Sesión 4: Entrada/Salida

Índice

- Flujos de E/S
- Entrada y salida estándar
- Acceso a ficheros
- Codificación de datos
- Serialización de objetos

Entrada/Salida

- Flujos de E/S
- Entrada y salida estándar
- Acceso a ficheros
- Codificación de datos
- Serialización de objetos

Flujos de E/S

- **Las aplicaciones muchas veces necesitan enviar datos a un determinado destino o leerlos de una determinada fuente**
  - **Ficheros en disco, red, memoria, otras aplicaciones, etc**
  - **Esto es lo que se conoce como E/S**
- **Esta E/S en Java se hace mediante flujos (*streams*)**
  - **Los datos se envían en serie a través del flujo**
  - **Se puede trabajar de la misma forma con todos los flujos, independientemente de su fuente o destino**
    - **Todos derivan de las mismas clases**

Tipos de flujos según el tipo de datos

- **Según el tipo de datos que transportan, distinguimos**
  - **Flujos de bytes (con sufijos `InputStream` y `OutputStream`)**
  - **Flujos de caracteres (con sufijos `Reader` y `Writer`)**
- **Superclases**

	Entrada	Salida
Bytes	`InputStream`	`OutputStream`
Caracteres	`Reader`	`Writer`

Tipos de flujos según su propósito

- **Distinguimos:**
  - **Canales de datos**
    - Simplemente llevan datos de una fuente a un destino
      - Ficheros: `FileInputStream, FileReader, FileOutputStream, FileWriter`
      - Memoria: `ByteArrayInputStream, CharArrayReader, ...`
      - Tuberías: `PipedInputStream, PipedReader, PipedWriter, ...`
  - **Flujos de procesamiento**
    - Realizan algún procesamiento con los datos
      - Impresión: `PrintWriter, PrintStream`
      - Conversores de datos: `DataOutputStream, DataInputStream`
      - Bufferes: `BufferedReader, BufferedInputStream, ...`

Acceso a los flujos

- **Todos los flujos tienen una serie de métodos básicos**

Flujos	Métodos
`InputStream, Reader`	`read, reset, close`
`OutputStream, Writer`	`write, flush, close`

- **Los flujos de procesamiento**
  - **Se construyen a partir de flujos canales de datos**
  - **Los extienden proporcionando métodos de más alto nivel, p.ej:**

Flujos	Métodos
`BufferedReader`	`readLine`
`DataOutputStream`	`writeInt, writeUTF,...`
`PrintStream, PrintWriter`	`print, println`

Entrada/Salida

- Flujos de E/S
- Entrada y salida estándar
- Acceso a ficheros
- Codificación de datos
- Serialización de objetos

Objetos de la E/S estándar

- **En Java también podemos acceder a la entrada, salida y salida de error estándar**
- **Accedemos a esta E/S mediante flujos**
- **Estos flujos se encuentran como propiedades estáticas de la clase `System`**

	Tipo de flujo	Propiedad
Entrada	`InputStream`	`System.in`
Salida	`PrintStream`	`System.out`
Salida de error	`PrintStream`	`System.err`

Salida estándar

- La salida estándar se ofrece como flujo de procesamiento `PrintStream`
  - Con un `OutputStream` a bajo nivel sería demasiado incómoda la escritura
- Este flujo ofrece los métodos `print` y `println` que permiten imprimir cualquier tipo de datos básico
  - En la salida estándar

```
System.out.println("Hola mundo");
```

  - En la salida de error

```
System.err.println("Error");
```

Entrada/Salida

- Flujos de E/S
- Entrada y salida estándar
- Acceso a ficheros
- Codificación de datos
- Serialización de objetos

Flujos de ficheros

- **Canales de datos para acceder a ficheros**

	Entrada	Salida
Caracteres	`FileReader`	`FileWriter`
Binarios	`FileInputStream`	`FileOutputStream`

- **Se puede acceder a bajo nivel directamente de la misma forma que para cualquier flujo**

- **Podemos construir sobre ellos flujos de procesamiento para facilitar el acceso de estos flujos**

Lectura y escritura de ficheros

```
public void copia_fichero() {
     int c;
     try {
         FileReader in = new FileReader("fuente.txt");
         FileWriter out = new FileWriter("destino.txt");
         while( (c = in.read()) != -1)
         {
             out.write(c);
         }
         in.close();
         out.close();
     } catch(FileNotFoundException e1) {
         System.err.println("Error: No se encuentra el fichero");
     } catch(IOException e2) {
         System.err.println("Error leyendo/escribiendo fichero");
     }
}
```

Uso de flujos de procesamiento

```
public void escribe_fichero() {
     FileWriter out = null;
     PrintWriter p_out = null;
     try {
         out = new FileWriter("result.txt");
         p_out = new PrintWriter(out);
         p_out.println("Este texto será escrito en el fichero");
     } catch(IOException e) {
         System.err.println("Error al escribir en el fichero");
     } finally {
         p_out.close();
     }
}
```

Ficheros de propiedades (Properties)

- **La clase *Properties* maneja ficheros de propiedades, donde se guardan conjuntos de pares *clave=valor*, separados un por línea**
  - **Utiles para ficheros de configuración de aplicaciones**

```
#Comentarios
elemento1=valor1
elemento2=valor2
...
elementoN=valorN
```

Operaciones con propiedades

- **Cargar un fichero de propiedades**

```
Properties p = new Properties();
p.load(new FileInputStream('miFichero.txt');
```

- **Acceder a las propiedades**

```
// Si sabemos su nombre:
String valor = p.getProperty('miPropiedad');

// Si no, podemos recorrerlas todas
Enumeration en = p.propertyNames();
while (en.hasMoreElements()) {
    String nombre = en.nextElement();
    String valor = p.getProperty(nombre);
    ...
}
```

Operaciones con propiedades

- **Modificar el valor de una propiedad**

```
p.setProperty('miPropiedad','Otro valor');
```

- **Guardar las propiedades**

```
p.store(new FileOutputStream('miFichero.txt');
```

- **En general, los ficheros de propiedades suelen tener extensión *.properties***

Lectura de tokens

- **Si el fichero a leer tiene una gramática o estructura homogénea, puede ser más cómo leerlo mediante *tokens*, que identifiquen cada parte del fichero y la extraigan entera**
- **La clase *StreamTokenizer* parte la entrada en los diferentes *tokens* (elementos diferenciables) que contiene, sacándolos uno a uno.**
- **La clase contiene una serie de constantes que identifican los diferentes tipos de *tokens* que se pueden leer:**
  - ***StreamTokenizer.TT_WORD*: para palabras**
  - ***StreamTokenizer.TT_NUMBER*: para números**
  - ***StreamTokenizer.TT_EOL*: fin de línea**
  - ***StreamTokenizer.TT_EOF*: fin de fichero**

Operaciones con tokenizers

- Crear el *tokenizer*

```
FileReader r = new FileReader('miFichero.txt');
StreamTokenizer st = new StreamTokenizer(r);
```

- Leer los tokens del fichero

```
while(st.nextToken() != StreamTokenizer.TT_EOF) {
    switch(st.ttype) {
          case StreamTokenizer.TT_WORD:
                System.out.println('Palabra '+st.sval);
                break;
          case StreamTokenizer.TT_NUMBER:
                System.out.println('Num '+st.nval);
```

Rangos de caracteres en tokenizers

- **Podemos distinguir tres tipos de caracteres al leer mediante tokens:**
  - ***ordinaryChars*: caracteres que forman por sí mismos un token solo. Su valor se guarda en la propiedad *ttype* del tokenizer**
  - ***wordChars*: caracteres que pueden formar palabras combinados con otros *wordChars***
  - ***whiteSpaceChars*: caracteres que se utilizan para separar tokens**
- **Podemos establecer cada uno de estos tres subconjuntos con métodos como:**

```
// Indicar que en las palabras habrá cualquier caracter entre 32
// y 127
st.wordChars(32,127);

// Establecer como separadores de tokens los dos puntos y el –
st.whiteSpaceChars(':',':');
st.whiteSpaceChars('-','-');
```

Sistema de ficheros

- La clase `File` contiene utilidades para trabajar con el sistema de ficheros
  - Constantes para indicar los separadores de directorios (‘/’ ó ‘\’)
    - Hace las aplicaciones independientes de la plataforma
  - Crear, borrar o renombrar ficheros y directorios
  - Listar los ficheros de un directorio
  - Comprobar y establecer los permisos sobre ficheros
  - Obtener la ruta de un fichero
  - Obtener datos sobre ficheros (tamaño, fecha, etc)
  - Etc...

Acceso a recursos

- **Los recursos incluidos en un JAR no se encuentran directamente en el sistema de ficheros**
  - **No podremos utilizar los objetos anteriores para acceder a ellos**
- **Accedemos a un recurso en el JAR con**

```
getClass().getResourceAsStream("/datos.txt");
```

- **Anteponiendo '/' se busca de forma relativa al raíz del JAR**
- **Si no, buscará de forma relativa al directorio correspondiente al paquete de la clase actual**

Entrada/Salida

- Flujos de E/S
- Entrada y salida estándar
- Acceso a ficheros
- Codificación de datos
- Serialización de objetos

Codificación

- Podemos codificar de forma sencilla los datos para enviarlos a través de un flujo de bytes (en serie)
- Utilizaremos un flujo `DataOutputStream`

```
String nombre = "Jose";
String edad = 25;

ByteArrayOutputStream baos = new ByteArrayOutputStream();

DataOutputStream dos = new DataOutputStream(baos);
dos.writeUTF(nombre);
dos.writeInt(edad);

dos.close();
baos.close();

byte [] datos = baos.toByteArray();
```

Descodificación

- Para descodificar estos datos del flujo realizaremos el proceso inverso
- Utilizamos un flujo `DataInputStream`

```
ByteArrayInputStream bais = new ByteArrayInputStream(datos);
DataInputStream dis = new DataInputStream(bais);

String nombre = dis.readUTF();
int edad = dis.readInt();

dis.close();
bais.close();
```

Entrada/Salida

- Flujos de E/S
- Entrada y salida estándar
- Acceso a ficheros
- Codificación de datos
- Serialización de objetos

Entrada/Salida de objetos

- Si queremos enviar un objeto a través de un flujo deberemos convertirlo a una secuencia de bytes
- Esto es lo que se conoce como *serialización*
- Java serializa automáticamente los objetos
  - Obtiene una codificación del objeto en forma de array de bytes
  - En este array se almacenarán los valores actuales de todos los campos del objeto serializado

Objetos serializables

- **Para que un objeto sea serializable debe cumplir:**

1. **Implementar la interfaz `Serializable`**

```
public MiClase implements Serializable {
    ...
}
```

   - **Esta interfaz no obliga a definir ningún método, sólo marca el objeto como serializable**

1. **Todos los campos deben ser**

   - **Datos elementales o**
   - **Objetos serializables**

Flujos de objetos

- **Para enviar o recibir objetos tendremos los flujos de procesamiento**

`ObjectInputStream`
`ObjectOutputStream`

- **Estos flujos proporcionan respectivamente los métodos**

`readObject`
`writeObject`

- **Con los que escribir o leer objetos del flujo**
  - **Utilizan la serialización de Java para codificarlos y descodificarlos**