J2SE
Entrada / Salida
Serialización
Scanner
Properties

Antonio Espín Herranz

ENTRADA / SALIDA

Streams

- Un Stream es un flujo de datos, una conexión entre nuestra aplicación y un origen / destino de los datos.
- Es un tipo de conexión es en serie, es decir, carácter a carácter.
- Todas las clases de entrada / salida están en el paquete: **java.io**

Entrada / Salida estándar

- Lectura de caracteres:
  - Debemos usar la clase Reader y sus derivadas.
  - Para el teclado:

```
// Para leer carácter a carácter.
InputStreamReader teclado = new InputStreamReader(System.in);

// Para leer frases enteras.
BufferReader teclado = new BufferedReader(new InputStreamReader(System.in));
```

Entrada / Salida estándar

- Ejemplo para leer líneas por pantalla:

```
import java.io.*;
public class TestTeclado {
    public static void main(String args[]){
        BufferedReader teclado = new BufferedReader(new
        InputStreamReader(System.in));
        String cadena="";
        do {
            try {
                cadena = teclado.readLine();
                System.out.println("Leído: " + cadena);
            } catch (IOException e){}
        } while (!cadena.equalsIgnoreCase("FIN"));
    }
}
```

Entrada / Salida sobre Ficheros

- Lo gestionan las clases:
  - FileInputStream → para leer.
  - FileoutputStream → para escribir.

  A partir de la clase File / FileDescriptor / String

  Con String le damos el nombre del fichero en windows con la ruta completa:

  “C:\trabajos\prueba.txt”

Entrada / Salida sobre Ficheros

- Para escribir en un fichero: // Tratamos los datos como bytes.

```
try {
    // Abrir el fichero.
    FileOutputStream fEscritura = new     FileOutputStream("Entrada.dat");

    // Para escribir en el fichero:
    String cadena = "datos que quiero escribir"
    fEscritura.write(cadena.getBytes());
    fEscritura.write("\n".getBytes());

    //Para cerrar el fichero:
    fEscritura.close();
} catch (IOException e){
    System.out.println("ERROR de E/S");
}
```

Entrada / Salida sobre Ficheros

- Para leer de un fichero: // Tratamos los datos como bytes.

```
try {
    // Abrir el fichero.
    FileInputStream fLectura = new  FileInputStream("Entrada.dat");
    // Leer del fichero:
    while (fLectura.available() != 0){ // Devuelve el número de bytes disponibles
            System.out.println((char)fLectura.read());
    }
    // Cerrar el fichero:
    fLectura.close();
} catch (IOException e){
    System.out.println("ERROR E / S");
}

// VER LAS CLASES: DataInputStream / DataOutputStream.
```

Sistema de archivos, la clase File

- La clase **File** nos permite acceder al sistema de archivo del S.O.
- Nos permite realizar operaciones con ficheros y directorios, crear, eliminar, renombrar, sobre el separador del path.
- Podemos listar el contenido de un directorio.

Métodos de la clase File

- char pathSeparatorChar(); // El separador.
- boolean canRead(); // Permiso lectura.
- boolean canWrite(); // Permiso escritura.
- boolean exists(); // Existe el fichero en el path.
- boolean isFile(); // Es un fichero.
- boolean isDirectory(); // Es un directorio.
- boolean isHidden(); // Está oculto.
- File [] listRoot(); // Devuelve un array con el contenido de la carpeta, ficheros y directorios.

Ficheros de Acceso Directo

- Representa un fichero de Acceso Directo.
- Implementa los interfaces: DataInput / DataOutput

- Constructores:
  - RandomAccessFile(File objFile, String acceso)
  - RandomAccessFile(String fichero, String acceso)
    - Acceso → “r” Lectura | “rw” Lectura / Escritura.

Métodos de RandomAccessFile

- close() → Cierra el fichero.
- seek(long pos) → mueve el puntero de lectura.
- void write(int b) → Escribir en el fichero.
- int read() → Leer del fichero.

PRÁCTICAS: Entrada / Salida

SERIALIZACIÓN

Serialización

- Se basa es la escritura y lectura de objetos en flujos.
- ¿Para que sirve? Podemos guardar el estado de nuestra aplicación cuando finaliza.
- Y cuando arrancamos de nuevo nuestra aplicación podemos recuperar el último estado.

Serialización

- Para que un objeto pueda almacenar su estado en un flujo, la clase debe implementar el interface Serializable.Ejemplo:

```
import java.io.*;
public class MiClase implements Serializable {…}
```

- Cuando un atributo de la clase nos interesa que no se serialice lo declararemos como transient.
  - public transient tipo nombreAtributo;

Métodos y Clases

- Las clases que permiten almacenar y recuperar el estado de un objeto son:
  - ObjectInputStream
  - ObjectOutputStream

- Y los métodos:
  - public Object readObject();
  - public void writeObject(Object unObjeto);

PRÁCTICAS: Serialización

Ejemplo: Serialización

```
MyClass object1 = new MyClass("Hello", -7, 2.7e10);
    System.out.println("object1: " + object1);
    FileOutputStream fos = new FileOutputStream("serial");
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(object1);
    oos.flush();
    oos.close();
```

```
// La declaración de la clase MyClass:
public class MyClass implements Serializable {
        private String s;
        private int i;
        private double d;
        // Constructores, get / set
}
```

Ejemplo: Deserialización

```
MyClass object2;
    FileInputStream fis = new FileInputStream("serial");
    ObjectInputStream ois = new ObjectInputStream(fis);

    // Ojo: CASTING a mi Clase. readObject me devuelve un objeto de la Clase Object
    object2 = (MyClass)ois.readObject();
    ois.close();
    System.out.println("object2: " + object2);
```

La Clase Scanner

La clase Scanner

- A partir de java 1.5.
- Se encuentra en el paquete java.util.
- Se utiliza para leer ficheros y también leer datos de teclado.
- Procesar un String palabra a palabra.

Ejemplo

```
package es.practica;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Main {
	public static void main(String[] args) {
		Scanner fichero = null;
		String datos;
		try {
			fichero = new Scanner(new File("fichero.txt"));
			while (fichero.hasNextLine()){
				datos = fichero.nextLine();
				System.out.println(datos);
			}

		} catch (FileNotFoundException e) {
			System.out.println("ERROR 1: " + e.getMessage());
		} finally {
			try {
				fichero.close();
			} catch (Exception e){
				System.out.println("ERROR 2: " + e.getMessage());
			}
		}

	}
}
```

Para leer de Teclado

- Scanner s = new Scanner(System.in);

- Después utilizar el método que corresponda para leer de teclado.
  - s.nextLine() → Para leer un String.
  - s.nextInt() → Para leer un int, así sucesivamente.

Properties

- En el mundo java es muy habitual utilizar ficheros de propiedades para almacenar parámetros de configuración.
  - Por ejemplo: los parámetros de conexión a una BD.

- Para ello tenemos la clase **Properties** en el paquete **java.util**.

El formato de los ficheros

- Pares de clave=valor.
  - nombre=Andres
  - apellidos=Sanz Perez

- Estos ficheros tienen extensión .**properties**.

Utilización

```
Properties prop;
prop = new Properties();
try {
    prop.load(new
      FileInputStream("src/propiedades.properties"));
    System.out.println("Nombre: " +
      prop.getProperty("nombre"));
    System.out.println("Apellidos: " +
      prop.getProperty("apellidos"));

} catch (IOException e) {
    e.printStackTrace();
}
```

System.getProperties()

- Dentro de la clase System disponemos de un método especial que nos devuelve propiedades del sistema en un objeto **Properties**.

- Podemos obtener todas las variables del sistema utilizando una **Enumeration** o un **Set**.

Ejemplo con un Set

```
Properties p = new Properties();
String clave;
p = System.getProperties();
System.out.println("Listado de Propiedades usando un Iterator
    ...");
Set conjunto = p.keySet();
Iterator it = conjunto.iterator();
int i = 1;
while (it.hasNext()){
      clave = it.next().toString();
      System.out.println(i + ") " + clave + " = " + p.get(it.next()));
      i++;
}
```

Ejemplo: Enumeration

```
System.out.println("\nListado de Propiedades, usando una
    Enumeration ...");
Enumeration en = p.keys();
i = 1;
String valor;

while (en.hasMoreElements()){
    valor = (String)en.nextElement();
    System.out.println(valor + " = " + p.get(valor));
    i++;
}
```