por pantalla de la siguiente manera:
 - El cumpleaños de Pepe es el día 14 de Agosto, nació en el año 1992, por lo que cumplirá 24 años.
 - El cumpleaños de Marta fue el día 11 de Febrero (nació en el año 1994) y cumplió 22 años.

EJERCICIOS

- **Ejercicio 28.- (OPTATIVO)** Crea un programa que muestre un menú con las opciones "Nuevo cumpleaños", "Consultar cumpleaños" y "Salir".
 - La primera opción pedirá un nombre de una persona y una fecha, y las almacenará en un fichero de objetos. El fichero no se sobrescribirá la siguiente vez que accedas al programa.
 - La segunda opción mostrará un listado con los nombres y fechas de cumpleaños almacenados en el fichero.
 - El menú se mostrará hasta que el usuario seleccione la opción de Salir.
- Implemente el programa haciendo uso de un método llamado `nuevoCumpleaños`, que reciba el nombre del fichero que sirve como agenda, y el cumpleaños a añadir.
- Implemente también un método `muestraCumpleaños`, que reciba como argumento el nombre del fichero que sirve como agenda, y muestre por pantalla todos los cumpleaños albergados en dicha agenda. Cada cumpleaños se mostrará por pantalla de la siguiente manera:
 - El cumpleaños de Pepe es el día 14 de Agosto, nació en el año 1992, por lo que cumplirá 24 años.
 - El cumpleaños de Marta fue el día 11 de Febrero (nació en el año 1994) y cumplió 22 años.

EJERCICIOS

- **Ejercicio 29.- (OPTATIVO)** Realiza un programa en JAVA que tenga un método que reciba como argumentos un carácter y el nombre de un fichero de texto. El método ha de leer el fichero de texto cuyo nombre ha recibido, y mostrar por pantalla aquellas líneas del mismo que contengan el carácter indicado en el argumento.
- Ejemplo de ejecución:

Introduzca la letra que desea buscar: q

Introduzca el nombre del fichero donde buscarla: texto.txt

La letra buscada se encuentra en las siguientes líneas:

Línea 3: en donde Don Quijote se encontraba sentado

Línea 11: si hubiera comido el gran queso presentado

EJERCICIOS

- **Ejercicio 30.- (OPTATIVO)** Escriba un programa que cuente las palabras contenidas en un fichero de texto.
- Ejemplo de ejecución:

*Introduzca el nombre del fichero para contar sus palabras: **texto.txt***

El fichero contiene un total de 288 palabras.

EJERCICIOS

- **Ejercicio 31.- (OPTATIVO)** Implemente un programa en JAVA que contenga un método que reciba como argumentos una palabra y un fichero. El método ha de leer el fichero de texto cuyo nombre ha recibido, y mostrar por pantalla el número de ocurrencias en el fichero de la palabra recibida como argumento.
- Ejemplo de ejecución:

*Introduzca la palabra que desea buscar: **guantes***

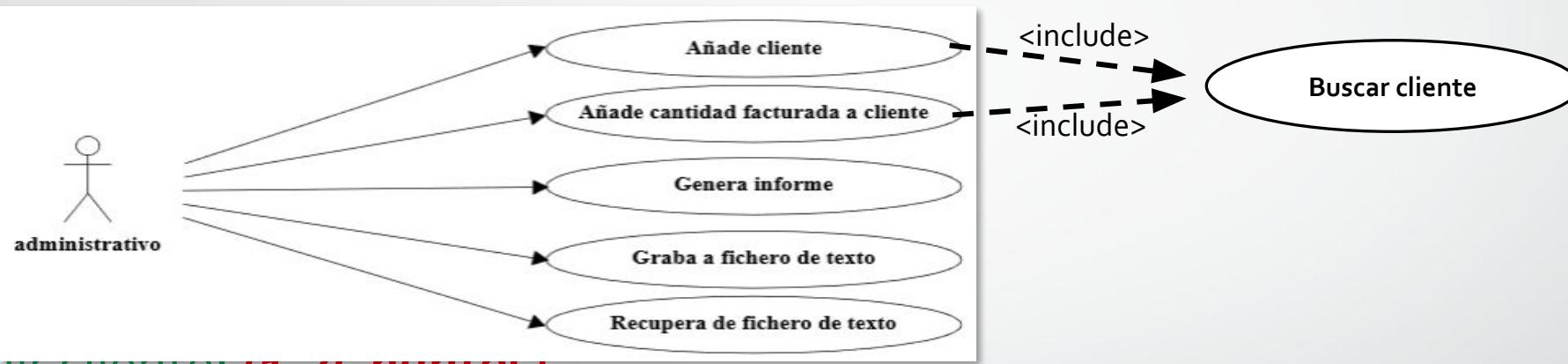
*Introduzca el nombre del fichero donde buscarla: **texto.txt***

*La letra buscada aparece en **5** ocasiones*

EJERCICIOS

- **Ejercicio 32.- (OPTATIVO)** Se desea realizar una aplicación que permita gestionar la deuda de los clientes de una empresa.
- La empresa tiene dos tipos de clientes, los normales y los preferentes.
 - De cada cliente normal se desea guardar su DNI, su nombre y su deuda.
 - Los preferentes heredan los atributos y métodos del cliente normal y además también se guardan los años de antigüedad y el descuento particular que le hacemos frente a su deuda (por ejemplo 5%)
- Las operaciones que se desea que el administrativo pueda realizar son las descritas en los siguientes casos de uso:

EJERCICIOS



• Añadir cliente: (1,75 puntos)

- El administrativo selecciona la opción “Añadir nuevo cliente” e introduce los datos correspondientes a un cliente (DNI y nombre si es un cliente normal; o DNI, nombre, antigüedad y descuento si es un cliente preferente)
- El sistema añade el cliente al fichero correspondiente. Si ya existe un cliente (en cualquiera de los 2 ficheros) con ese DNI se notifica y finaliza el caso de uso.

EJERCICIOS

- Añadir deuda: **(1,75 puntos)**

- El administrativo selecciona la opción “Añadir deuda a cliente” e introduce el DNI y la deuda a añadir.
- El sistema aumenta el total de deuda del cliente sumándole la cantidad introducida. Si no existe un cliente con ese DNI se notifica y finaliza el caso de uso.

- Generar informe: **(1 punto)**

- El administrativo selecciona la opción “Generar informe”.
- El sistema muestra por pantalla, en un formato agradable, la siguiente información de todos los clientes: DNI, nombre y deuda. (Ten en cuenta que los clientes preferentes mostrarán un valor de la deuda a la que le descontaremos el % del descuento)

EJERCICIOS

- Generar fichero de texto: **(2,5 puntos)**
 - El administrativo selecciona la opción “Generar fichero de texto” y a continuación el nombre del fichero deseado.
 - El sistema volcará a un fichero de texto, en un formato agradable, la información de todos los clientes.
- Recupera fichero de texto: **(3 puntos)**
 - El administrativo selecciona la opción “Recupera fichero de texto” y a continuación el nombre del fichero deseado.
 - El sistema sustituirá la cartera de clientes actual por la formada por los clientes cuyos datos están contenidos en el fichero.

EJERCICIOS

• Ejercicio 33.- **(OPTATIVO)**

Implemente un programa en JAVA para que puedan jugar al Trivial 2 amigos. Para ello necesitarás:

- Ficheros (por ejemplo, de texto) para guardar las preguntas y las respuestas.
- Un array de 49 casillas que hará las veces de tablero. Las casillas 0, 7, 14, 21, 28, 35 y 42 proporcionarán al jugador un quesito del color correspondiente si acierta la pregunta.



EJERCICIOS

- Cuando un jugador caiga en una casilla gris volverá a tirar. Si cae en otra de cualquier color se le hará la pregunta correspondiente y si acierta volverá a tirar y si falla pasará el turno.
- *Los colores de las casillas se corresponden a preguntas de las siguientes temáticas:*
 - Azul: Geografía.
 - Naranja: Deportes.
 - Rosa: Literatura.
 - Marrón: Cine.
 - Verde: Ciencias Naturales.
 - Amarillo: Historia.



EJERCICIOS

- Para la resolución del ejercicio se propone la siguiente solución:
- *Clase Jugador: Que contiene los atributos:*
 - *nombre: String*
 - *quesitosGanados: Lista de Strings con los colores*
 - *numeroDeCasillaEnLaQueSeEncuentra: int*
- *También tiene el método lanzarDado que devolverá un numero aleatorio entre 1 y 6.*
- *Clase Tablero: Que contiene un array de 49 Casilla y en cuyo constructor por defecto lo construiremos casilla a casilla.*

EJERCICIOS

- *Clase Casilla: Que contiene los atributos:*

- *color: String*
- *esEspecial: Boolean.*
- *nombreDelFichero: String.*

También tendrá el método obtenerPregunta al que le pasaremos un número aleatorio impar y nos devolverá la pregunta, y el método obtenerRespuesta al que le pasaremos ese número aleatorio + 1 y nos devolverá la respuesta.

A la hora de jugar el jugador lanzará el dado y le mostraremos la información de las dos posibles casillas a las que puede ir (izquierda y derecha) para que elija la que más le interese.

EJERCICIOS

- **Ejercicio 34.- (OBLIGATORIO)** Escribe un programa que gestione una agenda telefónica con ficheros de objetos.
 - Al inicio de la ejecución, nuestro programa cargará todos los contactos del fichero *agenda.obj* en un HashMap para, inmediatamente después, mostrar un menú al usuario con las siguientes opciones:
 1. Añadir contacto (del HashMap).
 2. Eliminar contacto (del HashMap).
 3. Mostrar agenda (mostrará todos los contactos del HashMap).
 4. Buscar contacto por teléfono (lo buscaremos dentro del HashMap).
 5. Salir.
 - Una vez el usuario salga de nuestro programa, debemos volcar todos los contactos al fichero *agenda.obj*.
 - NOTA: De cada contacto almacenaremos el nombre y el número de teléfono, que será único.

EJERCICIOS

- **Ejercicio 35.- (OBLIGATORIO)** Ayuda a un profesor a registrar todas las prácticas que le van enviando los alumnos.
- Para registrarlas, implementaremos la clase *Registro* que almacenará el nombre del alumno, el título de su práctica y la fecha de entrega.
- El programa mostrará al profesor el siguiente menú:
 1. Registrar práctica (escribirá en el fichero *registros.obj* un nuevo registro).
 2. Mostrar prácticas registradas (mostrará todos los registros del fichero).
 3. Buscar registro (Pediremos por pantalla el nombre del alumno y buscaremos si hay un registro en el fichero a su nombre).

EJERCICIOS

- **Ejercicio 36.- (OBLIGATORIO)** La EOI ha abierto el plazo de matriculación. Es por ello que, en los próximos días, recibirá multitud de solicitudes. Solicitudes que debemos gestionar.
- La EOI consta de varias sedes, que almacenaremos en un HashMap. De cada sede almacenaremos su nombre y su identificador (que serán las primeras 3 letras del nombre). Además, por supuesto, de sus solicitudes.
- Cada sede recibirá una serie de solicitudes, que almacenaremos por orden de llegada (para ello, utilizaremos un arrayList). De cada solicitud almacenaremos el nombre y el dni del solicitante.

EJERCICIOS

- Nuestro programa constará de las siguientes opciones:
 1. Añadir sede (sin solicitudes).
 2. Añadir solicitud (primero, pediremos el id de la sede y, si existe, los datos de la solicitud, que insertaremos en la sede correspondiente).
 3. Mostrar sedes junto con solicitudes.
 4. Buscar solicitud (pediremos el dni del solicitante y retornaremos el identificador de la sede donde se encuentra la solicitud. Si no existe, retornaremos un -1).
 5. Mostrar nombre de la sede con mayor número de solicitudes.
 6. Aplicar filtro (escribiremos dos ficheros: *admitidos.txt* y *excluidos.txt*. Todas las solicitudes cuyo dni sea un número par se escribirán en el fichero de admitidos y los impares en el fichero de excluidos. Se escribirán el nombre y el dni del solicitante separados por el carácter).
- NOTA:** Para facilitar la labor del alumno, se da por hecho que no hay ningún nombre de sede que empiece por las tres mismas letras.