UTN F r a
Avellaneda

Programación II

Clase 19

Lectura/escritura de archivos. Serialización en Java.

Lic. Matias Ramos

El concepto de Stream

- Un Stream es un flujo de datos que va de un origen a un destino.

- Entrada (Input): leer datos desde una fuente (archivo, teclado, red).
- Salida (Output): enviar datos a un destino (archivo, consola, red).

- Los Streams permiten leer/escribir byte a byte o carácter a carácter.

Tipos de Streams

Tipo	Flujo de datos	Ejemplo
Byte Streams	Datos binarios (imágenes, PDF, etc.)	FileInputStream, FileOutputStream
Character Streams	Texto (caracteres Unicode)	FileReader, FileWriter
Buffered Streams	Lectura/escritura eficiente mediante memoria intermedia	BufferedReader, BufferedWriter
Data/Object Streams	Lectura/escritura de tipos primitivos u objetos	ObjectInputStream, ObjectOutputStream

La clase File

- La clase java.io.File puede representar un archivo o un directorio
- Se construye a partir del nombre y opcionalmente un path (ruta), que si no se especifica utiliza la default, que es aquella donde se ejecuta el programa.
- Tiene asociado un path para localizarlo.

```
//Archivo en ubicación actual (raiz del proyecto, poco utilizado)
File file = new File("proj.txt");
//Archivo en ubicación absoluta, nomenclatura Windows. Un '\' se escapa con un '\'
File textFile = new File("c:\\data\\data.txt");
//Directorio, ubicación absoluta, nomenclatura Linux/Unix/Mac
File tempDir = new File("/tmp");
//Archivo, especificando directorio en primer parámetro.
File tempFile = new File("/tmp","info.bak");
```

La clase File - Metodos

- Si un archivo o directorio existe se utiliza el método **exists()**
- Saber si un objeto File es archivo o directorio se utilizar el método **isFile()** o **isDirectory()**
- En caso de directorio, si no existe crearlo. El método **mkdir()** crea un directorio y **mkdirs()** crea la ruta completa.
- El método **list()** lista de todos los nombres de archivos del directorio.
- El método **listFiles()** lista de todos los archivos del directorio.

Escritura en un archivo (flujo de salida)

1. Crear un objeto File que represente el archivo.
2. Usar FileWriter y BufferedWriter para escribir.
   FileWriter → escribe caracteres
   BufferedWriter → mejora la eficiencia (usa buffer)
3. Cerrar los flujos manualmente.

Escritura en un archivo (flujo de salida)

```
File archivo = new File(pathname: "mensajes.txt");
BufferedWriter bw = null;
try {
    // El segundo parámetro 'true' permite agregar
    FileWriter fw = new FileWriter(archivo, append
    bw = new BufferedWriter(fw);

    bw.write(str: "Hola, este es un mensaje guarda
    bw.newLine();
```

```
}catch (IOException e) {
    System.out.println("Error al escrib
} finally {
    try {
        if (bw != null) bw.close();
    } catch (IOException e) {
        System.out.println(" Error al c
    }
```


LIVE DEMO

Lectura desde un archivo (flujo de entrada)

1. Crear un objeto File que represente el archivo.
2. Usar FileReader y BufferedReader para leer su contenido.
   FileReader → lee carácter a carácter
   BufferedReader → permite leer líneas completas
3. Cerrar los flujos manualmente.
4. Siempre cerrar los flujos para liberar recursos.

Lectura desde un archivo (flujo de entrada)

```java
File archivo = new File(pathname: "mensajes.txt");
BufferedReader br = null;
try {
    FileReader fr = new FileReader(archivo);
    br = new BufferedReader(fr);

    String linea;
    System.out.println(x: "Contenido del archivo:");

    while ((linea = br.readLine()) != null) {
        System.out.println(linea);
    }
```

```java
} catch (FileNotFoundException e) {
    System.out.println("No se encontró e
} catch (IOException e) {
    System.out.println("Error al leer el
} finally {
    try {
        if (br != null) br.close();
    } catch (IOException e) {
        System.out.println(" Error al ce
    }
}
```


LIVE DEMO

Manejo de excepciones

- Todas las operaciones de archivo lanzan IOException.
- Siempre deben manejarse con try-catch o propagarse con throws.
- Conviene usar try-with-resources para evitar errores de cierre manual.


LIVE DEMO

Lectura y escritura de binarios

- Para archivos que no son texto, se usan los Byte Streams.
- Lectura/escritura byte a byte con FileInputStream y FileOutputStream.
- Muy útil para imágenes, audio, PDF, etc.

Lectura y escritura de binarios

```
File orig = new File("files", "orig.pdf");
File dest = new File("files", "dest.pdf");

FileInputStream fis = new FileInputStream(orig);//Lee el archivo definido en orig
BufferedInputStream bis = new BufferedInputStream(fis);//Optimiza lectura de disco
FileOutputStream fos = new FileOutputStream(dest);//Escribe en archivo dest
BufferedOutputStream bos = new BufferedOutputStream(fos);//Optimiza escritura a disco
/** Lee bloques de 1024 bytes desde el input stream, y los copia en el output stream. */
int size = 1024;
byte[] buf = new byte[size];
int len;
while ((len = bis.read(buf, 0, size)) > -1){
    bos.write(buf, 0, len);
}
//Fuerza escritura buffer
bos.flush();
//Se deben cerrar ambos, por ir a disco.
bis.close();
bos.close();
```

Serialización en Java

- Es el proceso de convertir un objeto en una secuencia de bytes.
- Permite guardar el estado de un objeto o transmitirlo por red.
- Se realiza con:
  ObjectOutputStream → escribir objeto
  ObjectInputStream → leer objeto
- Ideal para persistir datos de forma sencilla sin base de datos.

Requisitos para serializar

- La clase debe implementar Serializable.
- Todos sus atributos deben ser serializables.
- Se recomienda incluir un serialVersionUID.
- Ideal para persistir datos de forma sencilla sin base de datos.

```java
class Persona implements Serializable {
    private static final long serialVersionUID = 1L;
    private String nombre;
    private int edad;
}
```

Ejemplo de serializacion

```java
ObjectOutputStream oos = null;
try {
    FileOutputStream fos = new FileOutputStream(name: "persona.dat");
    oos = new ObjectOutputStream(fos);

    oos.writeObject(persona);
    System.out.println(x: "Persona serializada correctamente");

} catch (IOException e) {
    System.out.println("Error al serializar: " + e.getMessage());
} finally {
    try {
        if (oos != null) oos.close();
    } catch (IOException e) {
```

Ejemplo de deserializacion

```java
ObjectInputStream ois = null;
Persona persona = null;
try {
    FileInputStream fis = new FileInputStream(name: "persona.dat");
    ois = new ObjectInputStream(fis);

    persona = (Persona) ois.readObject();
    System.out.println(x: "Persona deserializada correctamente");

} catch (FileNotFoundException e) {
    System.out.println("Archivo no encontrado: " + e.getMessage());
} catch (IOException | ClassNotFoundException e) {
    System.out.println("Error al deserializar: " + e.getMessage());
} finally {
    try {
        if (ois != null) ois.close();
    } catch (IOException e) {
```


LIVE DEMO