Acceso a datos

Gestión de Información en Ficheros XML

Juan Carlos Filter Martin

2º DAM

RESUMEN

Este proyecto consiste en una aplicación de consola desarrollada en Java que gestiona una colección de libros almacenados en un archivo XML.

La aplicación permite al usuario realizar diversas operaciones sobre la lista de libros, tales como agregar nuevos libros, listar todos los libros existentes, modificar los datos de un libro específico o eliminar libros de la lista.

Además, incluye la opción de exportar la lista completa de libros a un archivo en formato CSV, facilitando así su uso en otras aplicaciones y formatos. La gestión del archivo XML se realiza mediante la biblioteca JAXB, garantizando un almacenamiento estructurado y fácil de manipular.

ÍNDICE

1. Introducción

Este proyecto tiene como objetivo desarrollar una aplicación en Java para la gestión de una lista de libros almacenados en un archivo XML.

La aplicación permite al usuario llevar a cabo operaciones básicas de gestión de información sobre los libros *"agregar nuevos títulos, modificar datos existentes, eliminar registros, listar todos los libros y exportar la lista de libros a un archivo en formato CSV para facilitar su lectura o procesamiento en otras aplicaciones".*

La idea de este proyecto es ofrecer una solución sencilla y eficiente para la administración de una colección de libros, almacenando la información en un formato estructurado XML. Este formato permite que los datos se gestionen y manipulen fácilmente, al tiempo que ofrece compatibilidad con otros sistemas y herramientas de procesamiento de datos.

Con esta aplicación se pone en práctica el uso de bibliotecas como JAXB para el manejo de XML en Java, así como la técnicas de persistencia, manejo de excepciones y control de flujos en una aplicación de consola.

El proyecto también se enfoca en asegurar una experiencia de usuario fluida mediante un menú en consola que facilita la navegación por las distintas funcionalidades de la aplicación.

2. Requisitos del Sistema

Para ejecutar y desarrollar esta aplicación de gestión de libros en Java, se requieren los siguientes elementos:

Java Development Kit (JDK)

- **Versión utilizada**: JDK 23.

  Asegúrate de tener instalada esta versión o una versión compatible para evitar problemas de compatibilidad.

Entorno de Desarrollo Integrado (IDE)

- **IDE utilizado**: IntelliJ IDEA.

  IntelliJ IDEA proporciona herramientas avanzadas para la gestión de dependencias y la depuración del código, lo que facilita el desarrollo de aplicaciones en Java.

Gestor de Dependencias - Maven

- **Archivo pom.xml**: El proyecto utiliza Maven para gestionar las dependencias, lo cual facilita la inclusión de bibliotecas necesarias (como JAXB) y mantiene un entorno controlado de versiones.

Sistema Operativo

- Compatible con Windows, macOS y Linux, siempre que sea compatible con Java.

3. Estructura del proyecto

El proyecto se organiza en varias clases que manejan distintas responsabilidades en la gestión de información de los libros. A continuación se detalla la estructura y la función de cada clase:

- Clase `Main`

Punto de entrada de la aplicación. Contiene el método `main` que inicia la ejecución del programa y crea una instancia de la clase `GestorLibros`.

- Clase `Libro`

Representa un libro individual. Esta clase contiene atributos como `título`, `autor` y `publicación`, junto con el constructor y sus métodos de acceso (`getters` y `setters`) para manipular estos atributos.

- Clase `Libros`

Contiene una colección de objetos `Libro`. Proporciona métodos para gestionar la lista de libros, como agregar un nuevo libro y obtener la lista actual. Inicializa la lista de libros para asegurarse de que nunca sea `null`.

- Clase `XmlLibroPersistencia`

Maneja la lectura y escritura de datos en un archivo XML. Utiliza JAXB (Java Architecture for XML Binding) para serializar y deserializar objetos de tipo `Libros` al formato XML y viceversa.

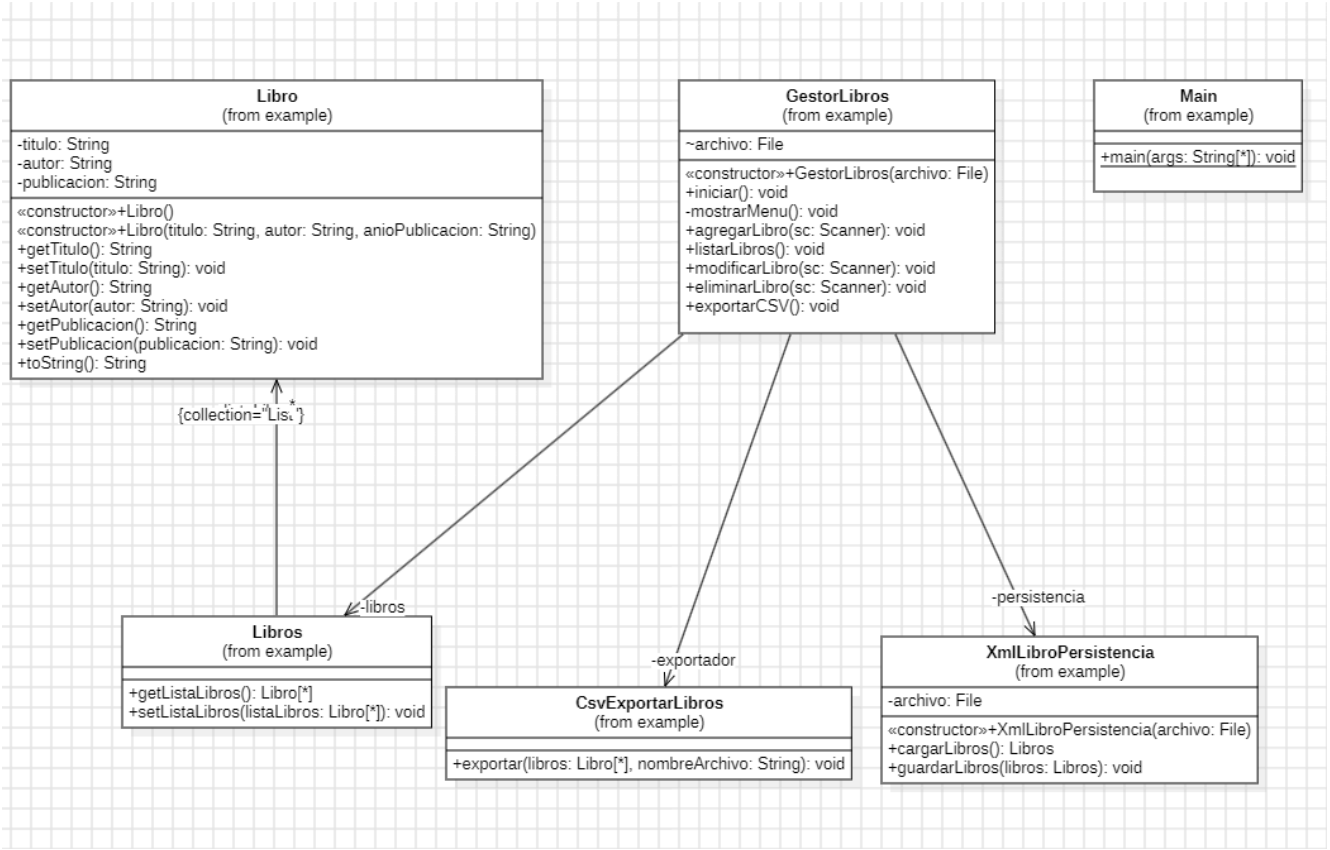
- Clase `CsvExportarLibros`

Proporciona funcionalidad para exportar la lista de libros a un archivo CSV. Incluye métodos para formatear los datos de los libros en un formato adecuado para el archivo CSV.

- Clase `GestorLibros`

Contiene la lógica de la aplicación. Aquí se gestionan las operaciones que el usuario puede realizar, como agregar, listar, modificar y eliminar libros. También se encarga de manejar la interacción con el usuario a través de un menú en la consola y de coordinar el acceso a las otras clases.

4. Diagrama de Clases



5. Funcionalidades

Agregar un nuevo libro

➔ **Proceso de toma de datos del usuario:**

El usuario podrá ingresar los detalles del nuevo libro, incluyendo:

- **Título del libro**: Se solicita al usuario que escriba el título.
- **Autor del libro**: Se pide el nombre del autor.
- **Año de publicación**: Se solicita el año en que se publicó el libro.

Estos datos se capturan utilizando un `Scanner` que lee la entrada desde la consola.

➔ **Almacenamiento en el XML:**

Una vez que el usuario ha ingresado todos los detalles, se crea un nuevo objeto `Libro` utilizando los datos proporcionados.

Este nuevo libro se agrega a una lista de libros (`Libros`).

Se llama al método `guardarLibros()` de la clase `XmlLibroPersistencia`, que utiliza JAXB para serializar la lista actualizada de libros y almacenarla en el archivo XML.

Esto asegura que los cambios se guardan y el archivo XML refleja la lista de libros actualizada.

Listar todos los libros

➔ **Lectura y visualización de libros en consola:**

Se invoca el método `listarLibros()`, que obtiene la lista de libros desde el objeto `Libros`.

El método `cargarLibros()` se utiliza internamente para cargar los libros del archivo XML en caso de que no se haya hecho previamente.

Se verifica si la lista de libros está vacía. Si está vacía, se muestra un mensaje indicando que no hay libros disponibles.

Si hay libros, se recorren en un ciclo `for`, mostrando en la consola:

- El título del libro.
- El autor del libro.
- El año de publicación.

Esta funcionalidad permite al usuario ver todos los libros almacenados de manera organizada.

Modificar un libro

➔ **Actualización de los atributos del libro en el XML:**

Primero, se listan todos los libros para que el usuario pueda elegir cuál desea modificar.

Se solicita al usuario que ingrese el número correspondiente al libro que quiere actualizar.

Se valida la selección, y si es válida, se obtienen los detalles actuales del libro seleccionado.

Se piden los nuevos valores para los atributos del libro (título, autor y año de publicación).

Una vez que se han ingresado los nuevos valores, se actualizan los atributos del libro seleccionado.

Finalmente, se llama al método `guardarLibros()` para serializar la lista de libros actualizada y sobrescribir el archivo XML con los cambios.

Eliminar un libro

➔ **Proceso de eliminación del XML:**

Al igual que en la modificación, se lista primero todos los libros para que el usuario seleccione cuál desea eliminar.

El usuario ingresa el número correspondiente al libro que desea eliminar.

Se valida la selección, y si es válida, se elimina el libro de la lista utilizando el método `remove()` de `List`.

Después de la eliminación, se llama al método `guardarLibros()` para actualizar el archivo XML, serializando la lista de libros sin el libro eliminado.

Esto asegura que la lista de libros se mantenga actualizada y refleja la eliminación en el archivo XML.

Exportar a CSV

➔ **Generación de un archivo CSV a partir de los datos:**

El método `exportarCSV()` se invoca cuando el usuario elige la opción de exportar.

Se utiliza un `CsvExportarLibros`, que tiene un método `exportar()` que toma como parámetros la lista de libros y el nombre del archivo CSV de salida.

Dentro de este método, se utiliza un `BufferedWriter` para crear y escribir en el archivo CSV.

Para cada libro en la lista, se escribe una línea en el formato especificado (título, autor y año de publicación).

Una vez que se han escrito todos los libros, se cierra el `BufferedWriter` automáticamente, asegurando que todos los datos se hayan escrito correctamente.

Se muestra un mensaje de confirmación al usuario, indicando que los libros se han exportado exitosamente al archivo CSV.

6. Manejo de Excepciones

Archivo no encontrado: Cuando la aplicación intenta cargar libros desde un archivo XML que no existe.

- **Manejo**:

En el método `cargarLibros()` de la clase `XmlLibroPersistencia`, se verifica si el archivo existe (`if (archivo.exists())`). Si no existe, la aplicación crea y retorna una nueva instancia de `Libros`.

Esto evita que se lance una excepción y permite a la aplicación seguir funcionando sin interrupciones.

Errores de I/O (Entrada/Salida): Problemas al intentar leer o escribir en el archivo, como falta de permisos de acceso o errores de disco.

- **Manejo**:

En el método `exportar()` de la clase `CsvExportarLibros`, se utiliza un bloque `try-catch` para manejar excepciones de tipo `IOException`.

Si ocurre un error al intentar crear o escribir en el archivo CSV, se captura la excepción y se imprime un mensaje de error en la consola (`System.out.println("Error al exportar a CSV: " + e.getMessage());`).

Esto proporciona información al usuario sobre lo que salió mal sin hacer que la aplicación se detenga abruptamente.

Errores de JAXB: Excepciones lanzadas por JAXB durante el proceso de unmarshalling (lectura) o marshalling (escritura) de XML, como errores de formato o estructura incorrecta del XML.

- **Manejo**:

En el método `cargarLibros()`, se captura la excepción `JAXBException` que puede ocurrir durante el unmarshalling.

Se imprime un mensaje que indica que hubo un error al cargar los libros (`System.out.println("Error al cargar libros: " + e.getMessage());`).

Esto permite que el usuario sepa que el archivo XML puede estar corrupto o mal formado y que se necesita una revisión.

En el método `guardarLibros()`, se captura la excepción `JAXBException` que puede ocurrir durante el marshalling.

Se imprime un mensaje que indica que hubo un error al cargar los libros (`System.out.println("Error al guardar libros: " + e.getMessage());`).

Esto permite que el usuario sepa que el archivo XML puede estar corrupto o mal formado y que se necesita una revisión.

Errores de entrada de usuario: Cuando el usuario ingresa datos no válidos (por ejemplo, un número para seleccionar un libro que no existe).

- **Manejo**:

Cuando el usuario introduce algo por teclado que no es válido el código imprime un mensaje indicando que la elección no es válida (`System.out.println("La elección del libro no es válida.");`).

Errores generales: Cualquier otro tipo de error inesperado que no se haya anticipado.

- **Manejo**:

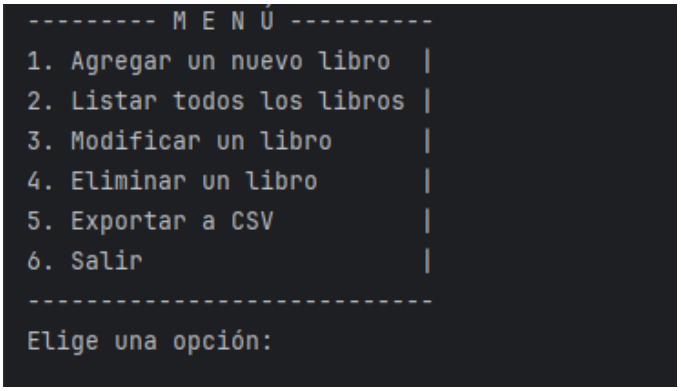
Se utiliza un bloque `try-catch` alrededor de la lógica principal en el método `iniciar()`, lo que permite capturar cualquier excepción no manejada y mostrar un mensaje de error general (`System.out.println("Ocurrió un error: " + e.getMessage());`).

7. Ejemplo de Uso

Ejemplo de Uso en Consola

Iniciar la aplicación

Cuando se inicia la aplicación, se ve lo siguiente en la consola:



Agregar un nuevo libro

Opción 1: Agregar un nuevo libro

Si el usuario elige la opción 1, la consola solicita los detalles del libro:

```
Elige una opción: 1
Título del libro: El principito
Autor del libro: Antoine de Saint-Exupéry
Año de publicación: 1943

--> Libro agregado!: El principito
```

Listar todos los libros

Opción 2: Listar todos los libros

Si el usuario elige la opción 2, se muestran todos los libros almacenados:

```
----- LISTA DE LIBROS ----
--------------------------
Título: LIBRO2
Autor: Juan Carlos
Año de Publicación: 1992
Título: El principito
Autor: Antonie de Saint-Exupery
Año de Publicación: 1943
--------------------------
```

Modificar un libro

Opción 3: Modificar un libro

Si el usuario elige la opción 3, se listan los libros y se solicita la selección para modificar:

```
----- LISTA DE LIBROS ----
-------------------------
Título: LIBRO2
Autor: Juan Carlos
Año de Publicación: 1992
Título: El principito
Autor: Antonie de Saint-Exupery
Año de Publicación: 1943
-------------------------
Selecciona el número del libro que deseas modificar: 2
Nuevo título (actual: El principito): El principito - Nueva edicion
Nuevo autor (actual: Antonie de Saint-Exupery): Antonie de Saint-Exupery
Nuevo año de publicación (actual: 1943): 2020

--> Libro Modificado!
```

Eliminar un libro

Opción 4: Eliminar un libro

Si el usuario elige la opción 4, se listan los libros y se solicita la selección para eliminar:

```
Elige una opción: 4

----- LISTA DE LIBROS ----
-------------------------
Título: LIBRO2
Autor: Juan Carlos
Año de Publicación: 1992
Título: El principito - Nueva edicion
Autor: Antonie de Saint-Exupery
Año de Publicación: 2020
-------------------------
Selecciona el número del libro que deseas eliminar: 1

--> Libro eliminado!
```

Exportar a CSV

Opción 5: Exportar a CSV

Si el usuario elige la opción 5, se genera un archivo CSV con los libros:

```
Elige una opción: 5

--> Libros exportados a libros.csv!
```

Salir de la aplicación

Opción 6: Salir

Si el usuario elige la opción 6, se cierra la aplicación:

```
Elige una opción: 6
Saliendo de la aplicación...

Process finished with exit code 0
```

8. Conclusiones y mejoras

Conclusión:

El proyecto de gestión de libros es una aplicación útil y funcional que permite realizar operaciones básicas como agregar, listar, modificar, eliminar y exportar libros.

A través de la implementación de tecnologías como JAXB para la persistencia en XML y el uso de estructuras de datos en Java, se ha logrado una gestión eficiente de los datos.

Los Aspectos Positivos:

Facilidad de Uso: La interfaz de consola es simple y permite al usuario interactuar fácilmente con las funcionalidades.

Persistencia de Datos: La utilización de archivos XML para almacenar datos garantiza que la información no se pierda.

Modularidad: El diseño basado en clases separadas y métodos facilita el mantenimiento y la reutilización de código.

Mejoras:

Algunas posibles ampliaciones y mejoras para el proyecto incluyen:

Interfaz Gráfica de Usuario (GUI):

Implementar una interfaz gráfica. Esto permitiría una interacción más intuitiva, como botones, menús desplegables…

Sistema de Búsqueda:

Implementar una funcionalidad de búsqueda. Se podría incluir opciones de búsqueda por título, autor o año de publicación.

Mejoras en la Persistencia:

Ampliar la funcionalidad de almacenamiento BD (MySQL, MongoDB, ORACLE …)

Exportación a Otros Formatos:

Permitir la exportación a otros formatos como JSON o PDF. A la hora de exportar que de la opción de elegir en qué formato se desea exportar.

Manejo de Usuarios:

Implementar un logueo de usuarios para gestionar sus propias listas de libros, sería de tal forma que al implementar con las BD haga login a ese usuario concreto con sus datos de libros.

9. Código Fuente

Clase `GestorLibros`

Esta clase se encarga de gestionar la aplicación y manejar las interacciones con el usuario.

```java
public class GestorLibros {  2 usages  Juan Carlos *
    //Instancias y objeto para manejar la persistencia y exportación de libros.
    private XmlLibroPersistencia persistencia;  5 usages
    private CsvExportarLibros exportador;  2 usages
    private Libros libros;  13 usages

    //Constructor
    public GestorLibros(File archivo){  1 usage  Juan Carlos *
        //Inicializar la instancia y cargaLibros al objeto libros
        this.persistencia = new XmlLibroPersistencia(archivo);
        this.exportador = new CsvExportarLibros();
        this.libros = persistencia.cargarLibros();
    }

    //Metodo para iniciar la app
    public void iniciar() {  1 usage  Juan Carlos
        try(Scanner sc = new Scanner(System.in)){
            boolean salir = false;

            while (!salir) {
                mostrarMenu();

                int opcion = sc.nextInt();
                sc.nextLine();

                switch (opcion) {
                    case 1:
                        agregarLibro(sc);
                        break;
                    case 2:
                        listarLibros();
                        break;
                    case 3:
                        modificarLibro(sc);
                        break;
                    case 4:
                        eliminarLibro(sc);
                        break;
                    case 5:
                        exportarCSV();
                        break;
                    case 6:
                        salir = true;
                        System.out.println("Saliendo de la aplicación...");
                        break;
                    default:
                        System.out.println("Opción no válida.");

                }
            }

        } catch (Exception e) {
            System.out.println("Ocurrió un error: " + e.getMessage());
        }
    }

    //Menu
    private void mostrarMenu() {  1 usage  Juan Carlos
        System.out.println("--------- M E N Ú ----------");
        System.out.println("1. Agregar un nuevo libro  |");
        System.out.println("2. Listar todos los libros |");
        System.out.println("3. Modificar un libro      |");
        System.out.println("4. Eliminar un libro       |");
        System.out.println("5. Exportar a CSV          |");
        System.out.println("6. Salir                   |");
        System.out.println("----------------------------");
        System.out.print("Elige una opción: ");
    }
```

Clase `Libros`

Esta clase representa la colección de libros y utiliza anotaciones JAXB para facilitar la conversión entre XML y objetos Java.

```java
//Define la clase como la raíz del XML con el nombre "libros"
@XmlRootElement(name = "libros") 7 usages  Juan Carlos *
public class Libros {
    //Lista para almacenar los objetos libro
    private List<Libro> listaLibros; 4 usages

    //Metodo para obtener la lista de libros, con anotación para cada "libro" en XML
    @XmlElement(name = "libro") 7 usages  Juan Carlos *
    public List<Libro> getListaLibros() {
        //Si es null inicializa la lsita creando una vacía
        if(listaLibros ==null){
            listaLibros = new ArrayList<>();
        }
        return listaLibros;
    }

    //Metodo para actualizar la listaLibros (recoge la lista por parámetros y actualiza la anterior
    public void setListaLibros(List<Libro> listaLibros) { 2 usages  Juan Carlos
        this.listaLibros = listaLibros;
```

Clase `Libro`

Se encarga de representar la entidad "Libro" con los atributos, constructor y los métodos setters y getters

```java
@XmlRootElement(name = "libro") 13 usages  Juan Carlos *
public class Libro {

    private String titulo; 4 usages
    private String autor; 4 usages
    private String publicacion; 4 usages

    public Libro(){}

    public Libro(String titulo, String autor, String anioPublicacion) {...}
    @XmlElement 3 usages  Juan Carlos
    public String getTitulo() { return titulo; }

    public void setTitulo(String titulo) { this.titulo = titulo; }

    @XmlElement 3 usages  Juan Carlos
    public String getAutor() { return autor; }

    public void setAutor(String autor) { this.autor = autor; }

    @XmlElement 3 usages  Juan Carlos
    public String getPublicacion() { return publicacion; }

    public void setPublicacion(String publicacion) { this.publicacion = publicacion; }
```

Clase `XmlLibroPersistencia`

Esta clase maneja la carga y el guardado de libros en un archivo XML utilizando JAXB.

```java
public class XmlLibroPersistencia {  2 usages  Juan Carlos *
    private File archivo; //Archivo donde se almacena los datos de los libros en XML  4 usages

    //Constructor que inicializa la clase con el archivo
    public XmlLibroPersistencia(File archivo) {  1 usage  Juan Carlos
        this.archivo = archivo;
    }

    //Metodo para cargar los libros desde el XML
    public Libros cargarLibros() {  1 usage  Juan Carlos *
        try {
            //Si archivo existe....
            if (archivo.exists()) {
                //Configura JAXBContext para la clase Libros
                JAXBContext context = JAXBContext.newInstance(Libros.class);
                //Crea el objeto Unmarshaller
                Unmarshaller unmarshaller = context.createUnmarshaller();
                //Convierte XML a objeto Libros
                return (Libros) unmarshaller.unmarshal(archivo);
            }

        } catch (JAXBException e) {
            System.out.println("Error al cargar libros: " + e.getMessage());
        }
        return new Libros(); //Retorna nueva instancia si no existe
    }

    //Metodo para guardar la lista de libros en el XML
    public void guardarLibros(Libros libros) {  3 usages  Juan Carlos *
        try {
            //configura JAXBContext para la clase Libros
            JAXBContext context = JAXBContext.newInstance(Libros.class);
            //Crea el objeto de la clase Marshaller
            Marshaller marshaller = context.createMarshaller();
            //Formatea el XML para que sea legible (forma arbol)
            marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
            //Convierte el objeto Libros a XML y se guarda en archivo
            marshaller.marshal(libros, archivo);

        } catch (JAXBException e) {
            System.out.println("Error al guardar libros: " + e.getMessage());
        }
    }
}
```

Clase `CsvExportarLibros`

Esta clase se encarga de exportar la lista de libros a un archivo CSV.

```java
public class CsvExportarLibros {  2 usages  Juan Carlos *

    //Metodo para exportar la lista de libros a CSV
    public void exportar(List<Libro> libros, String nombreArchivo) {  1 usage  Juan Carlos *
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(nombreArchivo))) {

            for (Libro libro : libros) {//Bucle foreach que recorre cada libro en la lista

                //Escribe cada libros con los getters de la clase
                writer.write( str: "Titulo: " + libro.getTitulo() + ", Autor: " + libro.getAutor() +
                        ", Año de publicación: " + libro.getPublicacion());

                //Inserta nueva linea
                writer.newLine();
            }

            System.out.println();
            System.out.println("--> Libros exportados a " + nombreArchivo + "!");
            System.out.println();
        } catch (IOException e) {
            System.out.println("Error al exportar a CSV: " + e.getMessage());
        }
```