Persistencia

Salvador López Mendoza

Diciembre de 2022

Introducción

Los datos con los que trabajan los programas desarrollados hasta ahora no permanecen de una ejecución del programa a otra.

En muchos programas, como la nómina o el mantenimiento de la colección de obras, es importante que los datos permanezcan más allá de la ejecución del programa.

En la mayoría de los lenguajes de programación existen instrucciones para almacenar (y recuperar) datos primitivos.

¿Cómo *almacenar* los resultados que se generan en nuestros programas?

¿Cómo *recuperar* la información que ha sido almacenada previamente?

Información persistente

En los sistemas de cómputo se cuenta con medios de almacenamiento persistente (discos magnéticos, discos de estado sólido, USB).

Es necesario poder distinguir a un elemento de otro.

En el caso de las computadoras de propósito general la información se almacena en **archivos**.

Cada archivo debe ser identificado en forma única. Generalmente se realiza mediante la asignación de *nombres únicos*.

La mayoría de los sistemas de archivos actuales utiliza una *organización jerárquica* para facilitar el almacenamiento de muchas cosas.

¿Es posible almacenar los resultados de un programa en un archivo?

Texto persistente

Todo lo que se maneja en Java se puede expresar como una **cadena de caracteres** (usando `toString`).

Los tipos de datos primitivos se expresan como cadenas de caracteres.

Al definir cada clase de objetos se debe definir el método `toString`.

¿Cómo *almacenar* (y después *recuperar*) cadenas de caracteres?

¡Cuidado!

El texto es una representación visual de los objetos. Almacenar y recuperar las cadenas de texto no es almacenar y recuperar los objetos.

Archivos de texto en Java

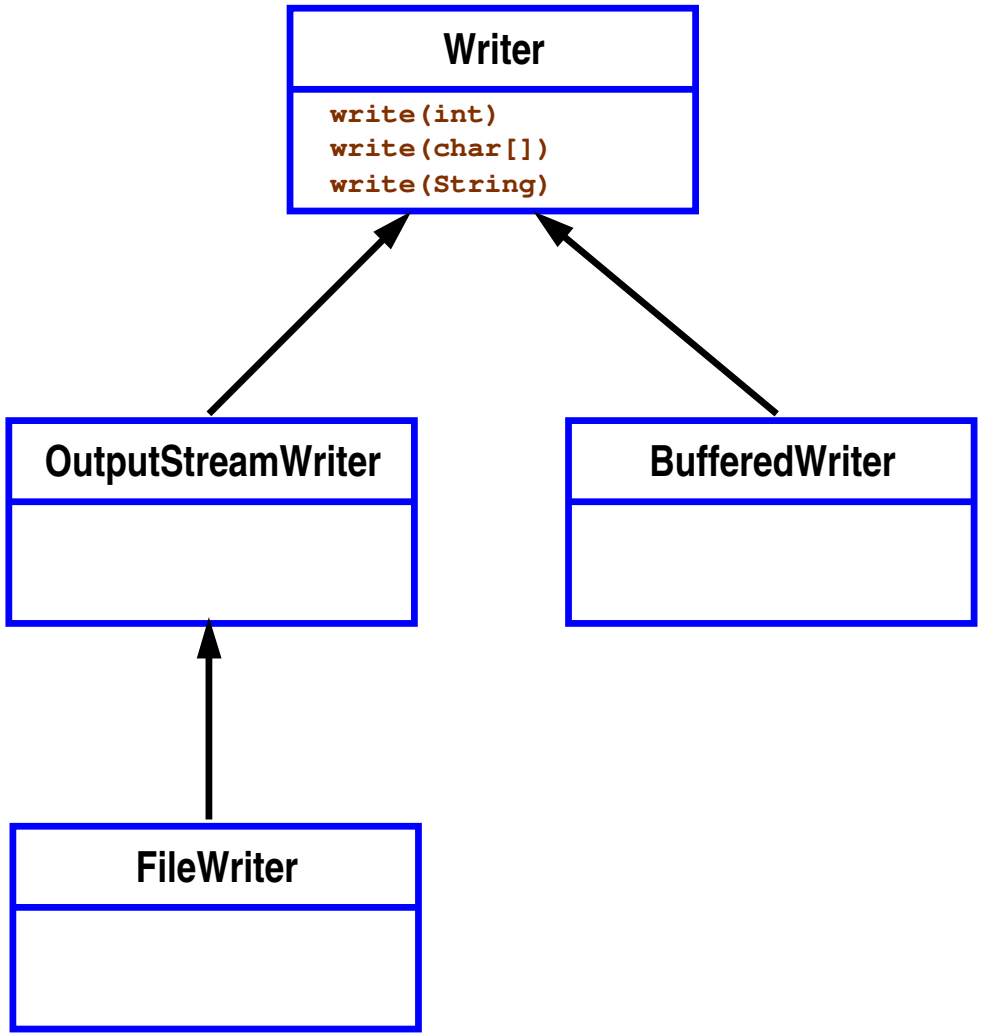
Java cuenta con clases para trabajar con archivos.

Las clases principales son `Writer` y `Reader`. Son clases abstractas.

Writer Define los métodos : `write(int)`, `write(char[])`, `write(String)`, `close()`.
`OutputStreamWriter`. Es una subclase. `FileWriter` es subclase de la subclase.
`BufferedWriter`. Es otra subclase.

Reader Define los métodos: `read()`, `read(char[])`, `skip(long)`, `close()`.
`InputStreamReader`. Es una subclase. `FileReader` es subclase de la subclase.
`BufferedReader`. Es otra subclase.

Archivos de texto en Java (II)



Objetos serializables

La serialización de objetos permite escribir objetos a archivos con una sola instrucción, con lo cual quedan grabados hasta que se decida eliminarlos o modificarlos.

Lo que se almacena es el **estado del objeto**.

También permite recuperar los objetos grabados en archivos.

La serialización de un objeto consiste en generar una secuencia de bytes lista para su almacenamiento o transmisión.

Mediante el proceso inverso, se puede recuperar el estado original del objeto.

Guardar un objeto, para que pueda existir incluso cuando la aplicación finaliza, se conoce como **persistencia de objetos**.

Interfaz Serializable

Para que objetos de una clase puedan serializarse, es decir hacerse persistentes, es necesario que la clase de tales objetos implemente la interfaz `Serializable`.

La interfaz `Serializable` tiene el siguiente código:

```
public interface Serializable {
}
```

Esta interfaz no define métodos, por tal razón es conocida como una *interfaz de marcado*.

Cualquier clase que implemente esta interfaz puede grabar y leer objetos de un archivo sin mayor intervención por parte del programador

Serialización

Para que un objeto sea serializable **todas los atributos** de la estructura **deben ser serializables**.

Todos los tipos primitivos, los objetos de la clase `String` y algunos de otras clases de Java son serializables.

```
import java.io.Serializable;

public class Persona implements Serializable {
  private String nombre;
  private int peso;
  private double estatura;
  ...
}
```

Para almacenar objetos serializados se requiere trabajar con un *archivo*.

Trabajo con archivos

Tareas principales:

- Abrir el archivo.
- Leer o escribir el objeto.
- Cerrar el archivo.

En cada operación pueden existir errores. No siempre son causados por el programador.

Recomendación

Los métodos que implementan estas tareas deben tratar con las excepciones correspondientes.

Trabajo con archivos. Escritura

Para grabar objetos en un archivo, es decir para serializar, se requiere un objeto de la clase `ObjectOutputStream`.

```
ObjectOutputStream obj = new ObjectOutputStream(...) ;
```

Es un flujo (de salida) de objetos persistentes.

El flujo de objetos se debe asociar con algún archivo para que los objetos queden almacenados en el sistema de archivos de la computadora.

El archivo con el que se trabaja se especifica como parámetro del constructor del flujo de objetos:

```
ObjectOutputStream obj = new ObjectOutputStream(
                           new FileOutputStream(nombreArch));
```

Trabajo con archivos. Escritura (II)

Una vez creado el objeto de la clase `ObjectOutputStream` se puede utilizar el método `writeObject(objeto)` para grabar el objeto que toma como parámetro.

Si el objeto que se intenta grabar no es de una clase que implemente la interfaz `Serializable` se dispara la excepción `NotSerializableException`.

Al manejar archivos es conveniente considerar (y atrapar) las posibles excepciones.

Trabajo con archivos. Lectura

Para recuperar objetos almacenados en un archivo se requiere un objeto de la clase `ObjectInputStream`.

```
ObjectInputStream lector = new ObjectInputStream(...) ;
```

Es un flujo (de entrada) de objetos.

El flujo de objetos se debe asociar con algún archivo para que los objetos se recuperen del sistema de archivos de la computadora.

El archivo con el que se trabaja se especifica como parámetro del constructor del flujo de objetos:

```
ObjectInputStream lector = new ObjectInputStream(
                           new FileInputStream(nombreArch));
```

Trabajo con archivos. Lectura (II)

Una vez creado el objeto de la clase `ObjectInputStream` se puede utilizar el método `readObject()`. Este método regresa un objeto.

El método `readObject()` no sabe a qué clase corresponde el objeto que regresará. Está definido para regresar un objeto de la clase `Object`.

Se aprovechan las propiedades de la herencia para que pueda regresar objetos de cualquier clase.

```
...
Persona otra ;
...
otra = (Persona) lector.readObject() ;
```

Trabajo con archivos (IV)

Al terminar de serializar todos los objetos se debe llamar al método `close` para asegurar que no se pierdan los objetos grabados en el archivo especificado.

Independientemente de que haya habido un error o no, es necesario cerrar el archivo.

Como la recomendación es que el manejo de archivos considere las posibles situaciones excepcionales, el bloque de instrucciones que requiere de operaciones con el archivo debe estar protegido en una claúsula `try`. Así que conviene incluir la operación para cerrar el archivo en la claúsula `finally`.

Archivos de texto

En muchas ocasiones lo que se necesita es procesar o generar *información textual*.

La captura de datos se ha realizado introduciendo texto, ayudados por la clase `Scanner`.

El parámetro de la clase `Scanner` es el nombre de un archivo. Se puede crear un archivo de texto con los datos que requiere un programa y al crear el `Scanner` indicar el archivo. Las operaciones de lectura tomarán los datos del archivo.

En `Java` existen muchas formas de acceder a información en archivos.

Archivos CSV

Cada programa, por ejemplo una hoja de cálculo, almacena la información en un formato especial, propio de la aplicación.

Muchos programas tienen una opción para generar un archivo de datos como texto y en el que la información aparece por columnas.

- Para identificar el lugar en el que inicia cada columna se utiliza un *caracter especial para separar columnas*.
  El caracter más usado es el caracter **coma**.

A los archivos cuya información se almacena en líneas de texto, separadas por comas, se les llama **archivos con valores separados por comas (CSV)**.

Manejo de archivos CSV

Alternativas para acceder a información en archivos CSV:

- Leer el archivo de texto, línea por línea, y que el programador revise la información, caracter por caracter, hasta encontrar el caracter de separación, en ese momento tiene a su disposición un elemento de información.
  El elemento reconocido debe transformarse a lo que se necesite (una secuencia de dígitos corresponden a un valor numérico).
- Leer el archivo de texto, línea por línea, usando los métodos de la clase `Scanner` para obtener cada elemento en la representación correcta (por ejemplo, usando `nextInt()`).
- Usar una clase especial para manejar archivos CSV.
  Para `Java` se han desarrollado muchas clases que procesan archivos CSV.
  Hay un paquete muy usado: `OpenCSV`.

Archivos CSV. Detalles

Aspectos a considerar al manipular archivos CSV:

- El caracter que se usa para separar columnas puede ser distinto a una *coma (,)*.
  En muchas ocasiones se usan otros separadores, sobre todo cuando la coma es parte de los datos por procesar.
- La cantidad de datos puede ser muy grande.
  Puede ser más eficaz leer todos los datos y procesarlos independientemente.
  Conviene usar arreglos.
  ¿De qué tamaño debe ser el arreglo?
- El archivo CSV puede incluir un renglón adicional (al inicio) con los nombres de las columnas.