UD08 - Apuntes - Ficheros. Flujos de entrada y salida. Regex

Excepciones:

Antes de comenzar a hablar de ficheros, es necesario hablar del concepto de Excepción, que lo veremos muy por encima, ya que posteriormente tenemos una unidad didáctica destinada a ellas.

En Java los errores en tiempo de ejecución (cuando se está ejecutando el programa) se denominan excepciones, y esto ocurre cuando se produce un error en alguna de las instrucciones de nuestro programa, como por ejemplo cuando se hace una división entre cero, cuando un objeto es 'null' y no puede serlo, cuando no se abre correctamente un fichero, etc. Cuando se produce una excepción se muestra en la pantalla un mensaje de error y finaliza la ejecución del programa.

Java nos permite hacer un control de las excepciones para que nuestro programa no se pare inesperadamente y aunque se produzca una excepción, nuestro programa siga su ejecución. Para ello tenemos la estructura "try – catch – finally" que la mostramos a continuación:

```
try {

    // Instrucciones cuando no hay una excepción

} catch (TypeException ex) {

    // Instrucciones cuando se produce una excepcion

} finally {

    // Instruciones que se ejecutan, tanto si hay como sino hay
excepciones

}
```

En las operaciones con ficheros pueden surgir una serie de problemas, algunos son:

• Intentar leer/borrar/escribir sobre un fichero que no existe

• Intentar leer/borrar/escribir sobre un fichero que ya está siendo modificado por otro programa

• Intentar escribir o leer un fichero cuando el puntero haya llegado al final del fichero (EOF)

Archivos:

Un archivo es un conjunto de datos estructurados guardados en algún medio de almacenamiento que pueden ser utilizados por aplicaciones.

Está compuesto por:

- Nombre: Identificación del archivo.
- Extensión: Indica el tipo de archivo (no obligatorio).

Tipos de ficheros:

- Fichero estándar: Fichero que contiene cualquier tipo de datos. Documentos, imágenes, audio, vídeo, etc.
- Directorio o carpeta: Fichero que contiene otros ficheros. Sirve para organizar de forma jerárquica los diferentes ficheros
- Ficheros especiales: Ficheros que sirven para controlar los diferentes periféricos conectados al ordenador

Rutas:

Para acceder a un determinado fichero, se utiliza la ruta (path)

Una ruta indica la dirección del fichero en el sistema de archivos

En una ruta, cada nivel de la jerarquía se representa delimitado por el símbolo /. En Windows, el símbolo separador es \

Además de /, existen 2 elementos especiales en la ruta

- **.** Representa al directorio actual
- **..** Representa al directorio padre en la jerarquía

Existen 2 tipos de rutas:

- Absoluta: Ruta al fichero desde el directorio principal (root). Ej: /home/Documentos/ejemplo.txt
- Relativa: Ruta al fichero desde el directorio actual. Ej: Estando en home, ./Documentos/ejemplo.txt

Creación de directorios en Java:

En Java se puede crear una carpeta o directorio con la ayuda del método mkdir y te permite crear uno o más directorios, veamos un ejemplo:

```
public static void main(String args[]){
    File directorio = new File("/ruta/directorio_nuevo");
    if (!directorio.exists()) {
        if (directorio.mkdirs()) {
            System.out.println("Directorio creado");
        } else {
            System.out.println("Error al crear directorio");
        }
    }
}
```

Operaciones con ficheros:

- Apertura: El programa abre el fichero y se prepara para leerlo o escribirlo. Suele "reservar" el fichero para sí
- Cierre: Indica que se ha finalizado con las operaciones sobre el fichero. Libera el fichero
- Lectura: Lee el fichero o una de sus partes
- Escritura: Permite escribir en el fichero, ya sea añadiendo datos o sobrescribiendo los ya existentes

- Ejecución: Similar a la lectura, pero utiliza los datos del fichero para ejecutar un software
- Creación: Crea un nuevo fichero con un nombre, extensión y ruta determinados
- Eliminación: Elimina un fichero determinado

Ficheros en Java:

La clase que manipula ficheros en Java es la clase File.

Con esta clase se pueden hacer un gran número de operaciones sobre un fichero y sus propiedades, pero no se permite leer ni escribir

También permite obtener datos del fichero, como rutas, nombres, permisos e incluso si existe

El resto de clases que manipulan ficheros parten de la existencia de una clase File, por lo que es la base de cualquier operación de manipulación de ficheros

Ejemplo de uso:

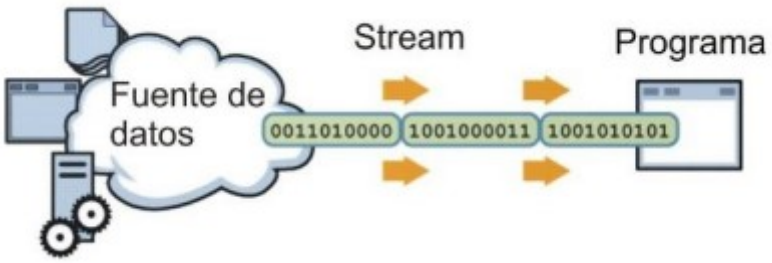
```
public static void main(String args[]) {
 File fichero = new File("FicheroEjemplo.txt");

 if (fichero.exists()) {
 System.out.println("Nombre del archivo "+ fichero.getName());
 System.out.println("Ruta "+ fichero.getPath());
 System.out.println("Ruta absoluta "+ fichero.getAbsolutePath());
 System.out.println("Se puede escribir "+fichero.canRead());
 System.out.println("Se puede leer "+fichero.canWrite());
 System.out.println("Tamaño "+fichero.length());
 }

 fichero.close();
}
```

Stream de datos:

Para poder realizar las operaciones de lectura y escritura de ficheros, Java establece lo que se conoce como un Stream de datos

Crea una vía de comunicación entre programa y fichero que permite "moverse" por las distintas partes del fichero

Existe un puntero que apunta a las partes del fichero



Para leer datos de un fichero de texto, utilizaremos las siguientes clases:

- Clase File: Para representar el fichero que se quiere leer
- Clase FileReader: Establece el stream de datos de lectura del fichero. Tiene una serie de métodos para leer carácter a carácter. Al constructor del FileReader recibe el objeto File
- Clase BufferedReader: Crea un buffer a través del FileReader, que permite leer mas de un carácter. El constructor recibe el FileReader como parámetro

Veamos un ejemplo:

```
import java.io.*;
class LeeFichero {
    public static void main(String [] arg) {
        File archivo = null;
        FileReader reader = null;
        BufferedReader buffer = null;
        try {
            archivo = new File("C:\\directorioArchivo\\archivo.txt");
            reader = new FileReader (archivo);
            buffer = new BufferedReader(reader);
            String linea;
            while( (linea=buffer.readLine()) != null) {
                System.out.println(linea);
            }
        } catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if( null != fr ){
                    fr.close();
                }
            }catch (Exception e2){
                e2.printStackTrace();
            }
        }
    }
}
```

Escritura de un fichero:

Para escribir datos en un fichero de texto, utilizaremos las siguientes clases:

- Clase File: Para representar el fichero que se quiere leer
- Clase FileWriter: Establece el stream de datos de escritura del fichero. Tiene una serie de métodos para escribir en ficheros. Al constructor del FileWriter recibe el objeto File
- Clase PrintWriter: Crea un buffer a través del FileWriter, que permite extender los métodos del FileWriter por otros similares a los que tenemos en la salida de pantalla. El constructor recibe el FileWriter como parámetro

Entre las funciones que tenemos en el PrintWriter, las más comunes son:

- print(“texto”). Imprime el texto pasado por parámetro
- println(“texto”). Imprime el texto pasado por parámetro y hace un salto de línea

El constructor del FileWriter puede recibir un segundo parámetro boolean que indica si queremos escribir el fichero desde cero (false) o si queremos añadir texto al existente (true)

```
FileWriter writer = new FileWriter(fichero);
FileWriter writer = new FileWriter(fichero, false);
FileWriter writer = new FileWriter(fichero, true);
```

Veamos un ejemplo:

```
import java.io.*;
public class EscribirFichero
{
    public static void main(String[] args)
    {
        File fichero = null;
        FileWriter writer = null;
        PrintWriter pw = null;
        try
        {
            fichero = new File("C:\\directorioArchivo\\archivo.txt");
            writer = new FileWriter(fichero);
            pw = new PrintWriter(writer);
            for (int i = 0; i < 10; i++) {
                pw.println("Linea " + i);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (null != fichero) {
                    fichero.close();
                }
            } catch (Exception e2) {
                e2.printStackTrace();
            }
        }
    }
}
```

Borrado de un fichero:

Para borrar un fichero, se utiliza la clase File

- Existe la función delete() que elimina el fichero

Veamos un ejemplo:

```
public static void main(String args[]) {
        File fichero = new File("FicheroEjemplo.txt");

        if (fichero.exists()) {
            System.out.println("Nombre del archivo "+
fichero.getName());
            System.out.println("Ruta "+ fichero.getPath());
                    System.out.println("Ruta absoluta "+
fichero.getAbsolutePath());
                            System.out.println("Se puede escribir
"+fichero.canRead());
```

```
            System.out.println("Se puede leer "+fichero.canWrite());
            System.out.println("Tamaño "+fichero.length());
        }

        fichero.delete();
    }
```