FICHEROS EN JAVA

Fernando Paniagua

Email: fernando.paniagua@gmail.com

I/O STREAMS

I/O STREAMS

- Definición
- Byte streams
- Character streams
- Buffered streams
- Data steams
- Object streams

I/O STREAMS-DEFINICIÓN

- Un stream es un canal de lectura y escritura entre dos puntos.
- Es unidireccional.
- Es independiente del tipo de dato y dispositivo.
- También llamado *flujo*.
- En Java se encuentran en el paquete io

I/O STREAMS-BYTE STREAMS

- Un *byte stream* permite realizar entradas y salidas de bytes (8 bits).
- Todos los *byte stream* heredan de las clases abstractas InputStream y de OuputStream
- Ejemplo: copia los bytes del fichero.txt entrada en el fichero salida.txt

```
FileInputStream in = null;
FileOutputStream out = null;

try {
      in = new FileInputStream("entrada.txt");
      out = new FileOutputStream("salida.txt");
      int c;
      while ((c = in.read()) != -1) {
            out.write(c);
      }
} finally {
      if (in != null) {
            in.close();
      }
      if (out != null) {
            out.close();
      }
}
```

I/O STREAMS-CHARACTER STREAM

- Un *character stream* permite realizar entradas y salidas de caracteres según el estándar Unicode.
- Todos los *character stream* heredan de Reader y de Writer
- Ejemplo: copia los bytes del fichero.txt entrada en el fichero salida.txt

```
FileReader inputStream = null;
FileWriter outputStream = null;

try {
      inputStream = new FileReader("entrada.txt");
         outputStream = new FileWriter("salida.txt");

      int c;
         while ((c = inputStream.read()) != -1) {
            outputStream.write(c);
      }
} finally {
         if (inputStream != null) {
                  inputStream.close();
         }
         if (outputStream != null) {
            outputStream.close();
      }
}
```

I/O STREAMS-BUFFERED STREAMS

- Un *buffered stream* permite optimizar los accesos a disco mediante el uso de bufferes.
- Hay cuatro clases que "envuelven" (*wrap*) los streams sin buffer:
  - *BufferedInputStream y BufferedOutputStream (para byte streams)*
  - *BufferedReader y BufferedWriter (para carácter streams).*
- Los bufferes de salida disponen de método flush.
- Ejemplo: copia los bytes del fichero.txt entrada en el fichero salida.txt

```
BufferedReader inputStream = null;
BufferedWriter outputStream = null;

try {
       inputStream = new BufferedReader(new FileReader("entrada.txt"));
       outputStream = new BufferedWriter(new FileWriter("salida.txt"));

       int c;
       while ((c = inputStream.read()) != -1) {
             outputStream.write(c);
       }
} finally {
        if (inputStream != null) {
                inputStream.close();
        }
        if (outputStream != null) {
             outputStream.close();
       }
}
```

I/O STREAMS-DATA STREAMS

- Un *data stream* proporciona operaciones de entrada y salida binarias de tipos de datos primitivos (boolean, char, byte, short, int, long, float y doublé).
- Permite operaciones con String.
- Implementan los interfaces DataInput y DataOutput.
- Las implementaciones más utilizadas son DataInputStream y DataOutputStream.
- Ejemplo: escritura de datos de diversos tipos primitivos y String.

```
DataOutputStream out;

out = new DataOutputStream(new BufferedOutputStream(
              new FileOutputStream(dataFile)));

out.writeDouble(12.22);
out.writeInt(15);
out.writeUTF("MADRID");

out.close();
```

I/O STREAMS-OBJECT STREAMS

- Un *object stream* proporciona operaciones de entrada y salida binarias de objetos.
- Los objetos deben ser "serializables" (implementan la interfaz Serializable).
- Las clases son ObjectInputStream y ObjectOutputStream.
- Ejemplo: almacenamiento y recuperación de un objeto que implementa la interfaz Serializable.
  - Consideraciones del ejemplo:
    - Clase Videojuego. Implementa la interfaz Serializable.
    - Clase Ejecutador: instancia un objeto Videojuego (paso 1), lo almacena en el fichero "objetos.out" (paso 2), lee el fichero "objetos.out", regenera la instancia e invoca al método "mostrar" para comprobar que el objeto se ha creado de manera correcta (paso 3).

I/O STREAMS-OBJECT STREAMS

- Ejemplo: clase Videojuego

```
public class Videojuego implements Serializable {
    private String titulo;
    private String plataforma;
    private int edadMinima;

    public Videojuego(String titulo, String plataforma, int edadMinima) {
        this.titulo = titulo;
        this.plataforma = plataforma;
        this.edadMinima = edadMinima;
    }
    public void mostrar(){
        System.out.format("Titulo:%s%n", titulo);
        System.out.format("Plataforma:%s%n", plataforma);
        System.out.format("Edad mínima:%d%n", edadMinima);
    }
}
```

I/O STREAMS-OBJECT STREAMS

- Ejemplo: clase Ejecutador (1/3)

```
public class Ejecutador {
    public static void main(String[] args) {
        //Creamos un objeto Videojuego
        Videojuego v = new Videojuego("Call of Duty","PS3",18);
        Videojuego v2;
```

I/O STREAMS-OBJECT STREAMS

- Ejemplo: clase Ejecutador (2/3)

```
//Creación y almacenamiento
FileOutputStream fos = null;
try {
    fos = new FileOutputStream("objetos.out");
    ObjectOutputStream ous = new ObjectOutputStream(fos);
    ous.writeObject(v);
} catch (FileNotFoundException ex) {
    ex.printStackTrace();
} catch (IOException ex) {
    ex.printStackTrace();
} finally {
    try {
        fos.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

I/O STREAMS-OBJECT STREAMS

- Ejemplo: clase Ejecutador (3/3)

```
//Recuperación e instanciación
FileInputStream fis = null;
try {
    fis = new FileInputStream("objetos.out");
    ObjectInputStream ois = new ObjectInputStream(fis);
    v2 = (Videojuego)ois.readObject();
    v2.mostrar();
} catch (FileNotFoundException ex) {
    ex.printStackTrace();
} catch (IOException ex) {
    ex.printStackTrace();
} catch (ClassNotFoundException ex) {
    ex.printStackTrace();
} finally {
    try {
        fis.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

STREAMS ORIENTADOS A CARACTERES

STREAMS ORIENTADOS A CARACTERES

- java.io.Writer. Clase abstracta para escribir caracteres.
  - java.io.BufferedWriter. Escribe texto en un stream de salida de tipo carácter utilizando buffer. Permite escribir caracteres, array de caracteres y Strings.
  - java.io.PrintWriter. **Permite escribir todo tipo de datos** en un stream de salida de tipo carácter.
  - Java.io.OutputStreamWriter. Permite escribir caracteres como bytes utilizando un conjunto de caracteres específico.

- java.io.Reader. Clase abstracta para leer caracteres.
  - java.io.BufferedReader. Lee texto de un stream de entrada de tipo carácter utilizando buffer. **Permite leer líneas completas**.
  - Java.io.InputStreamReader. Permite leer bytes como caracteres utilizando un conjunto de caracteres específico.

STREAMS ORIENTADOS A CARACTERES

- Ejemplo de escritura de String.

```
PrintWriter pw = new PrintWriter("D:/borrar.txt");
pw.println("LINEA 1");
pw.println("LINEA 2");
pw.println("LINEA 3");
pw.close();
```

STREAMS ORIENTADOS A CARACTERES

- **Ejemplo de lectura de String.**

```
FileReader fr = new FileReader("D:/borrar.txt");
BufferedReader br = new BufferedReader(fr);
String linea;
//Leemos la primera línea
linea = br.readLine();
System.out.println(linea);
//Leemos todas las líneas hasta que se acaba el fichero (lee null)
while ((linea = br.readLine())!=null){
    System.out.println(linea);
}
br.close();
fr.close();
```

FILE I/O

FILE I/O

- Clase File
- Operaciones con Path
- Operaciones con Files
- Operaciones con directorios
  - Creación y lectura
  - Navegación
- Operaciones con ficheros (creación, escritura, lectura)
- Operaciones con ficheros (búsquedas)

FILE I/O-CLASE FILE

- **Clase File**
- Operaciones con Path
- Operaciones con Files
- Operaciones con directorios
  - Creación y lectura
  - Navegación
- Operaciones con ficheros (creación, escritura, lectura)
- Operaciones con ficheros (búsquedas)

- La clase File está dentro del paquete java.io.
- Es una representación abstracta de un fichero o de una ruta (path), proporcionando una vista independiente del sistema operativo.
- Dispone de métodos para crear, borrar o renombrar ficheros y directorios, determinar la existencia de un fichero o directorio, determinar el tamaño de un fichero , comprobar permisos, determinar si es un directorio o un fichero, obtener el contenido de un directorio,…

https://docs.oracle.com/javase/8/docs/api/java/io/File.html

FILE I/O-CLASE FILE

- **Clase File**
- Operaciones con Path
- Operaciones con Files
- Operaciones con directorios
  - Creación y lectura
  - Navegación
- Operaciones con ficheros (creación, escritura, lectura)
- Operaciones con ficheros (búsquedas)

- Ejemplo: crear un fichero

```
File f = new File("D:/nuevofichero.txt");
f.createNewFile();
```

- Ejemplo: borrar un fichero

```
File f = new File("D:/nuevofichero.txt");
f.delete();
```

https://docs.oracle.com/javase/8/docs/api/java/io/File.html

FILE I/O-CLASE FILE

- **Clase File**
- Operaciones con Path
- Operaciones con Files
- Operaciones con directorios
  - Creación y lectura
  - Navegación
- Operaciones con ficheros (creación, escritura, lectura)
- Operaciones con ficheros (búsquedas)

- Ejemplo: muestra el contenido de un directorio

```
File f = new File("D:/software");
File fs[] = f.listFiles();
for (int i=0;i<fs.length;i++){
    System.out.println(fs[i].getName());
}
```

https://docs.oracle.com/javase/8/docs/api/java/io/File.html

FILE I/O-CLASE FILE

- **Clase File**
- Operaciones con Path
- Operaciones con Files
- Operaciones con directorios
  - Creación y lectura
  - Navegación
- Operaciones con ficheros (creación, escritura, lectura)
- Operaciones con ficheros (búsquedas)

- Ejemplo: renombra un directorio

```
File f = new File("D:/misprogramas");
f.renameTo(new File("D:/otrodirectorio"));
```

https://docs.oracle.com/javase/8/docs/api/java/io/File.html

FILE I/O-CLASE FILE

- **Clase File**
- Operaciones con Path
- Operaciones con Files
- Operaciones con directorios
  - Creación y lectura
  - Navegación
- Operaciones con ficheros (creación, escritura, lectura)
- Operaciones con ficheros (búsquedas)

- Ejemplo: crear un fichero temporal

```
File f = File.createTempFile("temporal",".txt",new File("D:/"));
f.deleteOnExit();
```

https://docs.oracle.com/javase/8/docs/api/java/io/File.html

FILE I/O-PATH

- Clase File
- **Operaciones con Path**
- Operaciones con Files
- Operaciones con directorios
  - Creación y lectura
  - Navegación
- Operaciones con ficheros (creación, escritura, lectura)
- Operaciones con ficheros (búsquedas)

- La clase Path aparece en la versión 7 de Java SE.
- Está incluida dentro del package java.nio.file
- Representa un fichero o un directorio.
- Es dependiente de la plataforma:
  - Unix, Linux: /home/paniagua/documents
  - Windows: C:\Users\Paniagua\Documents

FILE I/O-PATH

- Clase File
- **Operaciones con Path**
- Operaciones con Files
- Operaciones con directorios
  - Creación y lectura
  - Navegación
- Operaciones con ficheros (creación, escritura, lectura)
- Operaciones con ficheros (búsquedas)

- Maneras de construir un Path (atención a Paths):
  - `Path p1 = Paths.get("/home/micarpeta");`
  - `Path p2 = Paths.get(URI.create("file://home/datos.txt");`
  - `Path p3 = FileSystem.getDefaults().getPath("/home/micarpeta");`
- Métodos de la clase Path:
  - `getFileName()//Devuelve el nombre del fichero`
  - `getFileSystem()//Devuelve el Filesystem`
  - `getName(int index) // Devuelve el nombre del path que ocupa index en la ruta`
  - `getNameCount()//Devuelve el número de elementos en la ruta`
  - `getParent()//Devuelve el padre en la ruta`
  - `getRoot()//Devuelve el raíz de la ruta`
  - `toFile()//Convierte la ruta en un File`
  - `toUri()//Convierte la ruta en una URI`

FILE I/O-FILES

- Está incluida dentro del package java.nio.file
- Ofrece métodos estáticos de lectura, escritura y manipulación de ficheros y directorios.
- Los métodos de la clase Files funcionan sobre instancias de la clase Path.

FILE I/O-FILES

- Clase File
- Operaciones con Path
- **Operaciones con Files**
- Operaciones con directorios
  - Creación y lectura
  - Navegación
- Operaciones con ficheros (creación, escritura, lectura)
- Operaciones con ficheros (búsquedas)

- Métodos de la clase Files:
  - exists. Indica si el fichero existe (true) o no existe o su existencia no puede ser determinada (false).
  - notExists. Indica si el fichero no existe (true) o existe o su existencia no puede ser determinada (false).
  - isReadable
  - isWritable
  - isExecutable
  - delete. Elimina un Path. Con directorios deben estar vacíos.
  - deleteIfExists. Elimina un Path si existe. Evita tratar la excepción.
  - copy. Permite copiar un directorio (no el contenido) o un fichero.
  - move. Permite mover un directorio.
  - size, isDirectory, isHidden, getOwner,…

FILE I/O-DIRECTORIOS

- Creación:
  - Utilizando el método estático createDirectory de la clase Files.
  - Ejemplo:

```
Files.createDirectories(Paths.get("z:/automatico"));
```

- Lectura del contenido de un directorio:

```
Path dir = Paths.get("z:/automatico");
try (DirectoryStream<Path> stream = Files.newDirectoryStream(dir)) {
    for (Path file : stream) {
        System.out.println(file.getFileName());
    }
} catch (IOException | DirectoryIteratorException x) {
    System.err.println(x);
}
```

FILE I/O-DIRECTORIOS

- Clase File
- Operaciones con Path
- Operaciones con Files
- **Operaciones con directorios**
  - Creación y lectura
  - **Navegación**
- Operaciones con ficheros (creación, escritura, lectura)
- Operaciones con ficheros (búsquedas)

- Navegación:
  - Método getRootDirectories
  - Ejemplo: muestra todos los nodos que cuelgan de root (en Windows muestra las unidades de disco)

```
Iterable<Path> dirs = FileSystems.getDefault().getRootDirectories();
for (Path nombre: dirs) {
    System.err.println(name);
}
```

FILE I/O-DIRECTORIOS

- Clase File
- Operaciones con Path
- Operaciones con Files
- **Operaciones con directorios**
  - Creación y lectura
  - **Navegación**
- Operaciones con ficheros (creación, escritura, lectura)
- Operaciones con ficheros (búsquedas)

- Navegación:
  - Muestra el contenido de cada unidad.

```
Iterable<Path> dirs = FileSystems.getDefault().getRootDirectories();
for (Path name : dirs) {
     System.out.println(name);
     File dir = new File(name.toString());
     String[] ficheros = dir.list();
     if (ficheros!=null){
          for (int x = 0; x < ficheros.length; x++) {
               System.out.println(ficheros[x]);
          }
     }
}
```

FILE I/O-DIRECTORIOS

- Clase File
- Operaciones con Path
- Operaciones con Files
- **Operaciones con directorios**
  - Creación y lectura
  - **Navegación**
- Operaciones con ficheros (creación, escritura, lectura)
- Operaciones con ficheros (búsquedas)

- Navegación:
  - Muestra el contenido del contenido de cada unidad.

```
public static void main(String[] args) {
Iterable<Path> dirs = FileSystems.getDefault().getRootDirectories();
for (Path name : dirs) {
     System.out.println(name);
     File dir = new File(name.toString());
     File[] ficheros = dir.listFiles();
     if (ficheros != null) {
          for (int x = 0; x < ficheros.length; x++) {
               System.out.println(ficheros[x]);
               File[] masficheros = ficheros[x].listFiles();
               if (masficheros!=null){
                    for (int y=0; y < masficheros.length; y++){
                         System.out.println("\t" + masficheros[y]);
                    }
               }
          }
     }
}
```

FILE I/O-OPERACIONES

- Clase File
- Operaciones con Path
- Operaciones con Files
- Operaciones con directorios
  - Creación y lectura
  - Navegación
- **Operaciones con ficheros (creación, escritura, lectura)**
- Operaciones con ficheros (búsquedas)

- Creación, escritura, lectura:
  - Files.readAllBytes. Lee todos los bytes de un fichero (para ficheros pequeños).
  - Files.newBufferedReader. Proporciona un BufferedReader para leer ficheros de texto.
  - Files.newBufferedWriter. Proporciona un BufferedWriter para escribir ficheros de texto.
  - Files.newInputStream. Proporciona InputStream para leer ficheros de texto sin buffer.
  - Files.newOutputStream. Proporciona OutputStream para escribir ficheros de texto sin buffer.

FILE I/O-OPERACIONES

- Clase File
- Operaciones con Path
- Operaciones con Files
- Operaciones con directorios
  - Creación y lectura
  - Navegación
- Operaciones con ficheros (creación, escritura, lectura)
- **Operaciones con ficheros (búsquedas)**

- Java permite realizar operaciones de búsqueda de ficheros mediante el uso de la interfaz java.nio.PathMatcher.
- Cada sistema de ficheros proporciona una implementación a la que se accede mediante el método estático getPathMatcher

FILE I/O-OPERACIONES

- Clase File
- Operaciones con Path
- Operaciones con Files
- Operaciones con directorios
  - Creación y lectura
  - Navegación
- Operaciones con ficheros (creación, escritura, lectura)
- **Operaciones con ficheros (búsquedas)**

- Ejemplo: buscar los ficheros con extensión gif, png y jpg almacenados en una ubicación.

```
//El filtro del matcher no tiene en cuenta si las extensiones son
//mayúsculas o minúsculas
PathMatcher matcher
    = FileSystems.getDefault().getPathMatcher("glob:*.{gif,PNG,jpg}");
Path filename = Paths.get("E:\\datos\\fotografias\\PokemonGO");
try (DirectoryStream<Path> stream = Files.newDirectoryStream(filename)) {
    for (Path file : stream) {
        if (matcher.matches(file.getFileName())){
                    System.out.println(file.getFileName());
                }
        }
} catch (IOException | DirectoryIteratorException ex) {
    ex.printStackTrace();
}
```

ENLACES

- Oracle. Java Tutorial: Basic I/O
  - http://docs.oracle.com/javase/tutorial/essential/io/index.html
- Package java.nio
  - https://docs.oracle.com/javase/7/docs/api/java/nio/package-summary.html
- Package java.io
  - https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html
- tutorialpoint.com
  - https://www.tutorialspoint.com/java/java_files_io.htm
- Nanyang Technological University
  - http://www.ntu.edu.sg/home/ehchua/programming/java/j5b_io.html