UT0 – Repaso Ficheros.

Paquete **java.io** → contiene clases relacionas con la entrada (input) y la salida (output) de datos.

Java se basa en las secuencias de datos para facilitar la entrada y salida.

(Secuencia = corriente de datos entre emisor y receptor. Normalmente son secuencias de bytes pero podemos formatearlos para transmitir cualquier otro tipo de datos.

Stream = corriente de datos en serie, byte a byte. También se usa streams para transmitir caracteres Java (tipo **char** Unicode, de 2 bytes), se habla entonces de **reader** (si es de lectura) o **writer** (si es de escritura).

Excepciones E/S → Son derivadas de **IOException**. Son habituales ya que la entrada y salida de datos es una operación crítica. Por eso, la mayoría de las operaciones van en un try.

Clase File

- Pensada para realizar operaciones de información sobre archivos.
- No proporciona métodos de acceso a los archivos, sino operaciones a nivel del sistema de archivos (listado de archivos, crear directorio, cambiar nombre, borrar fichero…)
- Un objeto File represente un archivo o directorio y sirve para obtener información (tamaño, permisos, etc). También para navegar por la estructura de archivos.
- Si el archivo o carpeta que quiero examinar no existe, la clase File no devuelve una excepción. Luego hay que usar el método **exists**.

File archivo = new File (C:\datos);

InputStream / OutputStream. Corrientes de bytes

Son dos clases abstractas que definen las funciones básicas de r/w de una secuencia de bytes pura, sin estructurar. No representan ni textos ni objetos, son dato binarios puros.

Tienen muchas subclases. Casi todas las de lectura y escritura derivan de ellas.

Los métodos más importantes son **read()** y **write()** → Leen o escriben un byte del dispositivo de entrada.

Reader / Writer. Orientadas a caracteres

Clases abstractas que definen las funciones básicas de r/w basada en Unicode. Pertenecen a la jerarquía de r/w orientada a caracteres.

Métodos **read() y write()** adaptados a leer arrays de caracteres.

InputStreamReader / OutputStreamReader. Adaptar E/S

Clases que sirven para adaptar la E/S porque las correientes básicas de E/S son de tipo Stream. La función de estas clases es adaptarlas a corrientes Reader/Writer.

Derivan de Reader y Writer → ofrecen los mismos métodos.

Contructor: permite crear objetos InputStreamReader pasadn como parámetro un InputStream y objetos OutputStreamWriter a partir de objetos OutputStreamWriter.

DataInputStream / DataOutputStream

Lee datos en forma de byte y los adapta a datos simples (int,short, …, string).

Tienen varios métodos read() y write() para leer y escribir datos de todo tipo.

Construyen objetos a partir de corrientes InputStream y OutpuStream.

ObjectInputStream /ObjectOutputStream

Filtros de secuencia que permiten R/W objetos de una corriente de datos orientada a bytes.

Sólo tiene sentido si los datos almacenados son objetos.

Tienen los mismos métodos que DataInputStream y DataOutputStream, pero añadiendo lo métodos **readObject()** y **writeObject()**.

EL método writeObject puede dar excepciones IOException, NotSerializableException o InvalidClassException.

BufferedInputStream / BufferedOutputStream . BufferedReader / BufferedWriter.

Buffered = capacidad de almacenamiento temporal para R/W → Datos se almacenan en memoria temporal ante de ser leídos o escritos.

Son 4 clases que trabajan con métodos distintos, pero con las mismas corrientes de entrada, que pueden ser:

- Bytes -> InputStream / OutputStream
- Caracteres-> Reader /Writer

La clase BufferedReder incorpora el método **readLine()** → Lee caracteres hasta null o salto de línea.

Archivos Binarios

Los archivos binarios suelen usar las clases DataInpuStream y DataOutputStream, que son adecuadas para escribir todo tipo de datos.

Escribir en archivos binarios

Proceso:

- Crear objeto FileOutputStream a partir de un objeto File
- Crear objeto DataOutputStream asociado al FileOutputStream
- Usar el objeto DataOutputStream creado para escribir los datos con los métodos Write Tipo (donde el tipo es el tipo de datos que queremos escribir (int, doublé, etc). A este método se le pasa como único argumento los datos que queremos escribir.
- Cerrar con close() el objeto DataOutputStream

Ejemplo:

```
File f = new File (“c:\prueba\archivo.out”);
Ramdom r= new Ramdom();
Double d = 3.141592
try {     FileOutputStream fos = new FileOutputStream (f);
          DataOutputStream dos = new DataOutputStream (fos);
          for (int i=0; i<100; i++)
                    dos.writeDouble(r.nextDouble());
          dos.close();
} catch (FileNotFoundException ex) {
          System.out.println (“Error. No encuentro el archive”):
} catch (IOException ex) {
          System.our.println (“Error de escritura”);
}
```

Lectura de archivos binarios.

Tenemos que tener el cuenta el fin de archivo. Cuando llegamos al final, se produce la excepción EOFException (subclase de IOException) y tendremos que controlarla

Ejemplo: Leer el archivo anterior.

```
boolean EndFile = false;
try {
          FileInputStream fis = new FileInputStream (f);
```

```
        DataInptutStream dis = new DataInptutStream (fis);
        double d;
        while (!finArchivo ) {
                d.dis.readDouble();
                System.out.println (d);
        }
        dis.close ();
        } catch (EOFException ex) { finArchivo = true; }
        } catch (FileNotFoundException ex) {
        System.out.println (“Error. No encuentro el archive”):
} catch (IOException ex) {
        System.our.println (“Error de escritura”);
}
```