EXAMEN ACCESO A DATOS. 24 DE FEBREIRO 2025

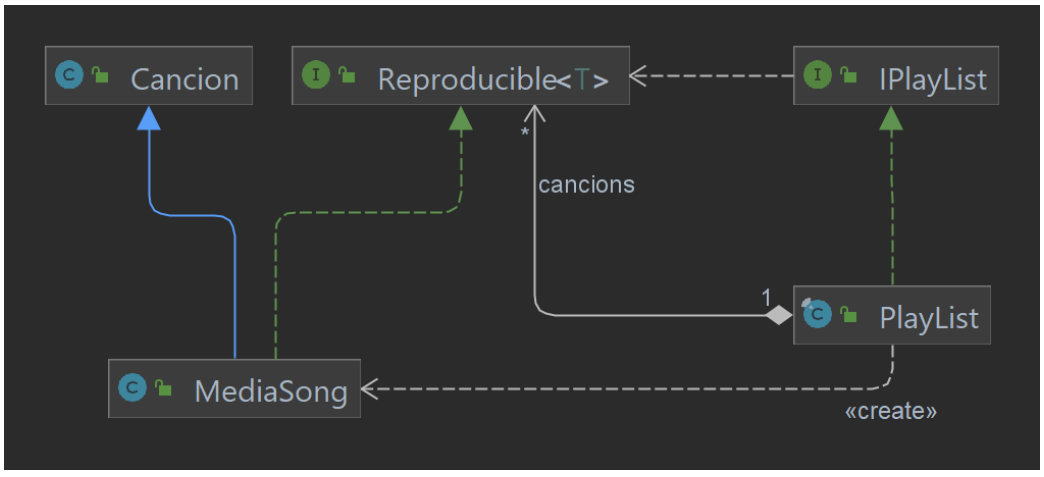
Usuario: **dam2ad**
Contraseña: **YelloSubmarine123$**

Los archivos de examen están en la carpeta: **S:_Materiais_Exames\2DAM-AD**
Debes **copiarlos al escritorio** (para Linux, la ruta es `/home/dam2ad/`) para las rutas que aparecen en el examen y las clases proporcionadas del proyecto.

Modelo de datos

El proyecto ya tiene implementadas las clases del modelo:

- ***Cancion***. Clase que representa una canción, con: ***idCancion*** (Long), ***titulo*** (String), ***autor*** (String), ***duración*** (int), ***dataPublicacion*** (LocalDate).
- ***MediaSong***. Hereda de *Cancion* y, a mayores, tiene el ***audio***, guardado como byte[].
- ***PlayList***. Contiene la lista de Reproducibles, en este caso, de tipo *MediaSong*, así como ***idPlayList*** (Long), ***nome*** (String) y ***dataCreacion*** (LocalDate).
- ***Reproducible***, ***IPlayList***, ***PlayListObserver***: interfaces que deben implantar, aquellas clases que sean reproducibles, una PlayList o un observador de PlayList respectivamente (no las precisáis).



Importante: cualquier uso de la creación de la clase MediaSong debe iniciarse el entorno de JavaFX, por lo que la primera línea del main debe ser (ya implantado):

```
com.sun.javafx.application.PlatformImpl.startup(() -> {});
```

En cualquier caso, no se precisa su reproducción, por lo que puedes implantar todas las clases sin hacer uso de esa instrucción y ***comentando el contenido del método setMediaPlayer().***

Propiedades de la conexión a la base de datos

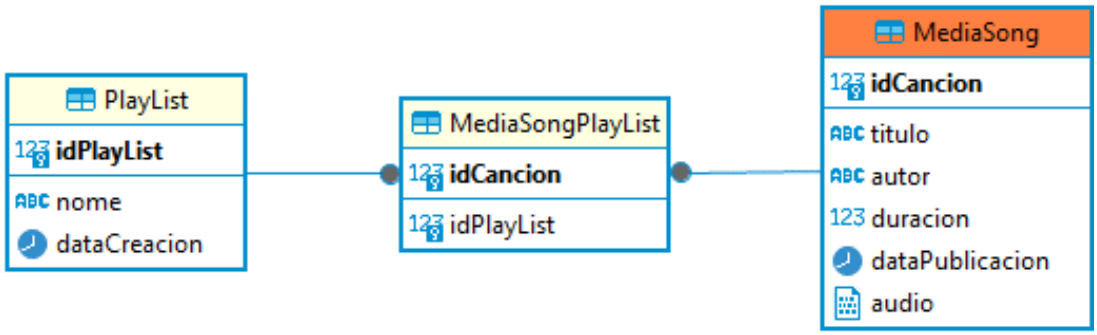
URL
(ojo, no incluyas la extensión al archivo "playlist" de la URL)

```
C:\\Users\\dam2ad\\Desktop\\playlist;
DB_CLOSE_ON_EXIT=TRUE;DATABASE_TO_UPPER=FALSE;FILE_LOCK=NO
```

DRIVER
```
org.h2.Driver
```

Se precisan los **archivos de la base de datos**:
```
playlist.mv.db
playlist.trace.db
```

Esquema de la base de datos



La base de datos ya está creada, aun así, para que conozcáis los parámetros de la base de datos, es Script de creación es el siguiente:

```
-- PUBLIC.Cancion definition

-- DROP TABLE PUBLIC.MediaSong;

CREATE TABLE PUBLIC.MediaSong (
       idCancion INTEGER NOT NULL AUTO_INCREMENT,
       titulo CHARACTER VARYING(255) NOT NULL,
       autor CHARACTER VARYING(255),
       duracion INTEGER,
       dataPublicacion DATE,
       audio BINARY LARGE OBJECT,
       CONSTRAINT idCancion_PK PRIMARY KEY (idCancion)
);
CREATE INDEX MediaSongTitulo_IDX ON PUBLIC.MediaSong (titulo);
CREATE UNIQUE INDEX MediaSong_PK ON PUBLIC.MediaSong (idCancion);
```

```sql
-- PUBLIC.PlayList definition

-- DROP TABLE PUBLIC.PlayList;

CREATE TABLE PUBLIC.PlayList (
	idPlayList INTEGER NOT NULL AUTO_INCREMENT,
	nome CHARACTER VARYING(255) NOT NULL,
	dataCreacion DATE,
	CONSTRAINT idPlayList_PK PRIMARY KEY (idPlayList)
);
CREATE UNIQUE INDEX PlayList_PK ON PUBLIC.PlayList (idPlayList);

-- PUBLIC.MediaSongPlayList definition

-- DROP TABLE PUBLIC.MediaSongPlayList;

CREATE TABLE PUBLIC.MediaSongPlayList (
	idCancion INTEGER NOT NULL,
	idPlayList INTEGER NOT NULL,
	CONSTRAINT idCancionIdPlayList_PK PRIMARY KEY (idPlayList,idCancion),
	CONSTRAINT MSPL_IdCancion_FK FOREIGN KEY (idCancion) REFERENCES
PUBLIC.MediaSong(idCancion) ON DELETE CASCADE ON UPDATE CASCADE,
	CONSTRAINT MSPL_IdPlayList_FK FOREIGN KEY (idPlayList) REFERENCES
PUBLIC.PlayList(idPlayList) ON DELETE CASCADE ON UPDATE CASCADE
);
CREATE INDEX MSPL_IdCancion_FK_IDX ON PUBLIC.MediaSongPlayList (idCancion);
CREATE INDEX MSPL_IdPlayList_FK_IDX ON PUBLIC.MediaSongPlayList (idPlayList);
CREATE UNIQUE INDEX MediaSongPlayListPK ON PUBLIC.MediaSongPlayList
(idCancion,idPlayList);
```

PARTE 1. BASES DE DATOS JDBC

- Todas las **sentencias** deben ser de tipo ***PreparedStatement***.
- Deben **capturarse todas las excepciones y cerrar las sentencias** en **try-catch-with-resources**.
- La conversión de ***java.sql.Date*** a ***java.time.LocalDate*** debe hacerse con el método de instancia ***java.sql.Date#toLocalDate()***, pero controlando antes que no sea el objeto no sea nulo.

1. *PlayerConnectionManager*

Crea una clase que, por medio del **patrón *Singleton Thread-save* de doble comprobación** se use como gestor de creación de la conexión a la base de datos.

Sólo puede crearse una instancia de ***PlayerConnectionManager*** y debe disponer de un método público para obtener la conexión, llamado **getConnection()**.

Debes hacer comprobaciones de conexiones no nulas, ver si la conexión no está cerrada, antes de devolverla o crearla, y capturar excepciones.

2. ***MediaSongDAO*** Implementa *PlayerDAO*<Long, MediaSong>

Implementación mediante patrón DAO de las operaciones con la **tabla *MediaSong***.

Implanta la interface genérica PlayerDAO, que dispone de dos parámetros genéricos: el **tipo de dato de la clave primaria**, de tipo *Long*, y el **tipo de dato de la clase asociada a la tabla**, *MediaSong*.

Atributo

Dispone de un **atributo privado y final, de tipo *java.sql.Connection*, *con***, para referenciar la conexión a la base de datos, y un **constructor que recoge la conexión**.

Métodos

- *List<MediaSong>* ***getAll****()*: devuelve todas las MediaSong de la base de datos. Si no hay canciones devuelve una lista vacía.

- *List<MediaSong>* ***getAllFromID****(Long idPlayList)*: devuelve la lista de canciones de una playList, recogiendo las canciones que están asociadas a la lista de reproducción con idPlayList. Si no hay MediaSong en esa PlayList debe devolver una lista vacía.

  Ayuda. La consulta debe ser un JOIN entre la tabla MediaSong y la que se relaciona con la PlayList, MediaSonPlayList:

```
SELECT C.idCancion, C.titulo, C.autor, C.duracion, C.dataPublicacion, C.audio FROM MediaSongPlayList CP INNER JOIN MediaSong C WHERE C.idCancion = CP.idCancion AND CP.idPlayList = ?
```

- *Long* ***save****(MediaSong cancion)*: guarda la canción en la tabla. Hay **que actualizar el idCancion del objeto *canción* con el idCancion insertado en la base de datos**. Devuelve el *idCancion* de la canción insertada.

  Ayuda: recuerda que para recuperar el id insertado se le debe pasar crear la sentencia con el argumento ***PreparedStatement.RETURN_GENERATED_KEYS***

3. *PlayListDAO* Implementa *PlayerDAO*<Long, PlayList>

Implementación mediante patrón DAO de las operaciones con la **tabla *PlayList***.

Implanta la interface genérica PlayerDAO, con el tipo de dato del identificador de *PlayList*, de tipo Long, y el tipo de dato de la clase, *PlayList*.

Al igual que la clase anterior, tiene un atributo privado, de tipo *java.sql.Connection*, ***con***, para referenciar la conexión a la base de datos, y un constructor que recoge la conexión. Dicho atributo debe ser final.

Métodos

- *List<PlayList>* ***getAll****()*: devuelve todas las listas de reproducción de la base de datos: `SELECT idPlayList, nome, dataCreacion FROM PlayList.`
  Debes recuperar las canciones asociadas a la lista por medio del método *getAllFromID* de *MediaSongDAO*.

- *List<PlayList>* ***getAllFromID****(Long idCancion)*: devuelve la lista de playList que contienen la canción con el idCancion. Si no hay listas con esa canción devuelve null o una lista vacía.
  Ayuda: la consulta debe ser la siguiente:

```
SELECT P.idPlayList, P.nome, P.dataCreacion FROM MediaSongPlayList MP
INNER JOIN PlayList P WHERE P.idPlayList = MP.idPlayList AND
MP.idCancion = ?
```

- *public Long* ***save****(PlayList playList)*: guarda la *PlayList* en la tabla. Hay que actualizar el idPlayList del objeto *playList* con el idPlayList insertado en la base de datos. Devuelve el *idPlayList* de la lista de reproducción insertada.

  Hágase por medio de una **transacción al inicio del método y confirmándola si ha tenido éxito**, descartándola en caso contrario. Ten en cuenta que es importante introducir la lista, las canciones de la lista y las referencias en una única transacción.

  Ayuda: debes (1) insertar la PlayList en la tabla obteniendo la clave insertada. Además, también (2) insertar las canciones de la lista por medio del método "save" de MediaSongDAO para cada una de las canciones, recuperar el idMediaSong e (3) insertar ambos ids en la tabla MediaSongPlayList.

  Para convertir la fecha de creación de la PlayList, de tipo LocalDate en java.sql.Date puedes hacerlo con:

  java.sql.Date.valueOf(playList.getDataCreacion())

Consultas (la inserción de la MediaSong se hace invocando al save de MediaSongDAO):

```
INSERT INTO PlayList (nome, dataCreacion) VALUES (?, ?)
INSERT INTO MediaSongPlayList (idCancion, idPlayList) VALUES (?, ?)
```

4. *AppPlayListDB* Impl

Case de la aplicación, ya creada, que:

- **Cree una conexión a la base de datos** haciendo uso de PlayerConnectionManager
- **Recupere todas las canciones de la base de datos**, haciendo uso del método getAll() de MediaSongDAO, y las muestra.
- **Recupera todas las PlayList** de la base de datos, haciendo uso del método getAll() de PlayListDAO, y la/s muestra.
- **Crea dos canciones**, por medio de alguno de los métodos que recogen la ruta al archivo.
- **Guarda las canciones en la base de datos** por medio del método “save" de MediaSongDAO, mostrando las canciones insertadas para comprobar que tienen idCancion.
- **Crea una PlayList con, al menos, dos canciones y la guarda en la base de datos, mostrando la lista para** comprobar que funciona correctamente.

PARTE FLUJOS, FICHEROS E/S Y GSON

Ejercicio 1. Ficheros y flujos

Deben realizar dos métodos, uno para **guardar el contenido del archivo** de audio de la ***MediaSong*** y otro **para cargar** la ***PlayList*** desde un **archivo de texto**.
Archivos necesarios para la ejecución:

ARCHIVO CON LOS DATOS DE LA PLAYLIST
`C:\Users\dam2ad\Desktop\playlist.txt`

DIRECTORIO CON EJEMPLOS DE ARCHIVOS DE AUDIO
`C:\Users\dam2ad\Desktop\mp3`

Debes crear los métodos en la Clase *AppPlayListFiles*, ya implantada.

A) saveMediaFile

*Crea un método para **guardar el archivo de la canción** **(cancion.getArquivo())** **de un objeto MediaSong en un archivo destino**. El método recoge la canción en formato MediaSong y el nombre con la ruta a un archivo destino, haciendo una copia del archivo multimedia (si existe) en dicha localización.*

La firma del método es la siguiente:

***public boolean saveMediaFile(MediaSong** mediaSong, **String** destino)*

La obtención del archivo de la MediaSong puede hacerse por medio del método:
public *Path **getArquivo**()*

El código del método principal de la aplicación está en la clase AppPlayListFiles, pero para pruebas, puedes crear un objeto de tipo MediaSong a partir de los constructores. Por ejemplo:

public ***MediaSong**(String arquivo)*
public ***MediaSong**(String archivo, String titulo)*
public ***MediaSong**(String archivo, String titulo, String autor)*

Importante: ***la lectura y escritura debe ser con Buffer****, además de leer en bloques* ***(no se admite la respuesta correcta cuando la lectura o escritura por medio de métodos estáticos de Files)****. Léanse en bloques de 1024 bytes.*

Devuelve ***true*** *si no se ha producido algún error y la copia ha sido* ***correcta****.*

B) loadPlayList (Dejadlo para el final)*

public PlayList ***loadPlayList****(String nomeArquivo)*

- Recoge el **nombre del archivo de texto, sólo el nombre**, que debe estar en el directorio de la aplicación, y **lee línea a línea** los datos del archivo. *Ayuda*: *la* ***ruta al directorio de ejecución*** *se puede conocer por medio de la propiedad del sistema "user.dir":* `System.getProperty("user.dir")`

Path *p =* ***Paths****.get(System.getProperty("user.dir"), nomeArquivo);*

- Los comentarios del archivo de texto (no se tienen en cuenta a la hora de leer el archivo desde el método) **empiezan por '#'** (se recomienda usar la constante definida en la clase como COMENTARIO).
- La **primera línea** que no es un comentario es el **título del disco**. Siempre la primera línea significativa.
- El resto de las líneas contiene los datos de la MediaSong con los **campos separados por |** (constante definida en la clase como SEPARADOR) con el siguiente formato:

`nombre archivo multimedia | título de la canción | autor`

- El campo con el nombre del **autor es opcional**.
- Antes de proceder a la lectura del archivo, **debe comprobarse que el archivo existe**.

Ejemplo de archivo para cargar el disco (existe un ejemplo compartido, *playlist.txt*):

```
Grandes clásicos Bach y Gershwin
# Lista de obras musicales.
# Bach
E:\38 - Audios\01.mp3| Concierto para violín, BWV 1043. Vivace|Bach
E:\38 - Audios\Variaciones Goldberg.mp3| Variaciones Goldberg |Bach
# Gershwin
E:\38 - Audios\Americano en París.mp3| Un americano en París|George Gershwin
E:\38 - Audios\mp3\06. Rhapsody in blue.mp3| Rhapsody in blue|George Gershwin
```

Ejercicio 2) JSON

Se trata de hacer una aplicación, ya implementada, que **escriba y lea la PlayList de un archivo JSON**.

La clase de la aplicación ***AppPlayListJSON***, ya está implantada, falta crear los adaptadores de tipo y completar el código del método main.

Para ello precisamos un **adaptador para la clase MediaSong y otra para la PlayList**.

A) MediaSongTypeAdapter

Clase de tipo ***TypeAdapter<MediaSong>*** para adaptar la clase ***MediaSong***, que contiene *idCancion*, *título*, autor, *duración*, *dataPublicacion* y *audio* como array de tipo byte.

Debe tener un **atributo de tipo Path, directorio, con el directorio con el que se guardan/leen los archivos** tras la serialización/deserialización.

Además, el **constructor recoge la ruta al directorio**.

Al a hora de escribir (y leer) debe hacerlo con el siguiente formato, serializando nulos:

```
{
  "idCancion": null,
  "título": "Piano Trio in E-Flat, Op. 100",
  "autor": "Various Artists",
  "duración": 258,
  "dataPublicacion": null,
  "audioPath": "C:\\Users\\dam2ad\\Desktop\\json\\tmpmedia773993131.mp3"
}
```

Al serializar, además, **debe guardar el archivo, creándolo en el directorio**.

Al deserializar debe cargar el archivo en el atributo correspondiente, por medio del método ***setAudio*** o ***setBytes*** (son el mismo método).

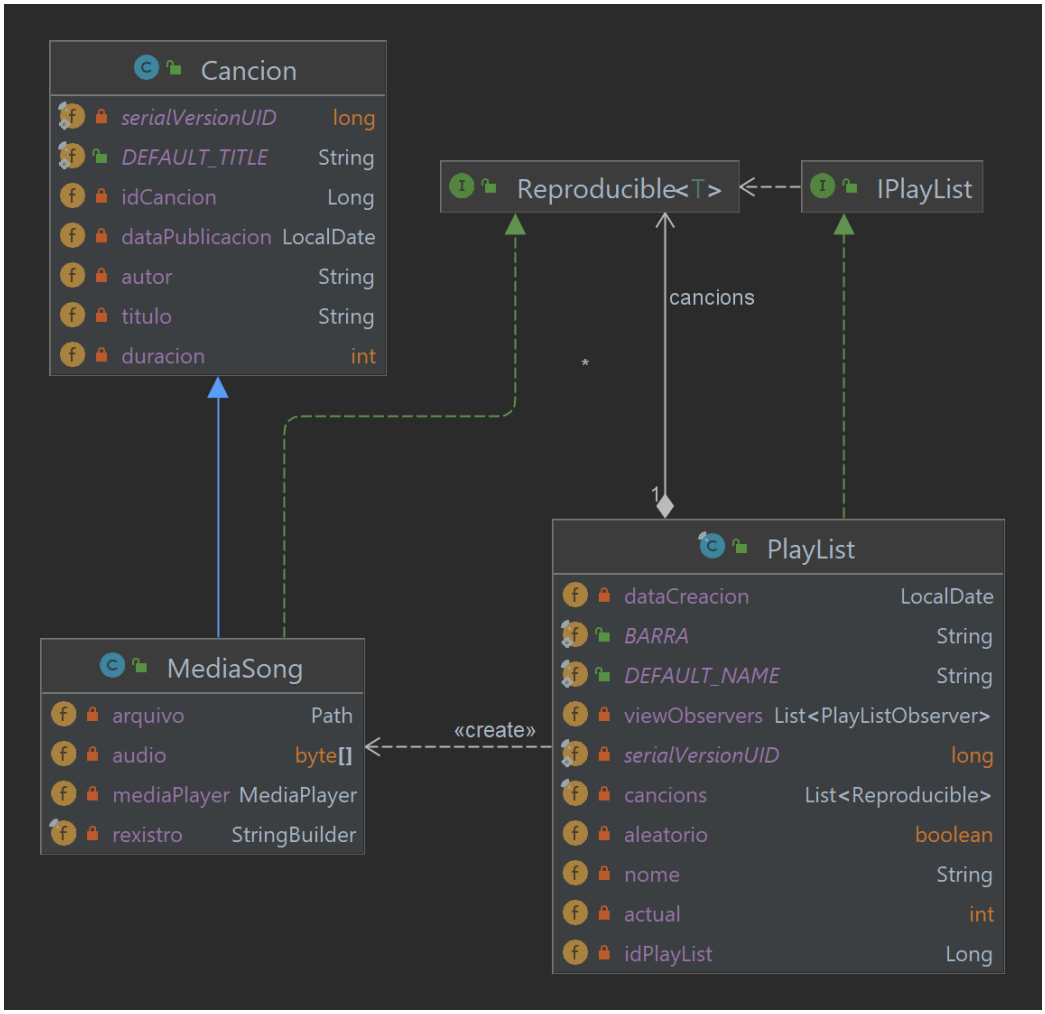
B) PlayListSerializer

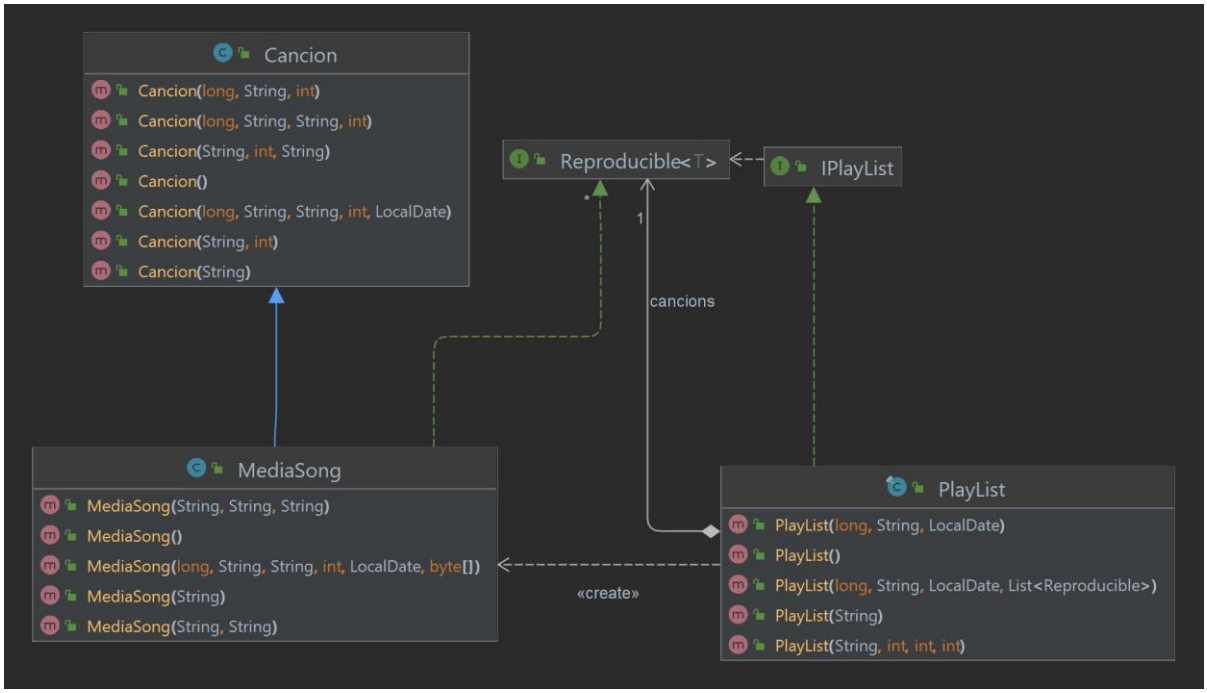
Adaptador de la *PlayList* de tipo ***JsonSerializer<PlayList>***, que serializa la PlayList en un Objeto JSON con el ***idPlayList***, ***nome***, ***dataCreacion*** y array de canciones.

El formato de salida tiene la siguiente estructura:


```
{
  "idPlayList": null,
  "nome": "Grandes de la música Clásica",
  "dataCreacion": "2023-12-11",
  "cancions": [
    {
      "idCancion": null,
      "título": "Symphony no. 9 in D minor, op. 125 IV. Presto",
      "autor": "Various Artists",
      "duración": 98,
      "dataPublicacion": null,
      "audioPath": "C:\\Users\\dam2ad\\Desktop\\json\\tmpmedia131855.mp3"
    },
    {
      "idCancion": null,
      "título": "Concerto for Two Harpsichords and Orchestra in C-Minor",
      "autor": "Various Artists",
      "duración": 316,
      "dataPublicacion": null,
      "audioPath": "C:\\Users\\dam2ad\\Desktop\\json\\tmpmedia150046.mp3"
    },
    {
      "idCancion": null,
      "título": "Prokofiev: Romeo and Juliet, Op. 64/Act 1 - Dance Of The
Knights\u0000",
      "autor": "Sergei Prokofiev",
      "duración": 307,
      "dataPublicacion": null,
      "audioPath": "C:\\Users\\dam2ad\\Desktop\\json\\tmpmedia140855.mp3"
    },
    {
      "idCancion": null,
      "título": "Piano Trio in E-Flat, Op. 100",
      "autor": "Various Artists",
      "duración": 258,
      "dataPublicacion": null,
      "audioPath": "C:\\Users\\dam2ad\\Desktop\\json\\tmpmedia1704034.mp3"
    }
  ]
}
```

DETALLES DE LAS CLASES DEL MODELO





DEPENDENCIAS DEL PROYECTO MAVEN

```
    <dependencies>
     <!-- Recuperación -->
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.10.1</version>
        </dependency>
     <!-- Fin recuperación -->
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-media</artifactId>
            <version>20.0.2</version>
        </dependency>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-base</artifactId>
            <version>20.0.2</version>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <version>2.1.214</version>
        </dependency>
    </dependencies>
```