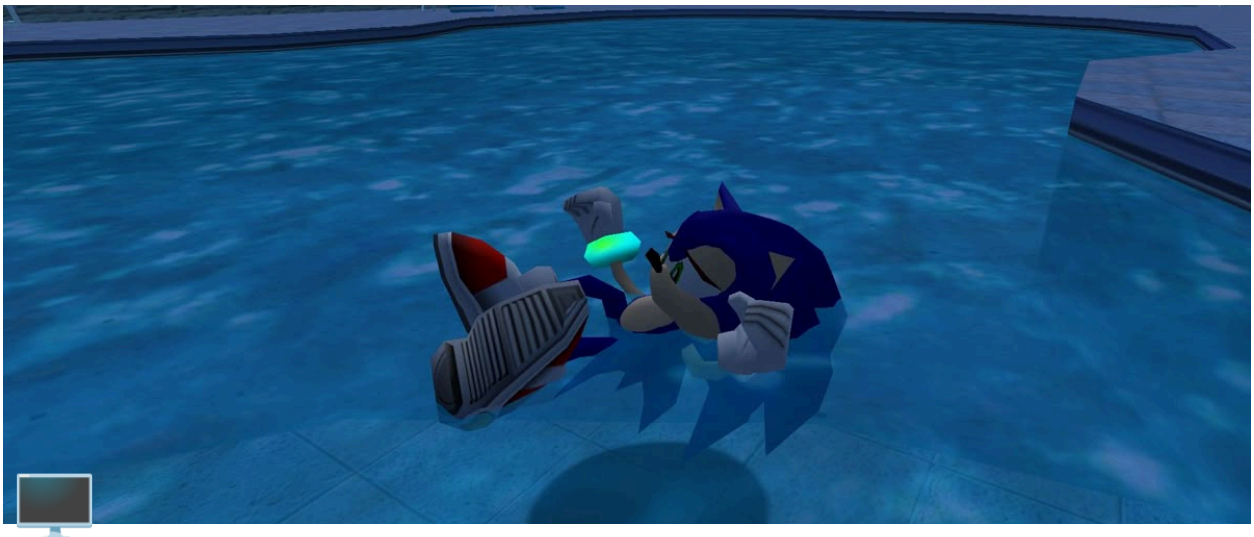


ACTIVIDAD 1.1 - Investigación sobre Ficheros y Tratamiento I

El objetivo de la actividad es realizar labores de investigación y tratamiento de un fichero con extensión txt haciendo uso de las clases del paquete java.io de Java:

▼ A. Investigación sobre clases del paquete java.io

¿Para qué se utilizan las siguientes clases?

- **File**: en Java, la clase File forma parte del paquete `java.io` y se utiliza para representar nombres de ruta de archivos y directorios de forma abstracta. Proporciona métodos para crear, eliminar e inspeccionar archivos y directorios, lo que lo convierte en un concepto fundamental para el manejo de archivos en las aplicaciones Java. Se utiliza principalmente para realizar operaciones con archivos y directorios, como comprobar si un archivo existe, crear archivos o directorios nuevos y recuperar propiedades de archivos.

```
File file = new File("path/to/file");
```

- **FileReader:** se utiliza principalmente en Java para leer datos basados en caracteres desde un archivo de texto de forma directa y sencilla. Suele combinarse con un BufferedReader para almacenar los datos en búfer y leer líneas completas de manera mucho más eficiente.

```
FileReader fr = new FileReader("ruta/al/archivo.txt");
```

- **BufferedReader:** es una clase con la que se puede leer texto desde un flujo de entrada, como por ejemplo desde el teclado o un archivo. Lee bloques grandes de datos a la vez. Es como un

"intermediario" entre el flujo de entrada y el código. Almacena temporalmente los datos que va leyendo, para que cuando los necesitemos podamos obtenerlos más rápido.

```
BufferedReader br = new BufferedReader (new FileReader("ficheros/nombres.txt"));
```

- **FileWriter:** permite escribir caracteres en archivos, lo que es una herramienta esencial para almacenar datos, configuraciones, entre otros.

```
FileWriter writer = new FileWriter ("hola.txt");
```

- **BufferedWriter:** es una clase que se utiliza para escribir texto en un flujo de salida, como puede ser un archivo o la consola. Es parecido a BufferedReader (que leemos en el flujo de entrada), solo que el BufferedWriter se asegura de que los datos se escriban en bloques grandes, para mejorar la velocidad y reducir la cantidad de operaciones de escritura. No escribe los datos directamente al destino, sino que primero los almacena en un búfer, y luego los escribe en bloque.

```
BufferedWriter escritor = new BufferedWriter(archivo);
```

- **PrintWriter:** es una clase que se usa para escribir cualquier forma de datos, por ejemplo int, float, double, en forma de texto, ya sea en la consola o en un archivo en Java.

¿Qué diferencia hay entre FileReader y BufferedReader? ¿Cuál es más eficiente en el tratamiento de información? ¿Por qué?

La diferencia principal está en **cómo acceden a los datos** y cómo gestionan la memoria durante la lectura.

- **FileReader:** se conecta directamente al archivo físico en el disco duro. Cada vez que el programa ejecuta una instrucción de lectura, `FileReader` realiza una operación de entrada/salida directa con el SO.
- **BufferedReader:** actúa como un intermediario. No lee directamente el archivo, sino que toma un `Reader` (como `FileReader`) y crea un bloque de memoria interna (el búfer). Lee grandes fragmentos de texto del disco de una sola vez, los guarda en su búfer en la RAM y va entregando los datos al programa desde allí.

BufferedReader es más eficiente en la mayoría de los escenarios. En la práctica, `FileReader` casi nunca se usa solo, sino que siempre se envuelve dentro de un `BufferedReader`.

El acceso al almacenamiento físico (disco duro) es una de las operaciones más lentas para un procesador, mientras que acceder a la memoria RAM es casi instantáneo. Si queremos leer un archivo de texto de 8.000 caracteres:

- Usando solo `FileReader`, el programa realiza **8.000 peticiones de hardware** al disco.

- Usando `BufferedReader`, hace 1 sola petición al disco para cargar un bloque de 8.000 caracteres en su búfer interno de memoria. Las 7.999 restantes que hace el programa se responden instantáneamente desde la RAM.

¿Qué diferencia hay entre FileWriter, BufferedWriter y PrintWriter?

- **Nivel de conexión:** `FileWriter` es el motor básico y el único que se conecta directamente al archivo físico del disco. Las otras dos son clases "accesorias" que lo envuelven para añadirle mejoras.
- **Velocidad (uso de memoria):** la escritura básica envía datos al disco constanemente, lo cual es muy lento. Añadir `BufferedWriter` crea un almacenamiento temporal en la memoria RAM para agrupar los textos y escribirlos todos de golpe, multiplicando el rendimiento.
- **Comodidad:** las clases de bajo nivel obligan a convertir cualquier dato a texto manualmente y a lidiar con advertencias de error constantes. En cambio, `PrintWriter` traduce números y variables a texto automáticamente, incluye métodos cómodos como `println()` y absorbe los fallos para que el código quede mucho más limpio.

Entre FileWriter y PrintWritter, ¿cuál de las dos clases lanza excepciones de tipo IOException? ¿Cómo gestionará los errores la otra clase? ¿Cuál es las dos clases usaría spara añadir/escribir texto al final de un fichero?

`FileWriter` lanza IOException. Tanto al abrir el archivo como al usar sus métodos de escritura, te obliga a capturar los errores usando `try-catch` o a declararlos en la firma del método.

Los errores los gestiona silenciándolos durante la escritura. Si algo falla, el programa no se detiene ni lanza excepciones; en su lugar, activa una advertencia interna. Para saber si la escritura falló, debemos invocar manualmente su método `checkError()`, el cual devolverá `true` si ocurrió algún problema.

Para añadir texto al final debemos usar `FileWriter`. Es esta clase la que posee la capacidad de abrir un archivo en modo "append" (añadir) mediante su constructor, pasándole `true` como segundo parámetro: `new FileWriter ("archivo.txt", true))`

▼ B. Tratamiento de un fichero con extensión "txt"

Dado el fichero **informacion_paises.txt**, situado en el directorio que se describe en la **Nota 2**, se pide crear un proyecto **Ficheros_01** JAVA con un programa que recorra el fichero, tratando únicamente la información referente a los ciclos, imprimiendo por consola literales con la información que aparece abajo:

Ejemplo: `INSERT INTO T_CICLO VALUES ('IFC01', 'Título Profesional Básico en Informático de Oficina', 2000, 'IFC');`

Nota 1: a partir de ahora, si no se indica lo contrario, el criterio a utilizar, el paquete JAVA, de trabajo para cada actividad será: **org.palomafp.aadd.rmr**

Nota 2: a la hora de colocar el/los fichero/s de las actividades, el criterio será situarlos en un directorio llamado **c:\RAIZ_TU_USUARIO/aadd/ficY**

Nota 3: en la entrega se incluirá un enlace a un repositori odentro de la plataforma basada en Gitea:

- Nombre repositorio: **aadd_fic01**

- Repositorio: **público**
- El repositorio contendrá:
  - El código del proyecto JAVA
  - El documento PDF con la información resultado de la investigación.

```
package org.palomafp.aadd.rmr;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class Main {

    public static void main(String[] args) {

        try {

            BufferedReader br = new BufferedReader (new FileReader("C:/User
s/Rocío/aadd/fic01/informacion_paises.txt"));
            String linea = "";

            while ((linea = br.readLine()) != null) {
                if (linea.startsWith("#")) {

                } else {
                    String lineabien = linea.replace('\"', '\'');
                    System.out.println("INSERT INTO T_PAIS VALUES (" + line
abien + ");");
                }
            }
            br.close();
        } catch (FileNotFoundException e) {
            System.out.println("No se encuentra el fichero información_");
        } catch (IOException e) {
            System.err.println("No se puede leer el fichero");
        }

    }

}
```

- Se utiliza `FileReader` para acceder al archivo en la ruta indicada. Lo he envuelto en un `BufferedReader` porque permite leer el archivo línea a línea de forma mucho más rápida y eficiente.
- El `while` recorre el documento hasta que se acaban las líneas (`!= null`).
- El `if` comprueba si la línea empieza por `#`. Si es así, se la salta.
- Se reemplazan las comillas dobles por simples, necesario porque SQL utilzia comillas simples para los campos de texto.
- Finalmente, se concatena el texto limpio con el INSERT y se muestra el resultado final