**Clase #10 – Streams**

**Que son los Streams?:**

Es un medio utilizado para leer datos de una fuente y para escribir datos en un destino.

La fuente como el destino pueden ser archivos, sockets, memoria, cadenas de caracteres y procesos.

Los Streams se caracterizan por ser **unidireccionales**, es decir que un Stream se utiliza solo para leer o solo para escribir. Pero no ambas funciones al mismo tiempo.

Los Streams son canales o flujos de información

**Codigo Netbeans**

****

**Codigo Eclipse:**

****

**Tipos de Streams (Clases Principales):**

*Orientados a Carácter:* Reader- Writer

*Orientados a Byte:* InputStream – OutputStream

Codigo: **Clase\_12\_a**

1. Vamos al **ejercicio 2**. Objetivo: Escribir en un archivo.

Vemos un **File.** Representa un archivo o a un directorio.

1. En **//Instancia un objeto File de salida** colocamos el siguiente codigo:

// Instancia un objeto File de salida

File outputFile = new File("destino.txt");

1. Luego se crea una instancia de un **FileWriter** a partir de un objeto **File**.

1. Luego se coloca un string donde le vamos a poder colocar la información que queramos.
2. Luego se coloca un logaritmo de escritura con un **for.** Carácter a carácter lo voy escribiendo.
3. Ejecutamos.
4. Vemos como se genero el archivo en la misma ruta donde se encuentra mi proyecto: **C:\Users\lrlopez\Documents\NetBeansProjects\Clase\_12\_a**
5. Pero si le agrego la ruta completa hacia nuestro **main** podemos colocarlo en la ruta que queramos. Agregamos:

// Instancia un objeto File de salida

File outputFile = new File("C:\\Users\\lrlopez\\Documents\\NetBeansProjects\\Clase\_12\_a\\src\\main\\java\\com\\educacionit\\jse\\streams\\ejercicio2\\destino.txt");

1. Vamos a **Ejercicio 1.** Objetivo: Leer un archivo.

1. En la linea de //Instancia un objeto File de entrada colocamos el codigo:

// Instancia un objeto File de entrada

File inputFile = new File("destino.txt");

1. Creo un objeto de tipo **File**.
2. Creo un objeto **FileReader** donde le paso como parámetro el objeto **File.**
3. Se utiliza el algoritmo de lectura que dice mientras haya información la voy leyendo.
4. Previamente se genera una variable con el carácter a leer.
5. A cada carácter le voy a asignar el in.read() que cuando me da -1 ya indica que es el fin del archivo.

Vemos de donde viene el método **read().**

<https://docs.oracle.com/javase/7/docs/api/java/io/FileReader.html>

Retorna un numero el cual luego será casteado a carácter para poder ser

mostrado como carácter. O retorna -1 si el fin del stream fue alcanzado.

1. Vamos al **ejercicio 3**. Objetivo: Hacer un **copiador**.
2. Se crean dos streams. Uno de lectura y otro de escritura.
3. Se creara un nuevo stream con la cadena de caracteres copiada.
4. Para poder verlo vamos a la ruta de nuestro proyecto:

**C:\Users\lrlopez\Documents\NetBeansProjects\Clase\_12\_a**

1. O bien incluimos la ruta en los archivos:

File inputFile = new File("C:\\Users\\lrlopez\\Documents\\NetBeansProjects\\Clase\_12\_a\\src\\main\\java\\com\\educacionit\\jse\\streams\\ejercicio2\\destino.txt");

File outputFile = new File("C:\\Users\\lrlopez\\Documents\\NetBeansProjects\\Clase\_12\_a\\src\\main\\java\\com\\educacionit\\jse\\streams\\ejercicio3\\destinoNuevo.txt");

**Superclases de todos los Streams:**

**Clase InputStream (bytes)**

<https://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html>

La diferencia con InputStreamReader es que esta clase lee caracteres y la clase InputStream lee bytes. En ambos casos hay que definir una variable auxiliar de tipo int.

La clase InputStreamReader funciona como un puente entre los byte streams y los char streams. Lee los bytes y los decodifica en caracteres utilizando un charset especifico.

**Clase OutputStream (bytes)**

<https://docs.oracle.com/javase/7/docs/api/java/io/OutputStream.html>

OutputStreamWriter convierte un OutputStream en un Writer. Codifica caracteres en base a bytes.

FileOutputStream es una OutputStream que escribe bytes en un archivo. OutputStreams no aceptan chars (o strings). Utilizando un OutputStreamWriter ahora si tendremos un Writer que acepta chars (o strings).

**Clase FileReader (Uso especifico- para archivos-Char)**

<https://docs.oracle.com/javase/7/docs/api/java/io/FileReader.html>

**Clase FileWriter (Uso especifico para Archivos – Entendible por los humanos - char)**

<https://docs.oracle.com/javase/7/docs/api/java/io/FileWriter.html>

Writers escriben caracteres mientras que OutputStreams escriben bytes especificando como se codifican esos bytes partiendo de ser caracteres.

**Buffers**

Son una alternativa a las clases básicas de entrada y salida. Es decir ir leyendo carácter por carácter no resulta del todo practico. Los buffers son las alternativas que ofrece Java y son subclases de las clases Reader, Writer, InputStream y OutputStream.

Cumplen el mismo objetivo pero son mas eficientes. Porque guardan la información en buffers en lugar de carácter por carácter.

d.Vamos a **Ejercicio 5. Objetivo:** Hacemos un **bufferWriter**

1. Ingresamos el siguiente codigo:

// Instancia un objeto File de salida

File outputFile = new File("ejemploBuffer.txt");

1. Se crean tres líneas.
2. A partir del File outputFile voy a poder crear un BufferedWriter para poder escribir. Sin embargo, toma un FileWriter y lo mejora. Entonces para crear un BufferedWriter tengo que crear primero un FileWriter de la misma manera que lo creábamos antes.

// Construye un BufferedWriter

BufferedWriter writerMejorado = new BufferedWriter(new FileWriter(outputFile));

1. Finalmente mencionamos el método **write()** que me permite escribir no carácter por carácter. Sino **linea** por **linea.**
2. No necesito poner un **/n** como un enter. Sino que tenemos el método **newLine().**
3. Y para que me salga el archivo en la misma carpeta que mi clase reemplazamos el siguiente codigo:

// Instancia un objeto File de salida

File outputFile = new File("C:\\Users\\lrlopez\\Documents\\NetBeansProjects\\Clase\_12\_a\\src\\main\\java\\com\\educacionit\\jse\\streams\\ejercicio5\\ejemploBuffer.txt");

e.Vamos a **Ejercicio 4. Objetivo:** Hacemos un **bufferReader.**

1. Ingresamos el siguiente codigo (creamos un File):

// Instancia un objeto File de entrada

File inputFile = new File("ejemploBuffer.txt");

2.Creamos un BufferedReader. Debo crear un FileReader primero.

// Construye un BufferedReader

BufferedReader readerMejorado = new BufferedReader(new FileReader(inputFile));

3.**eof:** end of file. Se crea un string vacio. En false. Hacemos un while que lo aplique. Con **readLine()** si es el fin del archivo me da un null. Sino me da el contenido de la linea. Entonces el While pregunta si eof es true or false para continuar o no.

4.Lo corremos y vemos como lee el archivo.

f.­­\_**Ejercicio 6. Objetivo:** Obtener un **Stream** de un **archivo** **remoto**. Abrimos una

conexión contra una pagina web y nos podemos traer el codigo fuente de la

pagina HTML.

1. Generamos un objeto de tipo URL.

2. Se crea un BufferedReader. En este caso como creamos un

InputStreamReader se coloca dentro un

url.openConnection().getInputStream().

**openConnection():** Metodo de la clase final **URL**. Devuelve una

conexión contra la URL.

**getInputStream():** Metodo de la clase abstracta **URL Connection**.

3.\_Mientras que no sea el fin de un archivo voy a ir leyendo.

4. Muchas veces los sitios por un tema de seguridad no permiten traer el codigo fuente. Probemos con el siguiente codigo:

// Construye un objeto URL

URL url = new URL("http://www.mercadolibre.com");

g. **Ejercicio 7. Proyecto jse-Tut-12-JavaStreams, ejercicio 7.**

Import=> General => Existing Projects into workspace=> Next => browse =>

Seleccionamos el proyecto y lo importamos.

**Objetivo:** Leer un archivo que se encuentra en un paquete especifico. En este caso el

Archivo esta en una carpeta posterior a la de mi clase **LecturaDesdeClassPath**

1. LEEMOS UN ARCHIVO DE TEXTO UBICADO EN UN CLASSPATH DE NUESTRO PROYECTO,
2. Boton derecho sobre el proyecto ver RUNTIME CLASSPATH

Primero indicamos la ruta del paquete.

**C:\Users\lrlopez\Desktop\jse-Lab-12-JavaStreams - Tutorial\target\classes\streams\recursos**

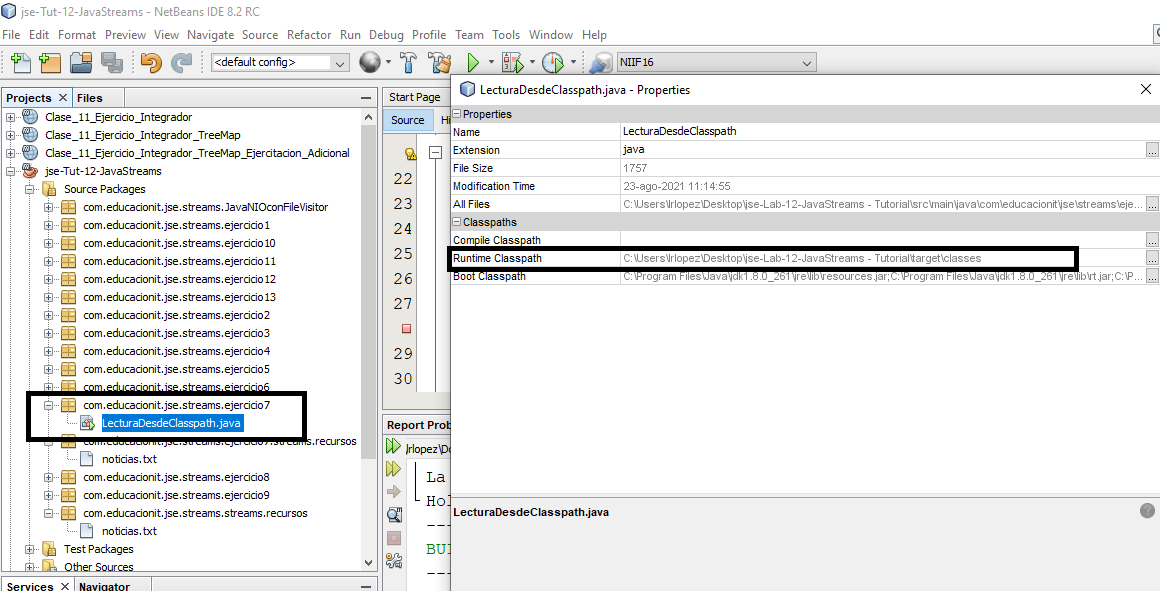
**(reemplazar por la ruta local).**

1. Luego se coloca un InputStreamReader. Para crearlo se le hace a la clase donde estoy en cuestión**.class.getResourceAsStream(ArchivoALeer));**
2. El resto ya lo conocemos. Ejecutamos.
3. Obtenemos por consola el contenido de ese archivo.

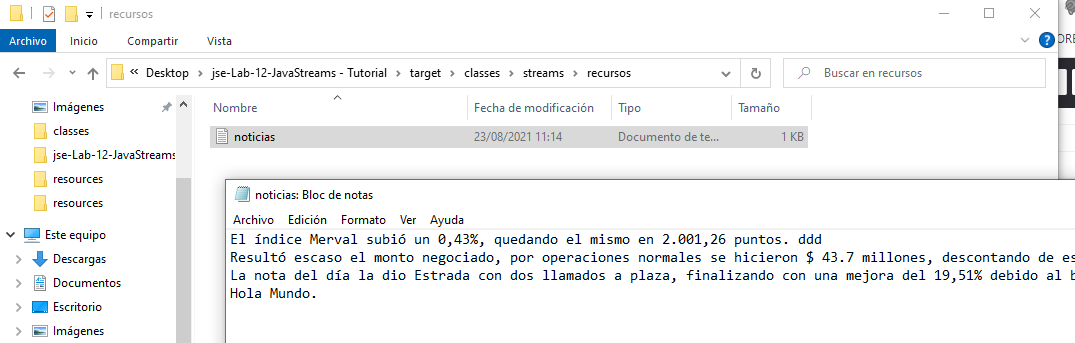
En este caso para buscar el archivo **noticias.txt** se esta ubicando desde el classpath el cual es:

C:\Users\lrlopez\Desktop\jse-Lab-12-JavaStreams - Tutorial\target\classes

A partir del mismo se creo el directorio **com** en donde están todos los códigos y el directorio **streams** dentro del cual se coloco **noticias.txt.** para detectar cual es el **classpath** de nuestro proyecto hacemos botón derecho sobre la clase y lo vemos en la siguiente opción:



Por eso el archivo se encuentra en mi caso en:



**JAVA NIO**

A diferencia de los anteriores, trabaja en forma BIDIRECCIONAL. Puedo utilizar un bloque de codigo para leer y para escribir. Son mas performantes porque trabajan en bloques.

h. **Ejercicio 8. Proyecto Clase\_12\_a,** ejercicio 8**.**

1. Vamos a la clase **FilesNIOExamples.**

2. Para ver información sobre LinkOption del método exists() podemos

ver:

<https://docs.oracle.com/javase/7/docs/api/java/nio/file/class-use/LinkOption.html>

Nos permite saber si existe un path determinado.

<https://docs.oracle.com/javase/7/docs/api/java/nio/file/LinkOption.html>

LinkOption es un Enum que maneja links simbólicos (path) para determinar como son manejados.

1. **CREAMOS un directorio**: Descomentamos estas dos líneas y vemos como se crea el directorio en la ruta de nuestro Proyecto.

C:\Users\lrlopez\Desktop\JAVA STANDARD PROGRAMMING\CLASE 12\jse-Lab-12-JavaStreams - Tutorial1

System.out.println("Creamos un directorio");

createDirectory();

Vemos como podemos especificarle cualquier ruta para crear el directorio.

Por ejemplo: Agrego la siguiente url al método **createDirectory:**

Path path = Paths.get("C:\\Users\\lrlopez\\Documents\\data2");

La clase **Files** tiene varios métodos útiles que son estáticos.

1. **BUSCAR UN ARCHIVO:**
2. Vemos si existe un archivo llamado “logging properties”.
3. Si no existe cambiarle el nombre a “**noticias (txt)”.** Ubicado en la misma ruta de mi proyecto.

C:\Users\lrlopez\Desktop\JAVA STANDARD PROGRAMMING\CLASE 12\jse-

Lab-12-JavaStreams - Tutorial1

Un **symlink** (también llamado enlace simbólico) es un tipo de archivo en Linux que apunta a otro archivo o una carpeta en tu computador. Los enlaces simbólicos son similares a los accesos directos en Windows. Por lo tanto el hecho de que el codigo tenga NO\_FOLLOW\_LINKS significa que no tomara en cuenta los accesos directos en Windows que tengan el mismo nombre.

1. **COPIAR UN ARCHIVO:**
2. Tomamos en consideración el archivo “noticias” que se encuentra en nuestra ruta del proyecto.
3. Descomentamos **copyFile().**
4. Colocamos el siguiente codigo en el método:

Path sourcePath = Paths.get("noticias.txt");

Path destinationPath = Paths.get("noticias1.txt")

1. Vemos como se copio un nuevo archivo.
2. **MOVER UN ARCHIVO:**
3. Descomentamos MoveFile().
4. Insertamos el siguente codigo en el método.

Path sourcePath = Paths.get("noticias.txt");

Path destinationPath = Paths.get("data/noticias.txt");

1. Y después hacemos el camino inverso:

Path sourcePath = Paths.get("data/noticias.txt");

Path destinationPath = Paths.get("noticias.txt");

1. **ELIMINAR UN ARCHIVO:**
2. Descomentamos deleteFile().
3. Ingresamos el siguiente codigo en el método.

Path path = Paths.get("noticias.txt");

**FILE VISITOR**

1. **FILE VISITOR:** Es una interfaz que me va a permitir trabajar con el **WalkFileTree()** método que (pertenece a una clase final llamada Files). Cualquier clase que implemente la interfaz FILEVISITOR tendrá que colocar inteligencia a 4 metodos. Esto dispara EVENTOS. Yo voy a estar manejando cada EVENTO.

Lo que hace FileVisitor es permitir recorrer una ruta de directorios. Implementa los 4 metodos de la interfaz.

1. Abro las clases **Visitador** y **WalKFileTreeExample.**
2. Nos posicionamos sobre **WalkFileTreeExample** y **RUN.**
3. Vemos con en la consola se van listando los directorios del path indicado en el main.

***EJERCICIO combinado FILE VISITOR con JAVA NIO:***

1. Vamos a investigar sobre el directorio **target** ubicado en mi caso en:

C:\Users\lrlopez\Desktop\jse-Lab-12-JavaStreams - Tutorial\target\classes (reemplazar cada uno con su ruta correspondiente)

1. Pegar los siguientes archivos en las siguientes rutas



En C:\Users\lrlopez\Desktop\jse-Lab-12-JavaStreams - Tutorial\target\classes\com



En C:\Users\lrlopez\Desktop\jse-Lab-12-JavaStreams - Tutorial\target\classes\com

1. Luego aplicar el codigo del paquete **com.educacionit.jse.streams.Java.NIOconFileVisitor.**
2. **Resultado:** Veremos como podemos no solo visitar y mostrar sino que también borrar o interactuar con archivos y directorios.

i.-**EJERCICIO 9:** Objetivo: hacer un FileReader para leer un archivo

1. Vamos al ejercicio 9 del proyecto actual.

2. Generamos un reader.

j. **EJERCICIO 10:** Objetivo: hacer un FileWriter para escribir en un archivo.

1. Vamos al ej 10 del proyecto.

2. Generamos un FileWriter.

3. Vemos como se genera el archivo en la ruta correspondiente.

k. **EJERCICIO 11: Objetivo:** Codificar el copiador mediante **FileInputStream** y

**FileOutputStream.** Es un ejemplo de como se trabaja con archivos binarios.

El archivo “fuente.gif” debe estar ubicado en la raíz del proyecto.

1. Vemos el archivo fuente.gif ubicado en la ruta de nuestro Proyecto.
2. Ejecutamos el codigo.
3. Vemos como genera el archivo destino.

I.­\_**BUFFERS:**

La ventaja es que logran una mejora sustancial en la lectura/escritura. Son utilizados como Wrappers (envoltorios) para envolver las clases básicas.

**EJERCICIO 12: Objetivo:** Hacer un BufferReader.

1. Al igual que practicamos con el LectorMejorado hacemos otro BufferedReader pero sin recorrer todo el archivo.

El FileReader queda envuelto en el BufferedReader. El readLine() lee y retorna un

String, a diferencia del read() que retorna un solo carácter.

**EJERCICIO 13: Objetivo:** Hacer un BufferWriter.

1. Al igual que practicamos sobre el EscritorMejorado hacemos un Buffered Writer.

**BufferedWriter** hereda de Writer y se utiliza para envolver a otras subclases del tipo Writer como FileWriter.

Metodos: **write(String s, int offset, int length)** permite para escribir un conjunto de caracteres en lugar de leer carácter a carácter.

**newLine():** Enter.

Vemos como genera el archivo **destino.txt.**