Portfolio: Entrada y Salida de Información en Java

Jhonal Roca

1º DAW - 24/25

Interfaces de Entrada y Salida

En términos generales, las interfaces permiten el intercambio de datos entre un programa y su entorno. Se dividen en:

- **Interfaces de Entrada:** Reciben datos (por ejemplo, leer desde teclado).
- **Interfaces de Salida:** Envían datos (por ejemplo, mostrar en pantalla).

Flujos de Datos en Java

Los flujos de datos (*streams*) son secuencias ordenadas de información que se transmiten desde un origen hasta un destino. Según su dirección, pueden ser:

- **Flujos de Entrada:** Se utilizan para leer información.
- **Flujos de Salida:** Se usan para escribir información.

También se clasifican por el tipo de acceso:

- **Acceso Secuencial:** Los datos se leen o escriben en orden.
- **Acceso Directo:** Se puede acceder a cualquier dato sin recorrer los anteriores.

```
System.out.println("Hola, mundo!");  // Escribe en la salida estándar
```

Clase `Scanner` para Entrada de Datos

La clase **`Scanner`** facilita la lectura de datos desde la entrada estándar.

Ejemplo:

```
6
7    Scanner sc = new Scanner(System.in);
8    System.out.print("Introduce tu nombre: ");
9    String nombre = sc.nextLine();
10   System.out.println("Hola, " + nombre + "!");
11
```

Ficheros y Directorios

Guardar información en archivos permite la persistencia de datos más allá de la ejecución del programa. Java usa la clase **`File`** para gestionar ficheros y directorios.

Ejemplo de creación de un archivo:

```
File archivo = new File("datos.txt");
if (archivo.createNewFile()) {
    System.out.println("Archivo creado con éxito.");
} else {
    System.out.println("El archivo ya existe.");
}
```

Serialización: Guardando Objetos en Archivos

La **serialización** permite convertir un objeto en una secuencia de bytes para almacenarlo o enviarlo.

Ejemplo de serialización:

```
ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("objeto.dat"));
out.writeObject(new Persona("Fran", 25));
out.close();
```

Para deserializar el objeto:

```
ObjectInputStream in = new ObjectInputStream(new FileInputStream("objeto.dat"));
Persona persona = (Persona) in.readObject();
in.close();
System.out.println("Nombre: " + persona.getNombre());
```

XML en Java

XML es un formato estructurado para almacenar información. Java proporciona APIs como DOM para manipular archivos XML.

Ejemplo de lectura de un XML:

```
35  DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
36  DocumentBuilder builder = factory.newDocumentBuilder();
37  Document document = builder.parse(new File("datos.xml"));
38  NodeList nodos = document.getElementsByTagName("persona");
39  System.out.println("Personas en el XML: " + nodos.getLength());
40
```

Entrada y Salida de Información en Java

Concepto	Descripción	Ventajas	Desventajas
Flujos de datos(Stream)	Secuencia ordenada de datos entre un origen y un destino	Permiten manejar archivos, entrada de usuarios y más	Puede ser complicado en accesos grandes o sin control de excepciones
Clase(Scanner)	Permite la entrada de datos desde consola	Fácil de usar flexible	No es eficiente para grandes cantidades de datos
Clase(file)	Representa archivos y directorios en el sistema	Facilita la manipulación de ficheros	No crea ni abre archivos, solo los representa
Serelizacion	Guarda objetos en archivos binarios	Permite el almacenamiento persistente de objetos	Require implementar Serializable y puede aumentar el tamaño del archivo
XML con DOM	Manejo de archivos xml con estructura de árbol	Útil para estructurar y transmitir datos fácilmente	Puede ser más lento en documentos grandes comprando con otras tecnologías