índice

Tema 1 Introducción al manejo de ficheros

Ficheros y tipos de ficheros

Fichero: sucesión de bits almacenados en un dispositivo. Formado por un nombre y una extensión.

- **De texto (ASCII):** ficheros al abrirlos con el bloc de notas se pueden leer, txt, lenguajes de programación. Están en código ASCII
- **Binarios:** ficheros que están en código binario. Son los .Bin .dat

La clase File (Java.io.File)

crear, modificar y eliminar ficheros en java

Formas de acceder a un fichero

SE DEBE USAR TRY-CATCH para el manejo de errores. Al usar try-catch no es estrictamente necesario cerrar los flujos ya se cierran al terminar los try

- Acceso Secuencial:
  Secuencia de caracteres o bytes para acceder a uno determinado hay que pasar por los anteriores
  - **Caracteres:** se suele usar con BufferedReader y BufferedWriter (Mejora el rendimiento)
    - **FileReader** (entrada)
      - **read():** lee un solo carácter o un conjunto
      - **close():** cierra el flujo

```java
String ruta = "archivo.txt"; // Path to the file

        try (FileReader fr = new FileReader(ruta)) {
            int caracter;
            while ((caracter = fr.read()) != -1) {
                System.out.print((char) caracter);
            }
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("An I/O error occurred: " + e.getMessage());
        }
```

    - **FileWriter** (salida)
      - **write**(): escribe un byte o un array de bytes
      - **flush():** vacia el búfer, fuerza la escritura
      - close(): cierra el flujo

```
try {
            FileWriter fw = new FileWriter("archivo.txt");
            fw.write("Holaa Mundo");
            fw.close();

            System.out.println("Archivo escrito correctamente.");

        } catch (IOException e) {
            System.out.println("Ocurrió un error: " + e.getMessage());
        }
```

- **Bytes (binarios):**
  - **FileInputStream** (entrada) se suele usar con BufferedInputStream y BufferedOutputStream (Mejora entrada y salida)
    - **read():** lee un solo byte
    - **available():** informa de cuantos bytes quedan disponibles para leer
    - close(): cierra el flujo

```
try {
    FileInputStream fis = new FileInputStream("archivo.bin");
    int byteLeido;
    while( (byteLeido = fis.read()) != -1 ){
    System.out.print(byteLeido);
    }
    fis.close();


} catch (IOException e) {
    System.out.println("Ocurrió un error: " + e.getMessage());
}
```

  - **FileOutputStream** (salida)
    - **write**(): escribe un byte o un array de bytes
    - **flush():** vacia el búfer, fuerza la escritura
    - close(): cierra el flujo

```java
try {
    FileOutputStream fos = new FileOutputStream("archivo.bin");

    fos.write(65);
    fos.close();


} catch (IOException e) {
    System.out.println("Ocurrió un error: " + e.getMessage());
}
```

- Acceso aleatorio o directo:
  Acceder a un registro o posición especifica con un puntero en bytes indicamos la posición exacta de la lectura o escritura. Permite abrir un fichero en lectura “r”o lectura/escritura “rw”.
  - **RandomAccessFile** (entrada) / (salida)
    - **seek():** posicionarnos donde indiquemos en el fichero acepta tipo long
    - **getFilePointer():** obtener el número exacto de la posición del puntero en bytes
    - **read():** leer un byte del fichero
    - **write():** para escribir un byte. Dicho byte sera escrito en la posición actual donde se encuentre el puntero. Ejemplo: write(68); estaremos escribiendo la letra D que corresponde al número 68 en ASCII. Una vez escrito el fichero avanza de posición.

      - **Leer o escribir UN byte:**

```java
String ruta = "fichero.bin";
try {
     // Crear un objeto RandomAccessFile para leer y escribir
     RandomAccessFile file = new RandomAccessFile(ruta, "rw");

     // Posicionar el puntero en el byte 8
     file.seek(8);

     // Obtener la posición actual del puntero
     long filePointer = file.getFilePointer();
     System.out.println("Posición actual del puntero: " + filePointer);

     // Leer un byte desde la posición actual
     int unByte = file.read();
     System.out.println("Byte leído: " + unByte);

     // Escribir el valor 68 (que representa la letra 'D' en ASCII)
     file.write(68);
     System.out.println("Escrito el valor 68 (D) en el archivo.");

     // Cerrar el archivo
     file.close();


} catch (IOException e) {
    System.out.println("Ocurrió un error: " + e.getMessage());
}
```

- **Leer o escribir una VARIOS bytes**: un **array** de bytes

```
String ruta = "fichero.bin";
try {

   // Crear un objeto RandomAccessFile para leer y escribir
   RandomAccessFile file = new RandomAccessFile(ruta, "rw");

   // Array para almacenar los bytes leídos
   byte[] arrayBytes = new byte[1024];

   // Definir la posición de inicio dentro del array (normalmente 0)
   int inicioPuntero = 0;

   // Definir el número de bytes que se intentarán leer
   int size = 1024;

   // Leer los bytes desde el archivo
   int bytesLeidos = file.read(arrayBytes, inicioPuntero, size);

   // Verificar si se han leído bytes correctamente
   if (bytesLeidos != -1) {
    System.out.println("Bytes leídos: " + bytesLeidos);
} else {
    System.out.println("Se alcanzó el final del archivo.");
}
   file.close();


} catch (IOException e) {
    System.out.println("Ocurrió un error: " + e.getMessage());
}
```

- Buffer:
  es un espacio determinado y temporal que se aloja en memoria para realizar ciertas operaciones porque almacena en memoria bloques de bytes completos. Es más rápido respecto FileInputStream o FileOutputStream porque es más rápido acceder a memoria que a disco.
  - **Caracteres:** suele usarse con FileReader y FileWriter
    - **BufferedReader:**
      - **read():** lee un solo carácter o un conjunto
      - **readLine():** lee una línea completa de texto
      - **close():** cierra el flujo

```
    try {

        BufferedReader br = new BufferedReader(new FileReader("archivo.txt"));
        String linea = br.readLine();


    } catch (IOException e) {
        System.out.println("Ocurrió un error: " + e.getMessage());
    }

}
```

- **BufferedWriter:**
  - **write():** escribe
  - **newLine():** es cribe un salto de línea
  - **flush():** vacía el búfer, forza la escritura
  - **close():** cerrar el flujo

```
try {

    BufferedWriter bw = new BufferedWriter(new FileWriter("archivo.txt"));
    bw.write("Hola Mundo");
    bw.newLine();
    //bw.close();


} catch (IOException e) {
    System.out.println("Ocurrió un error: " + e.getMessage());
}
```

- **Bytes:**
  - **BufferedInputStream:**
    - **read():** lee un solo byte
    - **available():** informa de cuantos bytes se pueden leer sin bloquear
    - **close():** cierra el flujo

```
String ruta = "imagen.jpg";

try {

    BufferedInputStream bis = new BufferedInputStream(new FileInputStream("imagen.jpg"));
    int byteLeido = bis.read();

    while ((byteLeido = bis.read()) != -1) {
        // Por ejemplo, imprimir el byte leído en formato hexadecimal
        System.out.printf("%02X ", byteLeido);
    }


} catch (IOException e) {
    System.out.println("Ocurrió un error: " + e.getMessage());
}
```

- **BufferedOutputStream:**
  - **write**(): escribe un byte o un array de bytes
  - **flush():** vacia el búfer, fuerza la escritura
  - **close():** cierra el flujo

```
    byte[] datos = {65, 66, 67, 68};

    try {

        BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream("archivo.bin"));
        bos.write(datos);
        bos.flush();
        bos.close();

    } catch (IOException e) {
        System.out.println("Ocurrió un error: " + e.getMessage());
    }

}
```

Test:

1. La forma más eficiente para leer desde un punto específico de nuestro fichero será:
   - Usar un modo de acceso aleatorio o directo
2. Las clases FileInputStream y FileOutputStream estás orientadas para:
   - El acceso a ficheros binarios en Java
3. ¿Cuál es el propósito de la línea **`fichero.renameTo(fileDestino)?`**
   - Mover el archivo a una ubicación diferente.
4. Usaremos la Clase FileWriter para…
   - Escribir en ficheros que estén basados en la escritura con caracteres
5. ¿Qué valor devuelve el método read() de la clase FileReader?
   - Int
6. Si quisiera saber el posicionamiento del puntero usando una clase RandomAccessFile…
   - Usaríamos el método getFilePointer()
7. El método read() de FileInputStream…
   - Devuelve el primer Byte representado en número entero
8. Si el objetivo en nuestra implementación es manipular un fichero de caracteres:
   - Utilizaremos la clase FileReader

9. ¿Cuál de las siguientes clases se utiliza para escribir bytes brutos en un archivo en Java?

- FileOutputStream

10. En la clase RandomAccessFile el modo de acceso "r"...

- Hace referencia al modo lectura

11. La librería Java que se usará para la gestión de archivos, lectura, escritura y más funcionalidades es:

- Java.io

12. ¿Qué dos clases de acceso a un fichero existen?

- Acceso secuencial y acceso directo o aleatorio

13. Si usamos nuestra clase RandomAccessFile, y nos interesara posicionar el puntero en nuestro fichero:

- Ejecutaríamos el método seek()

14. ¿Qué clase se utiliza para abrir el archivo en modo lectura de caracteres?

- FileReader

15. ¿Qué nos devuelve el método getParent()?

- Devuelve el directorio superior

17. Usaremos el método mkdirs() de la Clase File...

- Para crear un nuevo directorio

18. ¿Qué clase se utiliza para leer líneas completas de texto desde un archivo de manera eficiente?

- BufferedReader

19. ¿A qué está orientado el uso de flujos de bytes?

- A la lectura/escritura de datos en ficheros binarios.

20. ¿Qué hace el método **`createNewFile()?`**

- Crea un nuevo archivo si no existe.