Entrada y salida de datos

Samsung
TECH INSTITUTE

Contenido

- Entrada y salida de datos.
- El paquete java.io
- Ficheros. La clase File
- Flujos binarios
- Flujos de caracteres
- La clase Scanner

Entrada y salida de datos (I/O)

- La entrada o salida de datos se refiere a la transferencia de datos entre un programa y los dispositivos de almacenamiento o de comunicación.
- La entrada se refiere a los datos que recibe el programa y la salida a los datos que transmite.
- La comunicación puede ser con un usuario humano, mediante un dispositivo físico (pantalla, teclado, impresora, …), con otro sistema (s. de ficheros o programas) dentro del mismo computador o en un computador remoto utilizando sockets, urls, …

El paquete java.io

- Este paquete proporciona lo necesario para la gestión de las entradas y salidas de datos de un programa (a través de flujos).

- Está constituido por una serie de interfaces y clases destinadas a definir y controlar el sistema de ficheros, los distintos tipos de flujos y las serializaciones de objetos.

Ficheros

- La forma de mantener información permanente en computación es utilizar ficheros (archivos).
- Un fichero contiene una cierta información codificada que se almacena en una memoria interna o externa como una secuencia de bits.
- Cada fichero recibe un nombre (posiblemente con una extensión) y se ubica dentro de un directorio que forma parte de una cierta jerarquía.
- El nombre y la ruta, o secuencia de directorios, que hay que atravesar para llegar a la ubicación de un fichero identifican a dicho fichero de forma unívoca.

La clase `File`

- Representa **caminos abstractos** (independientes del S.O.) dentro de un sistema de ficheros.
- Un objeto de esta clase contiene información sobre el nombre y el camino de un fichero o de un directorio.
- Constructores:

```
File(File padre, String fichero)
File(String padre, String fichero)
File(String rutaFichero)
```

- Los objetos de esta clase se pueden crear para directorios y ficheros que ya existan o que no existan.

`File`. Métodos de instancia

- Relativos al nombre y la ruta
  - `String getName()`
  - `String getAbsolutePath()`
  - `String getPath()`
  - `String getParent()`
  - `boolean isAbsolute()`
  - `boolean renameTo(File nuevoNombre)`

Operaciones con objetos File

- **Creación** – Para poder almacenar información en un fichero, previamente hay que crearlo. En el proceso de creación se registra la información necesaria para que el sistema pueda localizar el fichero y manipularlo (y otra información adicional).
- **Eliminación** – Esta operación normalmente elimina la entrada correspondiente al fichero en el directorio en el que esté, imposibilitando así su acceso por parte del S.O.

  - Métodos para crear/consultar ficheros y directorios:

```
boolean createNewFile()
boolean mkdir()
boolean mkdirs()
boolean exists()
boolean isFile()
boolean isDirectory()
```

  - Método para eliminar un fichero:

```
boolean delete()
```

`File`. Métodos para directorios

- Métodos para listar directorios

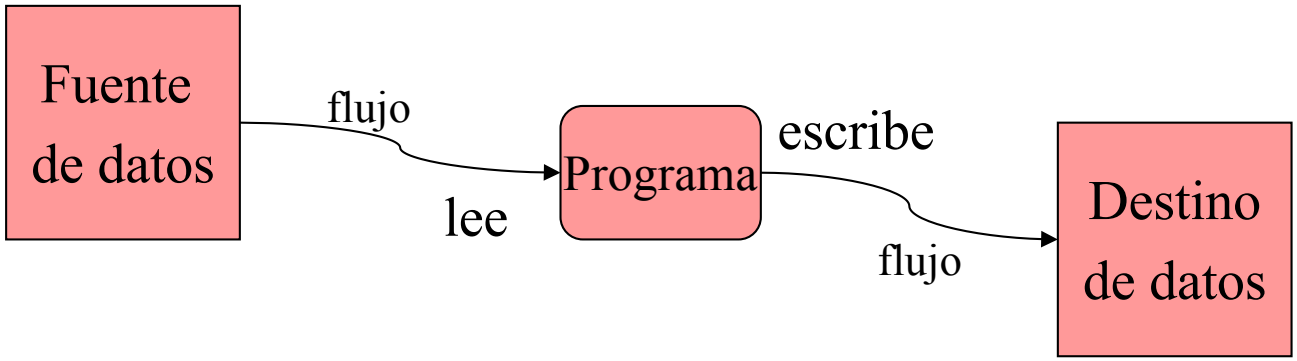
```
String[] list()
String[] list(FilenameFilter)
File[] listFiles()
File[] listFiles(FilenameFilter)
File[] listFiles(FileFilter)
....
interface FilenameFilter {
  boolean accept(File dirActual, String ent);
}
interface FileFilter {
  boolean accept(File path);
}
```

Flujos en Java

- En computación, un flujo de datos o **stream** es una secuencia de datos codificados que se utiliza para transmitir o recibir información.
- Java utiliza la noción de flujo como **objeto** que media entre una fuente o un destino y el programa.
- Java distingue entre:
  - **flujos de entrada y flujos de salida** (s/ sentido de circulación)
  - **flujos de texto y flujos binarios** (s/ codif. 16 u 8 bits)
  - **flujos primarios (iniciales) y flujos secundarios (envoltorios)**

Entrada/Salida basada en flujos

- Esquema de funcionamiento:



- Fuentes y destinos:
  - array de bytes,
  - fichero,
  - pipe,
  - conexión de red,
  - consola …

(Los flujos siempre conectan con un dispositivo físico)

Flujos de texto (caracteres)

Lectura de un flujo. Opción Scanner

- Se pueden construir objetos Scanner sobre objetos **`String`**, **`File`** y otros.
- Métodos de instancia:

```
boolean hasNextLine()
String nextLine()
```

Lectura de fichero

1) Crear un **`File`** sobre un nombre de fichero

```
File file= new File("datos.tex");
```

2) Crear un **`Scanner`** sobre el **`File`** anterior

```
Scanner sc = new Scanner(file);
```

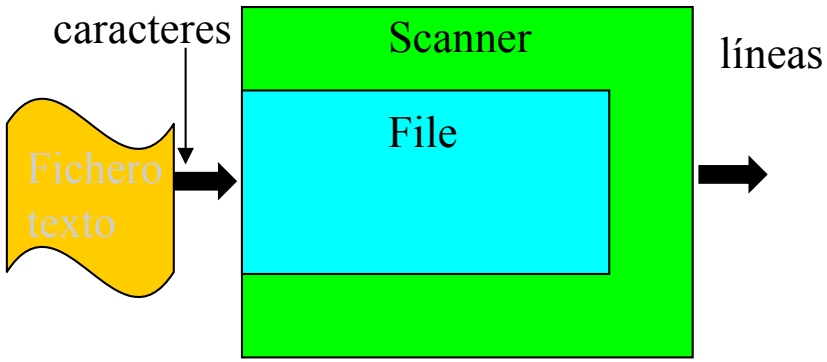
3) Leer líneas con **`hasNextLine()`** y **`nextLine()`** las veces que se necesite

```
while (sc.hasNextLine())
    String linea = sc.nextLine();
```

4) Cerrar el **`Scanner`**

```
sc.close();
```

Si se crea en try no hay que cerrarlo



Ejemplo

```
import java.io.*;
public class Ejemplo {
	public static void main(String[] args) throws IOException {
		if (args.length == 0) {
			System.out.println("ERROR: falta el nombre del fichero");
		} else  {
		// leer el fichero de palabras y mostrarlas en pantalla línea a línea
			Scanner sc = new Scanner(new File(args[0]));
			while (sc.hasNextLine()) {
				System.out.println(sc.nextLine());
			}
			sc.close();
		}
	}
}
```


38

Ejemplo

```
import java.io.*;
public class Ejemplo {
	public static void main(String[] args) throws IOException {
		if (args.length == 0) {
			System.out.println("ERROR: falta el nombre del fichero");
		} else  {
		// leer el fichero de palabras y mostrarlas en pantalla línea a línea
			try (Scanner sc = new Scanner(new File(args[0]))) {
				while (sc.hasNextLine()) {
					System.out.println(sc.nextLine());
				}
			}
		}
	}
}
```

Mejor así (con try).
Cierre automático

39

Ejemplo

```
import java.io.*;
import java.util.*;
public class Ejemplo {
        public static void main(String[] args) {
            // leer el fichero de palabras y mostrarlas en pantalla línea a línea
                try (Scanner sc = new Scanner(new File(args[0]))) {
                        while (sc.hasNextLine()) {
                                System.out.println(sc.nextLine());
                        }
                } catch (ArrayIndexOutOfBoundsException e) {
                        System.out.println("ERROR: falta el nombre del fichero");
                } catch (IOException e) {
                        System.out.println("ERROR: no se puede leer del fichero");
                }
        }
}
```

Mejor todavía (con try y capturando las excepciones)

40

```
ackage prTema6;
mport ...
```

Juan García,23
Pedro González:,15
Luisa López-19
Andrés Molina,-22

```
ublic class SPersona {
	public static Set<Persona> lee(String fichero) throws FileNotFoundException {
		Set<Persona> sp ;
		try (Scanner sc = new Scanner(new File(fichero))) {
			sp =lee(sc) ;
		}
		return sp;
	}

	public static Set<Persona> lee(Scanner sc) {
	    Set<Persona> personas = new HashSet<>();
	    while (sc.hasNextLine()) {
		String linea = sc.nextLine();
		try (Scanner scLinea = new Scanner(linea)) {
		    scLinea.useDelimiter("[,:-]+");
		    String nombre = scLinea.next ();
		    int edad = scLinea.nextInt();
		    Persona persona = new Persona(nombre, edad);
		    personas.add(persona);
		} catch (InputMismatchException e) {
		    throw e;
		    // throw new InputMismatchException(”Error de fichero. Número erróneo");
		} catch (NoSuchElementException e) {
		    throw e;
		    //throw new NoSuchElementException(”Error de fichero. Faltan datos");
		}
	    }
	    return personas;
	}
```

La clase `PrintWriter`

- Permite imprimir representaciones con formato de objetos y tipos básicos de Java sobre flujos de salida de caracteres

```
PrintWriter(File f)
PrintWriter(String nf)
PrintWriter(Writer sal)
PrintWriter(Writer sal, boolean flush)
PrintWriter(OutputStream sal)
PrintWriter(OutputStream sal, boolean flush)
```

- Métodos de instancia:

  Para imprimir todos los tipos básicos y objetos

  `print(...)` `println(...)` `printf(...)`

- Sus métodos no lanzan **`IOExeption`**

Escritura de fichero de texto: opción 4ª

1) Crear un **`PrintWriter`** directamente sobre un fichero

```
PrintWriter pwF = new PrintWriter("datos.tex");
```

Automáticamente se crea un **`FileOutputStream`** seguido de un **`OutputStreamWriter`** con decodificación por defecto

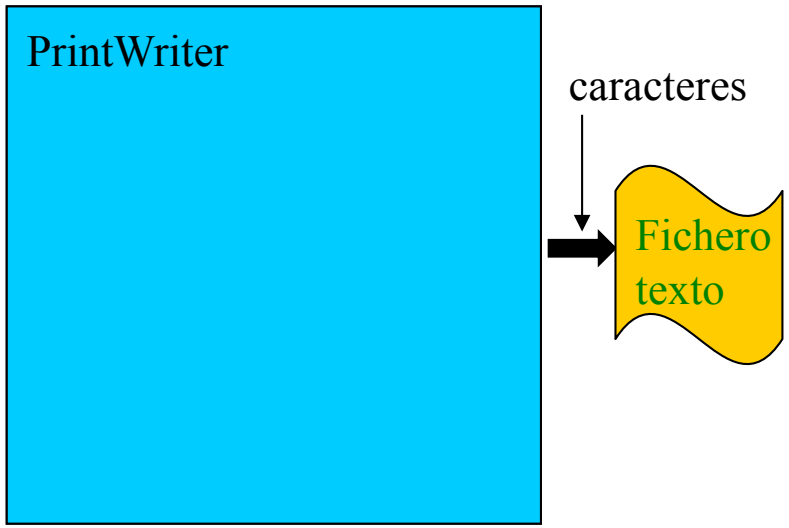
2) Escribir

```
pwF.println(”Hola a todos”);
```

3) Cerrarlo

```
pwF.close();
```

Si se crea en try no hay que cerrarlo



Ejemplo

```
import java.io.*;
public class Ejemplo {
	public static void main(String[] args) throws IOException {
		if (args.length == 0) {
			System.out.println("ERROR: falta el nombre del fichero");
		} else  {
			// crear un fichero de palabras
			PrintWriter pw = new PrintWriter(args[0])) ;
			pw.println("amor roma mora ramo");
			pw.println("rima mira");
			pw.println("rail liar");
			pw.close();
			}
		}
	}
}
```


47

Ejemplo

```
import java.io.*;
public class Ejemplo {
        public static void main(String[] args) throws IOException {
                if (args.length == 0) {
                        System.out.println("ERROR: falta el nombre del fichero");
                } else  {
                        // crear un fichero de palabras
                        try (PrintWriter pw = new PrintWriter(args[0])) {
                                pw.println("amor roma mora ramo");
                                pw.println("rima mira");
                                pw.println("rail liar");
                        }
                }
        }
}
```

Mejor así (con try).
Cierre automático

48

Ejemplo

```
import java.io.*;
public class Ejemplo {
    public static void main(String[] args)  {
        // crear un fichero de palabras
        try (PrintWriter pw = new PrintWriter(args[0])) {
                pw.println("amor roma mora ramo");
                pw.println("rima mira");
                pw.println("rail liar");
        } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("ERROR: falta el nombre del fichero");
        } catch (IOException e) {
                System.out.println("ERROR: no se puede leer del fichero");
        }
    }
}
```

Mejor todavía (con try y capturando las excepciones)

49

```java
package prTema6;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;

public class TestBufferedFile {
	public static void main(String[] args) throws IOException {
		if (args.length == 0) {
			System.out.println("ERROR: falta el nombre del fichero");
		} else {// imprimir un fichero de palabras
			try (PrintWriter pw = new PrintWriter(args[0])) {
				pw.println("amor roma mora ramo");
				pw.println("rima mira");
				pw.println("rail liar");
			}
			// leer el fichero de palabras
			try (Scanner sc = new Scanner(new File(args[0]))) {
				while (sc.hasNextLine()) {
					System.out.println(sc.nextLine());
				}
			}
		}
	}
}
```

```java
package prTema6;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;

public class TestBufferedFile2 {
	public static void main(String[] args) throws IOException {
		if (args.length == 0) {
			System.out.println("ERROR: falta el nombre del fichero");
		} else {// imprimir un fichero de palabras
			try (PrintWriter pw = new PrintWriter(args[0])) {
				pw.println("amor roma mora ramo");
				pw.println("rima mira");
				pw.println("rail liar");
			}
			// leer el fichero de palabras mostrando palabra a palabra
			try (Scanner sc = new Scanner(new File(args[0]))) {
				while (sc.hasNextLine()) {
					try (Scanner scLinea = new Scanner(sc.nextLine())) {
						while (scLinea.hasNext()) {
							System.out.println(scLinea.next());
						}
					}
				}
			}
		}

	}
}
```