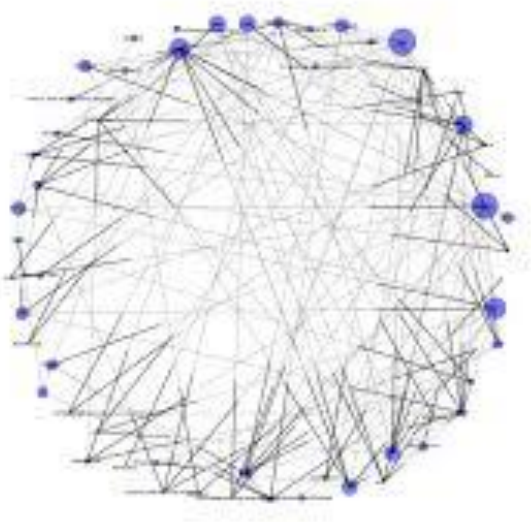
DAM
PROGRAMACIÓN



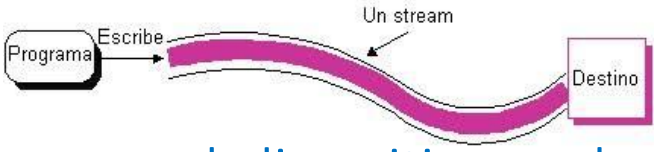
MANEJO DE FICHEROS EN JAVA
FLUJO DE CARACTERES

Introducción

- Hasta ahora, el ciclo de vida de los datos empezaba y finalizaba duranta la ejecución del programa.
- Si queremos que los datos tengan un vida más allá que la de los programas que los generan, deberemos hacer que los datos sean **persistentes**, objetivo que se logrará haciendo **uso de ficheros o bases de datos**.
- *En este UD veremos el uso básico de los archivos de texto en Java para conseguir persistencia de datos*.
- Las clases que usaremos para el tratamiento de ficheros están almacenadas en el paquete **java.io**.
- Cuando trabajamos con ficheros tendremos que tener en cuenta que se pueden producir problemas debido a distintas causas:
  - el archivo puede estar corrupto,
  - el soporte donde está el fichero no es accesible,
  - intentamos guardar información donde no hay espacio libre suficiente, etc
- Las excepciones que tienen en cuenta los problemas con los ficheros derivan de **IOException**.

Streams o flujos de datos

- Un **input file** es un fichero que es leído por un programa.
- Un **output file** es un fichero que es escrito por un programa.
- Las operaciones de entrada y salida a menudo se denominan de **E/S** o **I/O** (Input/Output)
- Un **flujo de datos** es una secuencia ordenada de datos que tiene una fuente (flujos de entrada) o un destino (flujos de salida).
- Las clases que manejan los flujos aíslan a los programadores de los detalles específicos del sistema de funcionamiento del dispositivo y del sistema de entrada y salida.
- Un ***stream*** es un objeto que representa el **flujo de datos** entre nuestra aplicación y una fuente de datos externa, y viceversa.
- Todo programa que necesite tener acceso a información de e/s necesita abrir un ***stream***, así podemos tener dos tipos de flujos:
  - Flujo de datos de entrada: **input stream**.
  - Flujo de datos de salida: **output stream**.

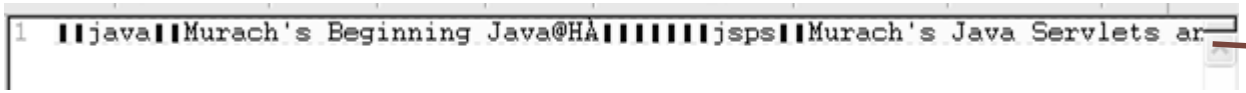


- Los flujos o ***streams*** actúan como interfaz con el dispositivo o clase asociada haciendo la e/s independiente del tipo de datos y del dispositivo.

Tipos de flujos

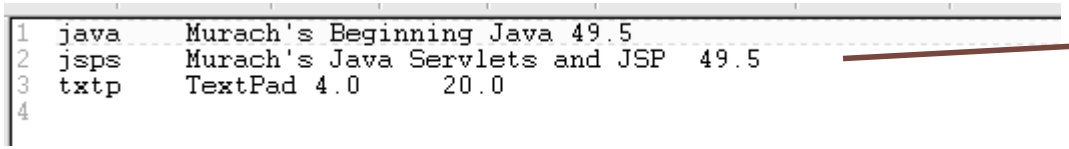
Todo flujo de datos debe ser cerrado al finalizar su uso

- **Flujos de bytes (8 bits):**
  - Se utiliza para leer ficheros binarios.
  - El flujo de datos se realiza **byte** a **byte**.
  - Las clases que trabajan con estos flujos descienden de las clases abstractas **InputStream** y **OutpuStream**.
  - Las clases más comunes son **FileInputStream** y **FileOutputStream** .

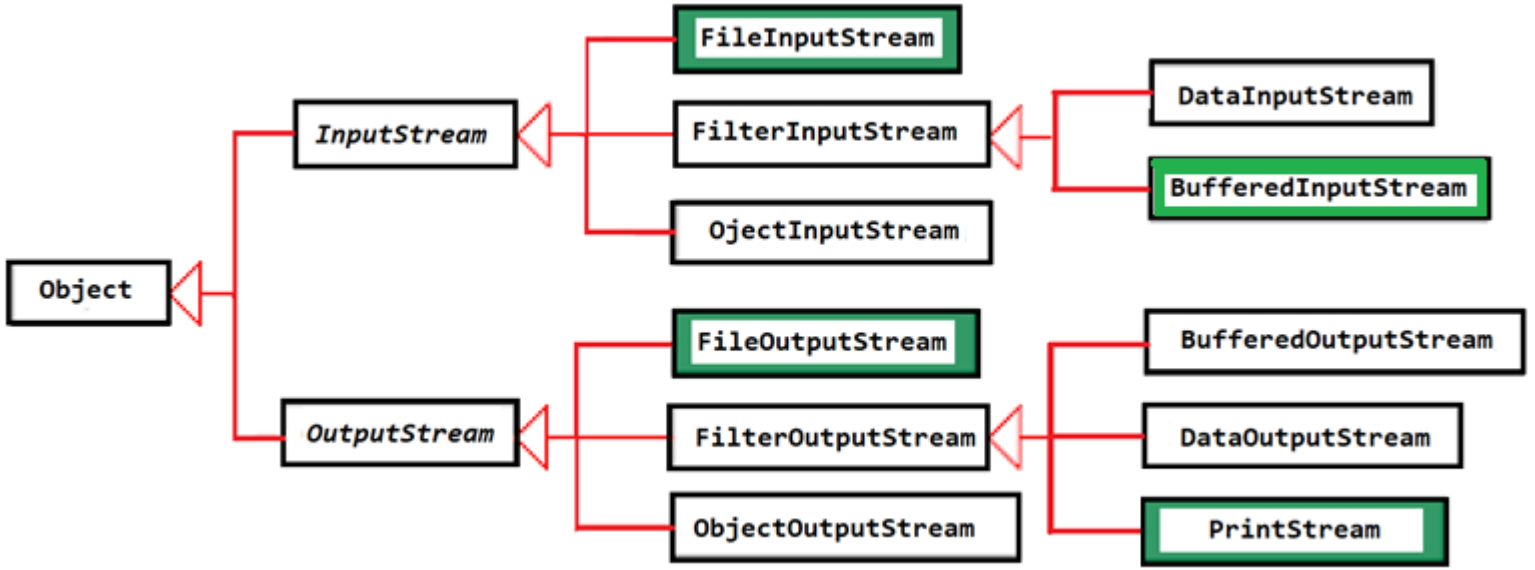


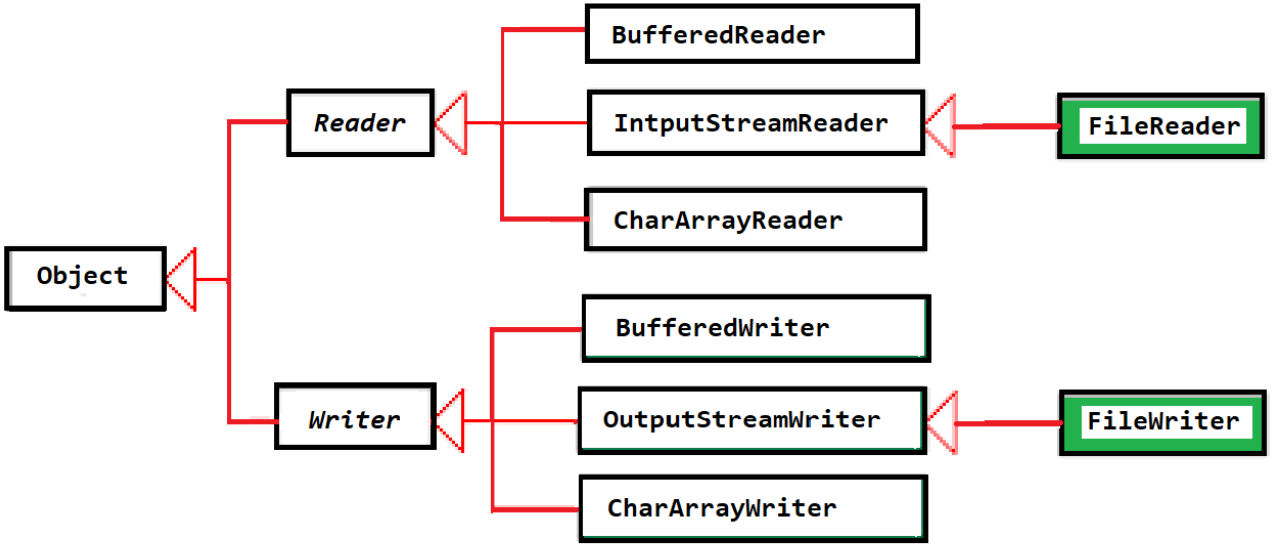
Fichero binario en un editor de texto. Puede contener caracteres así como otros tipos de datos que no pueden ser leídos por un editor de texto.

- **Flujos de caracteres (16 bytes):**
  - Se utilizan para leer ficheros de texto.
  - El flujo de datos se realiza de 16 en 16 bytes, facilitando el trabajo con los caracteres ***Unicode*** que tienen una codificación de 16 bytes.
  - Las clase que trabajan con estos flujos descienden de las clases abstractas **Reader** y **Writer**.
  - Las clases más comunes son **FileReader** y **FileWriter**



Fichero de texto en un editor de texto. Contiene caracteres. Los campos y registros pueden estar delimitados por caracteres especiales como tabulador y saltos de línea


Object
InputStream
FileInputStream
FilterInputStream
DataInputStream
BufferedInputStream
OjectInputStream
OutputStream
FileOutputStream
FilterOutputStream
BufferedOutputStream
DataOutputStream
PrintStream
ObjectOutputStream


Object
Reader
BufferedReader
IntputStreamReader
FileReader
CharArrayReader
Writer
BufferedWriter
OutputStreamWriter
FileWriter
CharArrayWriter

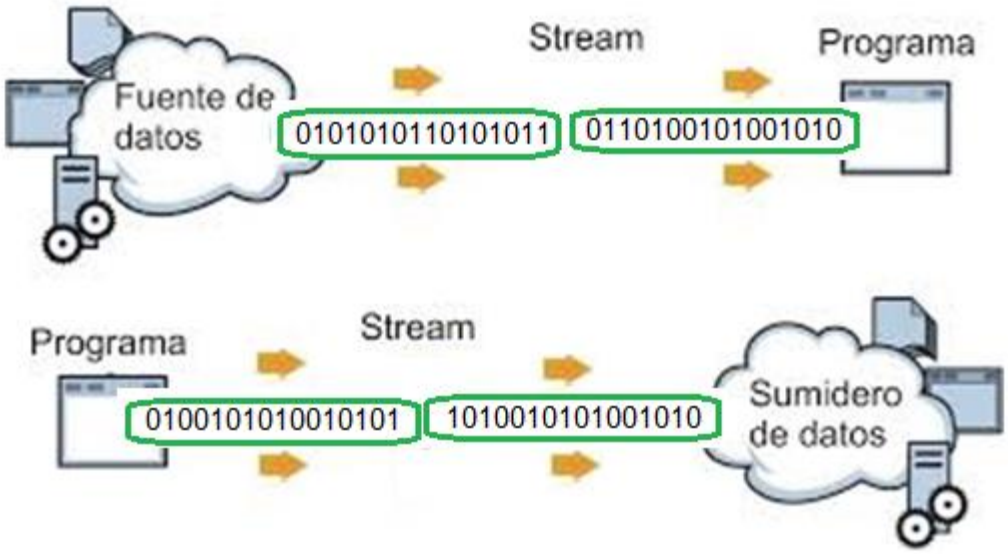
Fujos standar

- **Flujos estándar son flujos de bytes:**
  - **System.in**
    - Instancia de la clase **BufferedInputStream** que a su vez tiene como superclase a la clase abstracta **InputStream**: flujo de bytes de entrada
    - **Métodos**:
      - **read()** → permite leer un byte de la entrada como un entero.
      - **skip(n)** → ignora n bytes de la entrada.
      - **avaliable()** → número de bytes disponibles para leer en la entrada.
  - **System.out**
    - Instancia de la clase **PrintStream**: flujo de bytes de salida.
    - **Métodos**:
      - **print(String), println(String),** muestra por pantalla el **String**.
      - **flush(),** vacía el buffer de salida escribiendo su contenido.
  - **System.err**
    - Instancia de la clase **PrintStream**: flujo de bytes de salida.
    - Funcionamiento similar a **System.out**.
    - Se utiliza para enviar mensajes de error a un fichero *log* o a la pantalla.

- **Ejemplo1 de flujos estándar**:
  - Se lee una línea por teclado carácter a carácter utilizando el flujo estándar de entrada que es producido por el teclado.
  - Se muestra cada carácter leído en una línea.
  - Finaliza cuando introducimos el carácter de fin de línea (‘\n’)

```
static void lecturaTecladoFlujoBinario(){
    int c;
    int contador=0;
    try {
       while ((c=System.in.read())!='\n'){
            contador++;
            System.out.println((char)c);
        }
       System.out.println(”Se han introducido ”+contador+” caracteres”);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

FLUJOS DE CARACTERES



Archivos de Texto

Ficheros de texto

- Los ficheros de texto son aquellos que almacenan caracteres alfanuméricos en formato estándar (ASCII, UNICODE, UTF-8) y normalmente son generados por un editor.
- Para trabajar con ellos usaremos as clases **FileReader** para leer caracteres y **FileWriter** para escribir los caracteres en un fichero.

Reader: flujos de entrada de caracteres

- **Reader** es una clase abstractas que definen las funcionalidades básicas de lectura de **caracteres** *Unicode*.
- Soporta los métodos siguientes:
  - **int read()** → lee un carácter y devuelve el carácter leído pasado a entero devuelve –1 si el fichero se ha acabado.
  - **int read (char[]** *bufer***)** →lee un array de caracteres hasta *bufer.length*. Devuelve el número de caracteres leídos o -1 cuando llega al final del flujo.
  - **int read (char[]** *bufer***, int** *desde***, int** *numChar***)** → lee *numChar* caracteres de *flujo* (o los que pueda) y los coloca a partir de la posición *desde* del búfer.
  - **void close()** → cierra el fichero.

Documentación

FileReader

- La clase **FileReader** tiene como superclase a **InputStreamReader** que a su vez es hija de **Reader**.
- Se utiliza para determinar **un fichero como origen** de un flujo de entrada de caracteres.
- **Constructores**:
  - **FileReader (File *obj*)**

```
File miFile = new File("ruta\\nombreFichero");
Reader miFIS = new FileReader(miFile);
```

En Linux, la ruta utiliza como separador /
carpeta1/fichero.txt
En Windows, la ruta del fichero debe crearse utilizando el doble \\
C:\\carpeta1\fichero.txt

  - **FileReader ("ruta+*nombreFichero* ")**

```
Reader miFIS = new FileReader("ruta\\nombreFichero");
```

Documentación

- **Ejemplo de uso de FileReader:**
  - Se lee un fichero almacenado en la misma carpeta del proyecto *poesía.txt.*
  - El fichero se lee utilizando un búfer en forma de array de caracteres.
  - Los caracteres leídos se almacenan en un *StringBuilder* y se muestran por pantalla.

```
static void leeTodosLosCaracteres() throws IOException {
    //Objeto File con el fichero de texto
    File miFile= new File ("poesia.txt");
    //Objeto para almacenar los caracteres leídos del fichero
    StringBuilder miSB = new StringBuilder();
    //Objeto Reader para leer el fichero de texto en UT-8
    Reader miFileReader =
                    new FileReader(miFile,Charset.forName("UTF-8"));
    //buffer de char para ir leyendo del fichero
    char[] bufer = new char[1024];
   //Leo bufer a bufer hasta el final de fichero
    while (miFileReader.read(bufer)!=-1) {
        //Agrego a la cadena  los caracteres leídos
        miSB.append(bufer);
    }
    //Cierro el fichero
    miFileReader.close();
    //Muestro por pantalla la cadena, resultado de leer el fichero
    System.out.println(miSB.toString());
}
```

Writer: flujos de salida de caracteres

- **Writer** es una clase abstractas que definen las funcionalidades básicas de escritura de caracteres *Unicode* en ficheros de texto.
- Soporta los métodos siguientes:
  - **void write(String *s*)** → escribe en el fichero la cadena *s*.
  - **void write(char [] *bufer*)** → escribe un array de *caracteres* al final del fichero.
  - **void write(char [] *bufer*, int *start*, int *count*)** → escribe un *array bufer* de *caracteres*, empezando en la posición *start* y escribiendo *count* de ellos, deteniéndose antes si encuentra el final del *array*.
  - **void flush()** → vacía el flujo de modo que los *caracteres* que quedaran por escribir son escritos.
  - **void close()** → cierra el flujo de salida liberando los recursos asociados a ese flujo.

Documentación

FileWriter

- La clase **FileWriter** tiene como superclase a **OutputStreamWriter** que a la vez es hija de **Writer**.
- Se utiliza para determinar **un fichero como origen** de un flujo de entrada de caracteres.
- **Constructores**:
  - **FileWriter (File *obj*)**

```
File miFile = new File("ruta\\nombreFichero");
Writer miFOS = new FileWriter (miFile);
```

  - **FileWriter ("*nombreFichero*")**

```
Writer miFOS = new FileWriter ("ruta\\nombreFichero");
```

  - Cualquiera de los constructores puede tener un segundo parámetro *boolean* que en el caso de ser *true*, **añade el flujo al final de fichero**.

Documentación

- **Ejemplo de uso de Writer:**
  - Añade una cadena al final de un fichero existente.

```
static void escribeCaracteres() {
  //Creo un objeto FileWriter a partir de un fichero existente
  //para añadir texto al final
  Writer miFW = null;
   try {
     miFW = new FileWriter(new File("poesia.txt")), true);
     //Determino el texto a añadir
     String cadena = "\n Poema de Rosalía de Castro";
     //Agrego la cadena al final del fichero
     miFW.write(cadena);
   } catch (FileNotFoundException e) {
     e.printStackTrace();
   } finally {
     miFW.close();
   }
}
```

Sentencia try con recursos

- Java 7 incorporó la sentencia ***try con recursos*** con el objetivo de **cerrar** los recursos de forma automática en la sentencia *try-catch-finally* y hacer más simple el código.
- Para ***try con recursos***, el recurso utilizado debe implementar la interfaz **AutoCloseable**..
- CUANDO SE ABRE UN RECURSO EN LA SENTENCIA **TRY**, ÉSTE SE CIERRA AUTOMÁTICAMENTE CUANDO EL BLOQUE **TRY** SE EJECUTA O CUANDO SE PRODUCE UNA EXCEPCIÓN, SIN NECESIDAD DE CERRAR DE FORMA EXPRESA EL RECURSO EN EL BLOQUE **FINALLY**.

```
static void escribeCaracteres()
   //Try con recursos cierra los elementos abiertos de forma automática
   try (Writer miFW = new FileWriter(new File("poesía.txt", true)))
       //Determino el texto a añadir
        String cadena = "\n Poema de Rosalía de Castro";
       //Agrego la cadena al final del fichero
        miFW.write(cadena);
   } catch (FileNotFoundException e) {
            e.printStackTrace();
  }  //No es necesario cerrar los recursos, se cierran automáticamente
```

Buffered….

- Si usamos sólo **FileReader** o **FileWriter**, cada vez que hagamos una **lectura o escritura, se hará físicamente en el soporte de almacenaiento**. Si escribimos o leemos pocos caracteres cada vez, el proceso se hace costoso y lento, con muchos accesos a disco duro.
- Los **BufferedReader**, **BufferedWriter** **añaden un buffer intermedio**. Cuando leamos o escribamos, esta clase controlará los accesos a disco.
- Si vamos escribiendo, se guardará los datos hasta que tenga bastantes datos como para hacer la escritura eficiente.
- Esta forma de trabajar hace los accesos a disco más eficientes y el programa correrá más rápido. La diferencia se notará más cuanto mayor sea el fichero que queremos leer o escribir.

```
BufferedReader mBR = new BufferedReader(new FileReader(fichero))
```

Objeto que accederá al buffer para leer los datos que contiene

Crea un buffer donde guarda los caracteres del fichero en memoria

lee los caracteres del fichero

Documentación

BufferedReader/BufferedWriter

- El objeto **BufferedReader** toma como parámetro un objeto **FileReader**.
- El objeto **BufferedReader** es el que lee los datos de la fuente.
- De forma equivalente funciona el **BufferedWriter**, siendo este objeto el que realiza la operación de escritura

```
static void lecturaEscrituraBufferFlujoTexto_Texto() {
    int dato;
    File miFilesOrigen= new File ("poesia.txt");
    File miFilesDestino= new File("poesiaCopia.txt");
    //Objeto Reader para leer el fichero de texto en UT-8
    try(BufferedReader mBR = new BufferedReader(
            new FileReader(miFilesOrigen,Charset.forName("UTF-8")));
        BufferedWriter mBW= new BufferedWriter(
             new FileWriter(miFilesDestino, Charset.forName("UTF-8")))){
        //Leo dato a dato del buffer hasta el final de fichero
        while ((dato = mBR.read())!=-1) {
             mBW.write(dato); //Escribo en el búfer de salida
        }
    }catch (FileNotFoundException e){
       //Tratar la excepción
    }catch (IOException e){
        //Tratar la excepción
    }
}
```

Utiliza un buffer para leer el fichero y otro para escribir