# ACCESO A DATOS

UNIDAD 02 – FICHEROS

# INDICE

# 2.1 – Persistencia de datos en ficheros


# 2.1 – PERSISTENCIA DE DATOS EN FICHEROS

- Los ficheros son el método de almacenamiento de información más elemental.
- En última instancia, todos los métodos de almacenamiento, almacenan los datos en ficheros.
- Los ficheros contienen registros y estos a su vez, contienen un conjunto de campos de longitud fija.
- Para acelerar las búsquedas se usaban ficheros auxiliares de índice para localizar registros según un orden predeterminado.
- IBM desarrolló un avanzado sistema de gestión de ficheros llamado ISAM (indexed sequential Access method).


# 2.1 – PERSISTENCIA DE DATOS EN FICHEROS

- Fueron relegados en los 80 por las BBDD Relacionales.
- COBOL (1959) , aún hoy ampliamente usado en determinados ámbitos, proporciona un excelente soporte para ficheros indexados.
- Veremos organizaciones de ficheros y cómo realizar sencillos programas en Java para consultar, añadir, borrar y modificar información contenida en ellos.


# 2.1 – PERSISTENCIA DE DATOS EN FICHEROS

- Se siguen utilizando ficheros en aplicaciones sencillas.
- Muchos procesos masivos de ejecución periódica o puntual se realizan basándose en datos proporcionados en ficheros de texto.
- También se usan para la importación y exportación de datos entre sistemas.
- Otra utilidad es para copias de seguridad.

# 2.2 – Tipos de ficheros


# 2.2 – TIPOS DE FICHEROS

- Un fichero es una secuencia de bytes.
- Se identifica por su nombre y su ubicación dentro de la jerarquía de directorios.
- Pueden contener cualquier tipo de información, pero, a grandes rasgos, distinguimos dos tipos:
  - Ficheros de texto: única y exclusivamente secuencias de caracteres, (letras, números, espacios, etc).
  - Ficheros binarios: resto de ficheros que no son de texto. Se necesita programa especial dependiendo del fichero.


# 2.2 – TIPOS DE FICHEROS

- Da igual dónde se almacene un texto, se debe hacer con una **codificación**.
- Un texto es una secuencia de caracteres.
- El texto, como cualquier tipo de información, se almacena como una secuencia de bytes.
- Una codificación es un método para representar cualquier texto como una secuencia de bytes.
- El mismo texto, según la codificación empleada, puede representarse como una secuencia de bytes distinta.
- Hay diferentes codificaciones pero se ha estandarizado el Unicode y su codificación UTF-8 (compatible con ASCII).

# 2.3 – La clase File de Java

# 2.3 – LA CLASE FILE DE JAVA

- La versión utilizada durante este curso es Java SE 8, una versión LTE (long term support).
- Las clases que permiten trabajar con ficheros están en el paquete java.io.
- La documentación de java se puede consultar en https://docs.oracle.com/javase/8/docs/api/

# 2.3 – LA CLASE FILE DE JAVA

- La clase File permite obtener información relativa a directorios y ficheros dentro de un sistema de archivos.

| Categoría | Modif/tipo | Método(s) | Funcionalidad |
|---|---|---|---|
| Constructor | | File (String ruta) | Crea objeto File para la ruta indicada. |
| Consulta | boolean | canRead()<br>canWrite()<br>canExecute() | Comprobar si se tiene permisos |
| | boolean | exists() | Comprueba si el fichero/dir existe. |
| | boolean | isDirectory()<br>isFile() | Comprueba si se trata de un directorio o un fichero. |
| | long | length() | Devuelve longitud del fichero. |
| | File | getParent()<br>getParentFile() | Devuelve el directorio Padre. |
| | String | getName() | Devuelve el nombre del fichero |

# 2.3 – LA CLASE FILE DE JAVA

- Métodos de la clase File (Continuación)

| Categoría | Modif/tipo | Método(s) | Funcionalidad |
|---|---|---|---|
| Enumeración | String | list() | Devuelve array con los nombres dentro del directorio |
| | File[] | listFiles() | Devuelve array con los ficheros/dirs dentro del directorio |
| Creación, Borrado y renombrado | boolean | createNewFile() | Crea un nuevo fichero |
| | static File | createTempFile() | Crea un nuevo fichero temporal |
| | boolean | delete() | Borra fichero o directorio. |
| | boolean | renameTo() | Renombra fichero o directorio. |
| | boolean | mkdir() | Crea un directorio. |

# 2.3 – LA CLASE FILE DE JAVA

```java
/*
    Uso de la clase File para mostrar información de ficheros y directorios
 */
package ig.formacion.listadodirectorio;

import java.io.File;

/**...4 lines */
public class listadodirectorio {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        String ruta=".";
        if(args.length>=1) ruta=args[0];

        File fich=new File(ruta);

        if(!fich.exists()) {
            System.out.println("No existe el fichero o directorio ("+ruta+").");
        }
        else {
            if(fich.isFile()){
                System.out.println(ruta+" es un fichero.");
            }
            else {
                System.out.println(ruta+" es un directorio. Contenidos: ");
                File[] ficheros=fich.listFiles(); //Ojo, ficheros o directorios
                for(File f: ficheros) {
                    String textoDescr=f.isDirectory() ? "/":
                    f.isFile() ? "_": "?";
                    System.out.println("("+textoDescr+") "+f.getName());
                }
            }
        }
    }

}
```

- Ejemplo de programa que muestra un listado de ficheros y directorios.

# 2.3 – LA CLASE FILE EN JAVA

***ACTIVIDADES PROPUESTAS.***

a) Modifica el programa anterior para que muestre más información acerca de cada fichero y directorio, al menos el tamaño (si es un fichero), los permisos de los que se dispone sobre el fichero o directorio, y la fecha de última modificación. Los permisos hay que mostrarlos en el formato Linux (rwx ó r-x).

# 2.4 – Gestión de excepciones en java

# 2.4 – GESTIÓN DE EXCEPCIONES EN JAVA

Exception

- Antes de seguir, se debe hacer un repaso al uso de excepciones en el lenguaje Java.
- Cualquier programa Java debe hacer una adecuada gestión de excepciones.
- Una excepción es un evento durante la ejecución que interrumpe el curso normal de la misma.
- Una excepción podría darse, por ejemplo, al dividir por cero.
- Al darse una excepción no controlada, se mostrará mensaje de error y se abortará el programa.

Exception

# 2.4 – GESTIÓN DE EXCEPCIONES EN JAVA

- Tenemos dos formas de gestionar las excepciones, mediante bloques **try-catch** o con **throws**.
- Con **throws** no se gestiona la excepción sino que se deja su gestión al la clase.
- Con **Try-catch** se gestiona la excepción dentro del método.
- Se aconseja usar siempre que se pueda try-catch.

# 2.4 – GESTIÓN DE EXCEPCIONES EN JAVA

Exception

- Throws: Se usa para que se gestione la excepción en el método que lo llama (nivel superior)

```
// Declaración de excepciones lanzadas por método de clase con throws
package ig.formacion.listadodirectorio;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class excepcionesconthrows {
public File creaFicheroTempConCar(String prefNomFich, char car, int numRep) throws IOException {
    File f = File.createTempFile(prefNomFich,"");
    FileWriter fw = new FileWriter(f);
    for (int i=0; i < numRep; i++) fw.write(car);
    fw.close();
    return f;
}
}
```

# 2.4 – GESTIÓN DE EXCEPCIONES EN JAVA

Exception

- Las excepción en java las controlaremos con el bloque Try-catch.

```
1  import java.io.IOException;
2  import java.io.FileReader;
3  public class ExcepcionApp {
4
5      public static void main(String[] args) {
6
7          try{
8
9              prueba();
10
11             //Esta linea no se ejecuta
12             System.out.println("No veras este mensaje");
13
14         }catch(IOException e){
15             System.out.println("Error E/S: el fichero no existe");
16         }
17
18     }
19     public static void prueba() throws IOException{
20
21         //Lanzara una excepcion, pero se lanza en el try-catch del main
22         FileReader fr=new FileReader("casa");
23
24     }
25
26 }
```

# 2.4 – GESTIÓN DE EXCEPCIONES EN JAVA

Exception

- Try-catch simple

**Try** {

//código que se va ha ejecutar

} **catch** (ClaseExcepcion var){

//código si excepción

}

```
1  import java.io.IOException;
2  import java.io.FileReader;
3  public class ExcepcionApp {
4
5      public static void main(String[] args) {
6
7          try{
8              //si el nombre del fichero no existe, lanza el catch
9              FileReader fr=new FileReader("casa");
10
11             //Esta linea no se ejecuta
12             System.out.println("No veras este mensaje");
13         }catch(IOException e){
14             System.out.println("Error E/S: el fichero no existe");
15         }
16
17     }
18
19 }
```

# 2.4 – GESTIÓN DE EXCEPCIONES EN JAVA

Exception

- Try-catch compuesto
- Para distintos tipos de excepciones
- Primero se ponen las más específicas y luego las generales.

```
Try {
  //código que se va ha ejecutar
  } catch (ClaseExcepcionEspecifica var){
  //código si excepción específica
  } catch (ClaseExcepcionGeneral var2){
  //código si excepción general
  }
```

```java
import java.io.IOException;
import java.io.FileReader;
import java.io.FileNotFoundException;
public class ExcepcionApp {

    public static void main(String[] args) {

        FileReader fr=null;
        try{
            //si el nombre del fichero no existe, lanza el catch
            fr=new FileReader("casa");

            //Esta linea no se ejecuta
            System.out.println("No veras este mensaje");
        }catch(FileNotFoundException e){
            System.out.println("Error E/S: el fichero no existe");
        }
        catch(IOException e){
            System.out.println("Si el fichero no existe, este mensaje no se vera");
        }

    }
```

# 2.4 – GESTIÓN DE EXCEPCIONES EN JAVA

Exception

- Try-catch compuesto
- Para distintos tipos de excepciones
- Primero se ponen las más específicas y luego las generales.

```
Try {
  //código que se va ha ejecutar
  } catch (ClaseExcepcionEspecifica var){
  //código si excepción específica
  } catch (ClaseExcepcionGeneral var2){
  //código si excepción general
  }
```

```java
import java.io.IOException;
import java.io.FileReader;
import java.io.FileNotFoundException;
public class ExcepcionApp {

    public static void main(String[] args) {

        FileReader fr=null;
        try{
            //si el nombre del fichero no existe, lanza el catch
            fr=new FileReader("casa");

            //Esta linea no se ejecuta
            System.out.println("No veras este mensaje");
        }catch(FileNotFoundException e){
            System.out.println("Error E/S: el fichero no existe");
        }
        catch(IOException e){
            System.out.println("Si el fichero no existe, este mensaje no se vera");
        }

    }
```

Exception

# 2.4 – GESTIÓN DE EXCEPCIONES EN JAVA

- Excepciones, inicialización y liberación de recursos
- Es frecuente que un bloque de programa Java esté estructurado de la siguiente manera:

```
Inicialización y asignación de recursos
Cuerpo
Finalización y liberación de recursos
```

- La primera y la última parte se deben ejecutar siempre, independientemente de los errores que puedan suceder durante la ejecución del cuerpo.
- Para ello usaremos un bloque finally {}


# 2.4 – GESTIÓN DE EXCEPCIONES EN JAVA

- **Try-catch-finally**
- El código contenido en el **finally** se ejecutará se produzca o no excepción.

```
Try {
  //código que se va ha ejecutar
  } catch (ClaseExcepcionEspecifica var){
  //código si excepción específica
  } finally{
  //código a ejecutar siempre
  }
```

```java
import java.io.IOException;
import java.io.FileReader;
public class ExcepcionApp {

    public static void main(String[] args) {

        FileReader fr=null;
        try{
            //si el nombre del fichero no existe, lanza el catch
            fr=new FileReader("casa");

            //Esta linea no se ejecuta
            System.out.println("No veras este mensaje");
        }catch(IOException e){
            System.out.println("Error E/S: el fichero no existe");
        }finally{
            System.out.println("Este mensaje siempre se ejecutara");
        }

    }

}
```

# 2.4 – GESTIÓN DE EXCEPCIONES EN JAVA

***ACTIVIDADES PROPUESTAS.***

a) Crea un programa en Java que gestione excepciones de la siguiente forma: Crea un array y asigna a un elemento el valor -3. Comprueba la excepción generada y usa un bloque try-catch para controlar el error.

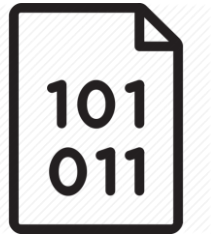
b) Modifica el programa añadiendo la creación de un entero y asignándole un valor de tipo carácter. Comprueba la nueva excepción y después incluye el bloque try-catch el control de la nueva excepción.

c) Modifica el programa para que haya o no excepción, se imprima siempre el texto “En Acceso a Datos, controlamos nuestro errores”.

# 2.5 – Formas de acceso a ficheros en Java

# 2.5 – FORMAS DE ACCESO A LOS FICHEROS


- Existen **dos formas** de acceder a los ficheros.
  - **Acceso secuencial:** comenzando desde el principio para llegar a un elemento hay que pasar por todos los anteriores.
  - **Acceso aleatorio:** se puede acceder directamente a cualquier elemento del fichero.
- Ambas permiten lectura/escritura.



# 2.6 – Operaciones sobre ficheros en Java

# 2.6 – OPERACIONES SOBRE FICHEROS EN JAVA

- Independientemente del tipo de fichero (binario o de texto) y del tipo de acceso (secuencial o aleatorio), las operaciones esenciales son las mismas:

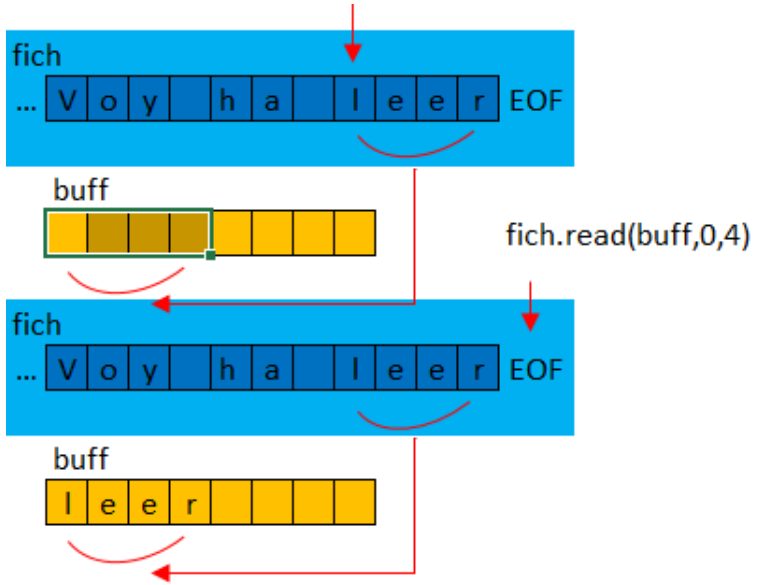

  - ***Apertura*** – Se realiza al crear una instancia de la clase.
  - ***Lectura*** – volcar contenidos desde el fichero a memoria con una instrucción del tipo *read()*.
  - ***Salto*** – hacer avanzar el puntero un número de bytes o caracteres hacia delante con la instrucción *skip()*.
  - ***Escritura*** - volcar contenidos desde memoria al fichero con una instrucción del tipo *write()*.
  - ***Cierre*** – Con una instrucción del tipo *close()*.


- Con el acceso aleatorio se puede situar el cursor en cualquier lugar del fichero.
- Con el acceso secuencial solo se mueve el cursor tras realizar operaciones de lectura, escritura o salto

# 2.6 – OPERACIONES SOBRE FICHEROS EN JAVA

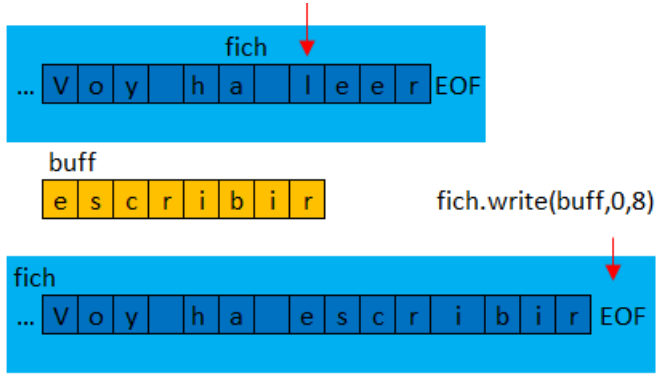

- En la **lectura** hay que indicar el buffer que recibirá los datos que se lean desde el fichero.
- Si no se indica el número de bytes o caracteres a leer, se leerá hasta llenar el buffer.


- fich.read(buff,0,4)



# 2.6 – OPERACIONES SOBRE FICHEROS EN JAVA


- En la **escritura** hay que indicar el buffer desde donde se leerán los datos que se escribirán en el fichero a partir de la posición que apunta el puntero.
- Si no se indica el número de bytes o caracteres a leer, se leerá hasta llenar el buffer.

# 2.7 – Acceso secuencial en Java

# 2.7 – ACCESO SECUENCIAL EN JAVA


- Las operaciones con **ficheros secuenciales** en Java se hacen con flujos (streams).
- Se usa la clase *java.io*.
- Los flujos son abstracciones de alto nivel para secuencias de datos.
- Existe una jerarquía de clases derivada de *cuatro clases*, combinación de flujos binarios y de texto con flujos de entrada y de salida.

# 2.7 – ACCESO SECUENCIAL EN JAVA


- Clases para flujos *secuenciales*
  - *Reader* → *FileReader* y *Writer* → *FileWriter.*
  - *Reader* → *BufferedReader* y *Writer* → *BufferedWriter.*
  - En general conviene utilizar las clases que proporcionan buffering por la mejora del rendimiento.


- Clases para flujos *binarios*
  - *InputStream* → FileInputStream y *OutputStream* → *FileOutputStream*

# 2.7 – ACCESO SECUENCIAL EN JAVA

- OPERACIONES DE LECTURA
- Clase *FileReader*: Permiten leer sobre un archivo de texto
  - Modo secuencial.


- Pertenece al paquete java.io.FileReader.
- Constructores Clase FileReader
  - FileReader (FileReader filereader)
  - FileReader (String fileName)

# 2.7 – ACCESO SECUENCIAL EN JAVA

- Clase *FileReader*
- Ejemplo código:

***FileReader fr = new FileReader("E:\\TMP\\prueba.txt") throws FileNotFoundException;***

*FileReader fr2 = new FileReader(File);*


- Excepcion lanzada:

***FileNotFoundException – Si el archivo no existe o no puede ser creado o no puede ser abierto por cualquier otra razón***

# 2.7 – ACCESO SECUENCIAL EN JAVA

- Clase *BufferedReader*: Permiten leer sobre un archivo de texto utilizando un buffer y funciones avanzadas.
  - Modo secuencial.
- Pertenece al paquete java.io.BufferedReader.
- Constructores Clase BufferedReader
  - BufferedReader (Reader in)
  - BufferedReader (Reader in, int size)

# 2.7 – ACCESO SECUENCIAL EN JAVA

- Clase *BufferedReader*
- Ejemplo código:

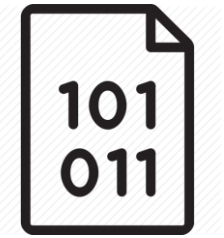
```
BufferedReader br = new BufferedReader(fr);

BufferedReader br = new BufferedReader(new FileReader
("E:\\TMP\\prueba.txt"));
```


- Excepcion lanzada:

***IOException - Si el archivo existe pero es un directorio en lugar de un archivo, no existe pero no puede ser creado o no puede ser abierto por cualquier otra razón.***

# 2.7 – ACCESO SECUENCIAL EN JAVA

- Clase *FileInputStream*: Permiten leer sobre un archivo binario (raw data).
  - Modo secuencial.
- Pertenece al paquete java.io.FileInputStream.
- Constructores Clase FileInputStream
  - FileInputStream (File file)
  - FileInputStream (String fileName)



# 2.7 – ACCESO SECUENCIAL EN JAVA

- Clase *FileInputStream*
- Ejemplo código:

***FileInputStream fis = new FileInputStream("E:\\TMP\\prueba.txt") throws FileNotFoundException;***


***FileInputStream fis2 = new FileInputStream(File);***

- Excepcion lanzada:

***FileNotFoundException – Si el archivo no existe o no puede ser creado o no puede ser abierto por cualquier otra razón.***

# 2.7 – ACCESO SECUENCIAL EN JAVA

- Operaciones de lectura para flujos de entrada.

| Categoría | Modif/tipo | Método(s) |
|---|---|---|
| InputStream | int | read()<br>read(byte[] buffer)<br>read(byte[] buffer, int offset, int longitud) |
| | long | skip(long n) |
| Reader | int | read()<br>read(char[] buffer)<br>read(char[] buffer, int offset, int longitud)<br>read(CharBuffer buffer) |
| | long | skip(long n) |
| BufferedReader | String | readLine() |

# 2.7 – ACCESO SECUENCIAL EN JAVA

- Ejemplo de programa que muestra el contenido de un fichero de texto línea a línea.
- Para leer se usa el método *readLine()* de la clase *BufferedReader*.

```java
package escribeconnumerodelineas;

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

public class EscribeConNumeroDeLineas {

  /**
   * @param args the command line arguments
   */
  public static void main(String[] args) {
    if (args.length < 1) {
      System.out.println("Indicar por favor nombre de fichero");
      return;
    }
    String nomFich = args[0];

    try (BufferedReader fbr = new BufferedReader(new FileReader(nomFich))) {
      int i = 0;
      String linea = fbr.readLine();
      while (linea != null) {
        System.out.format("[%5d] %s", i++, linea);
        System.out.println();
        linea = fbr.readLine();
      }
    } catch (FileNotFoundException e) {
      System.out.println("No existe fichero " + nomFich);
    } catch (IOException e) {
      System.out.println("Error de E/S: " + e.getMessage());
    } catch (Exception e) {
      e.printStackTrace();
    }

  }

}
```

# 2.7 – ACCESO SECUENCIAL EN JAVA

***ACTIVIDADES PROPUESTAS.***

a) Crea un programa que busque un texto dado en un fichero de texto, y que muestre para cada aparición la línea y la columna. Se recomienda leer el fichero línea a línea y, dentro de cada línea, buscar las apariciones del texto utilizando un método apropiado de la clase String. Se puede consultar la documentación de dicha clase en la API de Java (http://docs.Oracle.com/javase/8/docs/api).

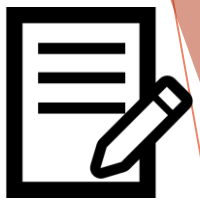
# 2.7 – ACCESO SECUENCIAL EN JAVA

- **Ejemplo con flujos binarios**.
- Hace un volcado binario de un fichero indicado.
- Los contenidos se leen en bloques de 32 bytes y se escribe en una línea de texto.
- Los bytes se escriben en hexadecimal.
- El programa muestra como máximo los primeros 2 kilobytes.

# 2.7 – ACCESO SECUENCIAL EN JAVA

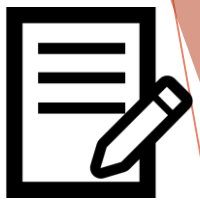
```java
package volcadobinario;

import java.io.InputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class VolcadoBinario {

  static int TAM_FILA = 32;
  static int MAX_BYTES = 2048;
  InputStream is = null;

  public VolcadoBinario(InputStream is) {
    this.is = is;
  }

  public void volcar() throws IOException {
    byte buffer[] = new byte[TAM_FILA];
    int bytesLeidos;
    int offset = 0;
    do {
      bytesLeidos = is.read(buffer);
      System.out.format("[%5d]", offset);
      for (int i = 0; i < bytesLeidos; i++) {
        System.out.format(" %2x", buffer[i]);
      }
      offset += bytesLeidos;
      System.out.println();
    } while (bytesLeidos == TAM_FILA && offset < MAX_BYTES);
  }

  public static void main(String[] args) {
    if (args.length < 1) {
      System.out.println("No se ha indicado ningún fichero");
      return;
    }
    String nomFich = args[0];

    try (FileInputStream fis = new FileInputStream(nomFich)) {
      VolcadoBinario vb = new VolcadoBinario(fis);
      System.out.println("Volcado binario de "+nomFich);
      vb.volcar();
    } catch (FileNotFoundException e) {
      System.err.println("ERROR: no existe fichero " + nomFich);
    } catch (IOException e) {
      System.err.println("ERROR de E/S: " + e.getMessage());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

}
```

# 2.7 – ACCESO SECUENCIAL EN JAVA



- OPERACIONES DE ESCRITURA
- Clase *FileWriter*: Permiten escribir sobre un archivo de texto
  - Modo secuencial.
- Pertenece al paquete java.io.FileWriter.
- Constructores Clase FileWriter
  - FileWriter (File file)
  - FileWriter (File file, boolean append)
  - FileWriter (String fileName)
  - FileWriter (String fileName, boolean append)

# 2.7 – ACCESO SECUENCIAL EN JAVA



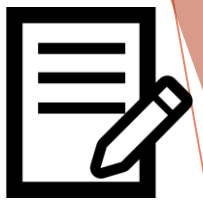
- Clase *FileWriter*
- Ejemplo código:

***FileWriter fw = new FileWriter("E:\\TMP\\prueba.txt") throws IOException;***


- Excepcion lanzada:

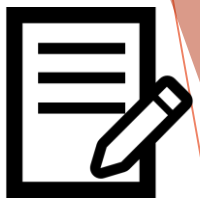
***IOException - Si el archivo existe pero es un directorio en lugar de un archivo, no existe pero no puede ser creado o no puede ser abierto por cualquier otra razón.***

# 2.7 – ACCESO SECUENCIAL EN JAVA



- Clase *BufferedWriter*: Permiten escribir sobre un archivo de texto usando un buffer y funciones avanzadas
  - Modo secuencial.
- Pertenece al paquete java.io.BufferedWriter.
- Constructores Clase bufferedWriter
  - BufferedWriter (Writer out)
  - BufferedWriter (Writer out, int size)

# 2.7 – ACCESO SECUENCIAL EN JAVA



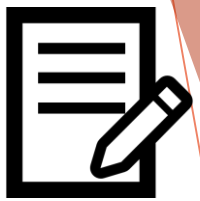
- Clase *BufferedWriter*
- Ejemplo código:

***BufferedWriter bw = new BufferedWriter (new FileWriter("E:\\TMP\\prueba.txt") )throws IOException;***


- Excepcion lanzada:

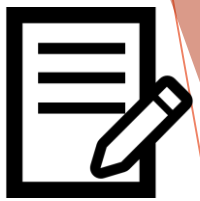
***IOException – Si el archivo existe pero es un directorio en lugar de un archivo, no existe pero no puede ser creado o no puede ser abierto por cualquier otra razón.***

# 2.7 – ACCESO SECUENCIAL EN JAVA



- Clase *FileOutputStream*: Permiten escribir sobre un archivo binario (raw data).
  - Modo secuencial.
- Pertenece al paquete java.io.FileOutputStream.
- Constructores Clase FileInputStream
  - FileOutputStream (File file)
  - FileOutputStream (File file, boolean append)
  - FileOutputStream (String fileName)
  - FileOutputStream (String fileName, boolean append)

# 2.7 – ACCESO SECUENCIAL EN JAVA



- Clase *FileOutputStream*
- Ejemplo código:

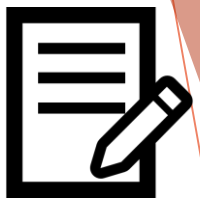
***FileOutputStream fos = new FileOutputStream(“E:\\TMP\\prueba.txt”) throws FileNotFoundException;***

***FileOutputStream fos2 = new FileOutputStream(File);***


- Excepcion lanzada:

***FileNotFoundException – Si el archivo no existe o no puede ser creado o no puede ser abierto por cualquier otra razón.***

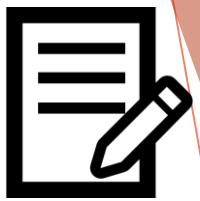
# 2.7 – ACCESO SECUENCIAL EN JAVA



- Operaciones de escritura para flujos de salida.

| Categoría | Modif/tipo | Método(s) |
|---|---|---|
| OutputStream | void | write(int b)<br>write(byte[] buffer)<br>write(byte[] buffer, int offset, int longitud) |
| Writer | void | write(int c)<br>write(char[] buffer)<br>write(char[] buffer, int offset, int longitud)<br>write(String str)<br>write(String str, int offset, int longitud) |
| | writer | append(char c)<br>append(CharSequence csq)<br>append(charSequence csq, int offset, int longitud) |
| BufferedWriter | void | newLine() |

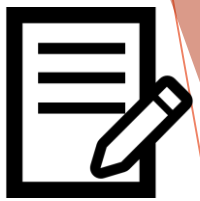
# 2.7 – ACCESO SECUENCIAL EN JAVA



- Métodos con y sin buffering.

| Flujo sin Buffering | Flujo con Buffering |
|---|---|
| new FileInputStream(“f.bin”) | new BufferedInputStream( new FileInputStream(“f.bin”)) |
| new FileOutputStream(“f.bin”) | new BufferedOutputStream( new FileOutpuStream(“f.bin”)) |
| new FileReader(“f.txt”) | new BufferedReader(new FileReader(“f.txt”) |
| new FileWriter(“f.txt”) | new BufferedWriter(new FileWriter(“f.txt”) |

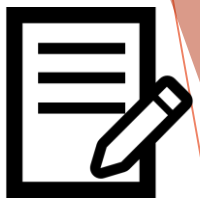
# 2.7 – ACCESO SECUENCIAL EN JAVA



- Ejemplo de escritura de un texto en un fichero.
- Escribe un texto en un fichero.
- Luego lo cierra y lo vuelve a abrir en modo *append* para añadir nuevos contenidos al final. A menos que el fichero ya exista, en cuyo caso no hace nada.
- Se añaden saltos de línea con *newLine()*.

```
1   package escribeenflujosalida;
2
3   import java.io.File;
4   import java.io.FileWriter;
5   import java.io.BufferedWriter;
6   import java.io.IOException;
7
8   // Este programa crea un fichero y escribe un texto en él.
9   // Después lo vuelve a abrir para añadir un texto al final de él.
10  // Si el fichero ya existe, sale sin hacer nada.
11
12  public class EscribeEnFlujoSalida {
13
14    /**...3 lines */
17    public static void main(String[] args) {
18
19      String nomFichero="f_texto.txt";
20      File f=new File(nomFichero);
21      if(f.exists()) {
22        System.out.println("Fichero "+nomFichero+" ya existe. No se hace nada");
23        return;
24      }
25
26      try {
27        BufferedWriter bfw=new BufferedWriter(new FileWriter(f));
28        bfw.write("  Este   es un fichero de texto. ");
29        bfw.newLine();
30        bfw.write(" quizá no está   del todo bien.");
31        bfw.newLine();
32        bfw.close();
33        bfw=new BufferedWriter(new FileWriter(f, true));
34        bfw.write(" Pero     se puede arreglar.");
35        bfw.newLine();
36        bfw.close();
37        System.out.println("Generado fichero " + nomFichero);
38      }
39      catch(IOException e) {
40        System.out.println(e.getMessage());
41      }
42
43      catch(Exception e) {
45        e.printStackTrace();
46      }
47    }
}
```

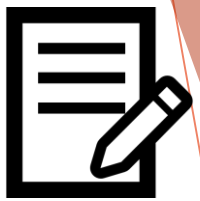
# 2.7 – ACCESO SECUENCIAL EN JAVA



- No es posible eliminar o reemplazar contenidos de un fichero solo con flujos porque es necesario leer y escribir del mismo fichero.
- Se puede hacer usando ficheros auxiliares creados con *createTempFile()* de la clase File.


- Los ficheros temporales creados con *createTempFile()*, normalmente, se crean en una carpeta especial para ficheros temporales (p.ej. /tmp en Linux o variable %temp% en Windows).
- Es conveniente borrar estos archivos antes de finalizar nuestro programa si ya no son necesarios.

# 2.7 – ACCESO SECUENCIAL EN JAVA



- El siguiente programa realiza diversos cambios como eliminar secuencias de espacios y hacer que todas las líneas empiecen por mayúsculas.
- Para las transformaciones en el texto se usa funcionalidades de la clase *Character*.
- Lo más importante es entender la técnica para modificar un fichero, leyendo de un *BufferedReader*, escribiendo en un *BufferedWriter*, utilizando ficheros temporales y renombrando ficheros.

# 2.7 – ACCESO SECUENCIAL EN JAVA

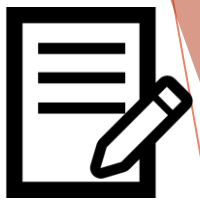
```java
package arreglaficherotexto;

import java.io.File;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Date;
import java.text.SimpleDateFormat;

public class ArreglaFicheroTexto {

    public static void main(String[] args) {

        String nomFichero = "f_texto.txt";
        File f = new File(nomFichero);
        if (!f.exists()) {
            System.out.println("Fichero " + nomFichero + " no existe.");
            return;
        }

        try (BufferedReader bfr = new BufferedReader(new FileReader(f))) {
            File fTemp = File.createTempFile(nomFichero, "");
            System.out.println("Creado fich. temporal " + fTemp.getAbsolutePath());
            try (BufferedWriter bfw = new BufferedWriter(new FileWriter(fTemp))) {
                String linea = bfr.readLine();
                while (linea != null) {   // En resumen, lee de bfr, escribe en bfw
                    boolean principioLinea = true, espacios = false, primerAlfab = false;
                    for (int i = 0; i < linea.length(); i++) {
                        char c = linea.charAt(i);
                        if (Character.isWhitespace(c)) {
                            if (!espacios && !principioLinea) {
                                bfw.write(c);
                            }
                            espacios = true;
                        } else if (Character.isAlphabetic(c)) {
                            if (!primerAlfab) {
                                bfw.write(Character.toUpperCase(c));
                                primerAlfab = true;
                            } else {
                                bfw.write(c);
                            }
                            espacios = false;
                            principioLinea = false;
                        }
                    }
                    bfw.newLine();
                    linea = bfr.readLine();
                }
            }
            String nomFichBackup = nomFichero
                    + "." + new SimpleDateFormat("yyyyMMddHHmmss").format(new Date()) + ".bak";
            if (f.renameTo(new File(nomFichBackup))) {  // Copia de seguridad
                System.out.printf("Fichero %s renombrado como %s", nomFichero, nomFichBackup);
                if (fTemp.renameTo(new File(nomFichero))) {  // Temporal sustituye a original
                    System.out.println("Fich. temporal renombrado como " + nomFichero);
                }
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

# 2.8 – Acceso aleatorio en Java

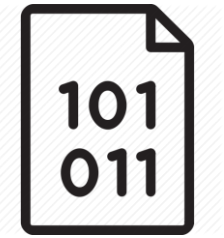
# 2.8 – ACCESO ALEATORIO EN JAVA


- Se usa la clase *RamdomAccessFile*.
- Pertenece al paquete java.io.RamdomAccessFile
- En todo momento el cursor se puede situar en cualquier posición mediante la función *seek()*.
- Sobre el fichero se pueden realizar tanto lectura como escritura (no es necesario dos clases distintas).
- Necesita especificar el modo de acceso al construir el objeto (r | w).
- Un archivo tiene sus registros de un tamaño fijo o predeterminado de antemano.

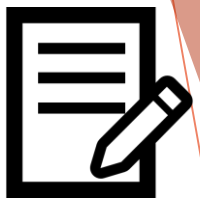
# 2.8 – ACCESO ALEATORIO EN JAVA



- Clase *RamdomAccessFile*: Permiten leer y escribir sobre un archivo binario de acceso aleatorio.
  - Modo aleatorio.
- Pertenece al paquete java.io.RamdomAccessFile.
- Constructores Clase RamdomAccessFile
  - RandomAccessFile(File file, String mode).
  - RandomAccessFile(String name, String mode).



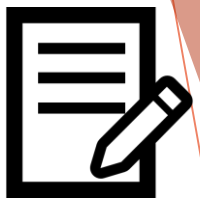
# 2.8 – ACCESO ALEATORIO EN JAVA



- Métodos de la clase RamdomAccessFile.

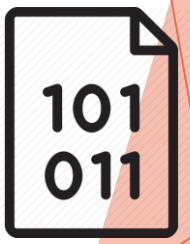
| Categoría | Modif/tipo | Método(s) | Funcionalidad |
|---|---|---|---|
| Constructor | | RandomAccessFile(File file, String mode).<br>RandomAccessFile(String name, String mode) | Abre el fichero en el modo indicado si se dispone de permisos. Modos: r, rw, rwd, rws |
| Consulta | void | close() | Cierra el fichero. |
| | void | seek(long pos) | Posiciona el puntero en la posición indicada |
| | int | read()<br>read(byte[] buffer)<br>read(byte[] buffer, int offset, int longitud) | Lee del fichero. Según la variante, un byte, hasta llenar el buffer, o el número de bytes indicados en la posición (offset) del buffer. -1 si no lee por estar al final. |
| | String | readLine() | Lee hasta el final de la línea actual. |
| | void | write()<br>write(byte[] buffer)<br>write(byte[] buffer, int offset, int longitud) | Escribe en el fichero. Según la variante, un byte, todo el contenido del buffer, o el nº de bytes indicados a partir de la posición del offset. |

# 2.8 – ACCESO ALEATORIO EN JAVA



**EJEMPLO**

- El siguiente ejemplo permite almacenar registros con datos de clientes en un fichero de acceso aleatorio.
- Los datos de cada cliente se almacenan en un ***registro***, que es una estructura de longitud fija dividida en campos de longitud fija.
- El *constructor* de la clase toma una lista con la definición del registro. Cada elemento de la lista contiene la definición de un campo en un par <nombre,longitud>.
- Los valores de los campos para un registro se almacenan en un *HashMap*, que contiene pares <nombre,valor>.
- El método *insertar* tiene dos variantes. Si no se le indica posición añade el registro al final del fichero. Con posición en ese lugar.
- El programa no gestiona las excepciones (throws IOException) y lo deja para el programa principal.



# 2.8 – ACCESO ALEATORIO EN JAVA

```java
package ficheroaccesoaleatorio;

import java.io.File;
import java.io.RandomAccessFile;
import java.io.IOException;
import java.util.List;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import javafx.util.Pair;

public class FicheroAccesoAleatorio {

    private final File f;
    private final List<Pair<String, Integer>> campos;
    private long longReg;
    private long numReg = 0;

    FicheroAccesoAleatorio(String nomFich, List<Pair<String, Integer>> campos) throws
        this.campos = campos;
        this.f = new File(nomFich);
        longReg = 0;
        for (Pair<String, Integer> campo : campos) {
            this.longReg += campo.getValue();
        }
        if (f.exists()) {
            this.numReg = f.length() / this.longReg;
        }
    }
```

```java

    public long getNumReg() {
        return numReg;
    }

    public void insertar(Map<String, String> reg) throws IOException {
        insertar(reg, this.numReg++);
    }

    public void insertar(Map<String, String> reg, long pos) throws IOException {
        try (RandomAccessFile faa = new RandomAccessFile(f, "rws")) {
            faa.seek(pos * this.longReg);
            for (Pair<String, Integer> campo : this.campos) {
                String nomCampo = campo.getKey();
                Integer longCampo = campo.getValue();
                String valorCampo = reg.get(nomCampo);
                if (valorCampo == null) {
                    valorCampo = "";
                }
                String valorCampoForm = String.format("%1$-" + longCampo + "s", valorCampo);
                faa.write(valorCampoForm.getBytes("UTF-8"), 0, longCampo);
            }
        }
    }
```

# 2.8 – ACCESO ALEATORIO EN JAVA

```
55      public static void main(String[] args) {
56
57          List campos = new ArrayList();
58          campos.add(new Pair("DNI", 9));
59          campos.add(new Pair("NOMBRE", 32));
60          campos.add(new Pair("CP", 5));
61
62          try {
63              FicheroAccesoAleatorio faa = new FicheroAccesoAleatorio("fic_acceso_aleat.dat", campos);
64              Map reg = new HashMap();
65              reg.put("DNI", "56789012B");
66              reg.put("NOMBRE", "SAMPER");
67              reg.put("CP", "29730");
68              faa.insertar(reg);
69              reg.clear();
70              reg.put("DNI", "89012345E");
71              reg.put("NOMBRE", "ROJAS");
72              faa.insertar(reg);
73              reg.clear();
74              reg.put("DNI", "23456789D");
75              reg.put("NOMBRE", "DORCE");
76              reg.put("CP", "13700");
77              faa.insertar(reg);
78              reg.clear();
79              reg.put("DNI", "78901234X");
80              reg.put("NOMBRE", "NADALES");
81              reg.put("CP", "44126");
82              faa.insertar(reg, 1);
83      //      faa.insertar(reg,25);  // Probarlo, interesante
84          } catch (IOException e) {
85              System.err.println("Error de E/S: " + e.getMessage());
86          } catch (Exception e) {
87              e.printStackTrace();
88          }
89
90      }
```

# 2.8 – ACCESO ALEATORIO EN JAVA

***ACTIVIDADES PROPUESTAS.***

a) Crear un programa Java que realice la inserción de datos de empleados en un fichero aleatorio llamado “AleatorioEmple.dat”. El programa insertará 7 registros donde para cada empleado se insertarán los siguientes registros:

   I. Un identificador (Int) que coincidirá con el índice +1 con el que se recorren los arrays. Apellido del empleado (String de 10 Caracteres).

   II. Ejemplo: {"FERNANDEZ","GIL","LOPEZ","RAMOS","SEVILLA","CASILLA", "REY"}

   III. Código de departamento (Int). Ejemplo: {10, 20, 10, 10, 30, 30, 20}

   IV. Salario (Double). Ejemplo:{1000.45, 2400.60, 3000.0, 1500.56, 2200.0, 1435.87, 2000.0}

   V. NOTA: El apellido debe tener/ocupar 10 caracteres. Utilizar la clase StringBuffer para almacenar el apellido.

b) Modificar el programa anterior para que lea y mueste los datos del fichero de la siguiente forma:

   i. ID: 1 Apellido: FERNÁNDEZ, Departamento: 10,  Salario: 1000.45

# 2.8 – ACCESO ALEATORIO EN JAVA

***ACTIVIDADES PROPUESTAS.***

a) ¿Qué crees que pasaría si se intentara usar la clase *FicheroAccesoAleatorio* para almacenar un registro en una posición mayor que el número de registros que contiene el fichero? Compruébalo modificando el método *main()* para hacer las pruebas oportunas.

b) Completa la clase *FicheroAccesoAleatorio* con un método que permita obtener el valor de un campo de un registro, dada la posición del registro y el nombre del campo. Por ejemplo: se podría acceder al valor del campo "nombre" del registro situado en la posición5 con *obtenValorCampo(4, "nombre")*. Por coherencia con la manera en que se ha implementado el método *insertar()*, la posición del primer registro debe entenderse que es 0 y no 1. *String* debe tener el mismo formato que en la línea siguiente. Se recomienda utilizar el constructor *String(byte[] bytes, Charset charset)* de *String*.

# RESUMEN

# RESUMEN

- ✓ Un fichero consiste en una secuencia de bytes y se identifica por su nombre y el directorio dentro de la jerarquía del sistema de ficheros.
- ✓ Todos los sistemas de persistencia de datos, almacenan, en última instancia, en ficheros.
- ✓ La persistencia basada en ficheros fue delegada a partir de los años ochenta por los sistemas de BBDD relacionales.
- ✓ Los ficheros pueden ser de texto o binarios. Los ficheros de texto solo contienen texto representado mediantes secuencias de bytes llamada codificación. La codificación más utilizada es UFT-8.

# RESUMEN

- ✓ Java proporciona soporte para operaciones con ficheros y directorios en el paquete java.io. En programación hay que gestionar las excepciones que se produzcan.
- ✓ La clase *File* permite acceder a propiedades de directorios y ficheros, así como crearlos, borrarlos, copiarlos y cambiar su ubicación dentro de la jerarquía de un sistema de archivos.
- ✓ Hay dos formas fundamentales de acceso a ficheros: *Acceso secuencial* y *acceso aleatorio*. En acceso secuencial para leer una información es necesario leer todas las posiciones anteriores. En acceso aleatorio es posible leer directamente la información en cualquier posición del fichero.

# RESUMEN

- Para el acceso secuencial en Java se usan streams. Un fichero es un tipo particular de stream. Java proporciona cuatro jerarquías de clases para streams, pertenecientes a las combinaciones por una parte de *entrada* y *salida* y por otra *binarios* y *texto*. Para streams binarios las clases de origen de las jerarquías son *InputStream* y *OutputStream*, y para streams de texto, *Reader* y *Writer*. Las clases *BufferedInputStream* y *BufferedOutputStream* proporcionan buffering para streams binarios lo que permite acelerar las operaciones. Las clases *BufferedReader* y *BufferedWriter* hacen lo propio para streams de texto, permitiendo además trabajar con líneas de texto.
- Para el acceso aleatorio de ficheros, Java proporciona la clase *RamdomAccessFile* el cual solo proporciona operaciones de lectura y escritura de bytes, lo que no significa que no se pueda utilizar para el acceso aleatorio a ficheros de texto.

# RESUMEN

- ✓ Las organizaciones más sencillas para almacenar datos en ficheros están basadas en registros de longitud fija, en los que cada registro está a su vez formado por varios campos de longitud fija. En cada registro puede existir un campo clave.
- ✓ La organización más sencilla es la organización secuencial. Para acelerar las búsquedas se pueden utilizar ficheros de índice externos, que permiten acceder a los registros contenidos en el fichero en un orden determinado.

# Acceso a Datos

***FIN DE LA UNIDAD***