Proyecto Spotify a Sqlite

Desarrollo de Aplicaciones para Ciencia de Datos (DACD)

Segundo curso

Grado en Ciencia e Ingeniería de Datos

Escuela de Ingeniería Informática (EII)

Universidad de Las Palmas de Gran Canaria (ULPGC)

Índice

1. Resumen:

Para el proyecto he divido las clases en paquetes en función de su relación. Cada objeto que puede ser insertado en las tablas de la base de datos posee un paquete. Todos tienen una estructura muy similar. El paquete artista, álbum y track poseen:

- Una clase con su mismo nombre que sirve para definir los constructores para crear objetos que los puedan representar, así como getters que devuelven los valores de los atributos.
- Una clase ***Accessor** que realiza la petición a la api de Spotify. En el caso de Artists y Albums usa como parámetro en esta petición la lista con los ids de los artistas y en los Tracks usa una lista de objetos de la clase AlbumList que explicaré a continuación.
- Una clase ***List** que contiene una lista de objetos de la clase de su propio paquete y tiene la función de facilitar la deserialización en bloque (POJO), en este marco se contiene la clase AlbumList mencionada anteriormente.
- Una clase **Create*** que traduce el json obtenido con el accesor get al objeto de la clase list que nos servirá de apoyo para luego insertarlos en una base de datos.

En el proyecto se encuentra un paquete llamado **DataBase** que contiene todas las clases para el manejo de la base de datos:

- Una clase **CreateConnection** que recibe como parámetro un nombre (String) para la base de datos y contiene el método connect que devuelve una conexión (Connection) con la base de datos recién creada.
- La clase **DataBaseInsert** recibe como parámetro una conexión a base de datos y contiene métodos de tipo void para insertar los artistas, álbumes y canciones en tablas de la base de datos. Invoca a la siguiente clase.
- La clase **DmlTranslator** recibe como parámetro una conexión y estable las instrucciones en Sql a ejecutar por los métodos en DataBaseInsert para poder introducir los valores de los atributos de los objetos a nuestra base de datos teniendo métodos para cada objeto (Artist, Album, Track).
- Ni DataBaseInsert, ni DmlTranslator tendrían sentido sin **DdlTranslator**, clase que recibe en el constructor una conexión y se encarga de las instrucciones de Sql para la creación de tablas, con un método para cada tabla (Artist, Album, Track).

Por último, tenemos el paquete interfaz que contiene la clase MainScreen con los mensajes que se le muestran al usuario y que contiene CreateArtist().La interfaz permite al usuario elegir si descargar todos los artistas disponibles o seleccionar entre ellos(mediante FIlterArtist y ArtistSelect), así como informa de las operaciones a medida que se van ejecutando.

2. Diagrama de clase:

Enlaces a imágenes:

- Album: https://github.com/Alejeglez/Spotify/blob/master/Diagramas/Album.png?raw=true
- Artist: https://github.com/Alejeglez/Spotify/blob/master/Diagramas/Artist.png?raw=true
- DataBase: https://github.com/Alejeglez/Spotify/blob/master/Diagramas/DataBase.png?raw=true
- Interfaz: https://github.com/Alejeglez/Spotify/blob/master/Diagramas/Interfaz.png?raw=true
- Track: https://github.com/Alejeglez/Spotify/blob/master/Diagramas/Track.png?raw=true

3. Recursos usados:

- **IDE:** IntelliJ IDEA Community Edition 2022.2.2.
- **Control de versiones:** GIT (integrado en IntelliJ)**.**
- **Herramienta diagrama de clases:** StarUml(Sin registrar, por lo que tiene marca de agua).
- **Herramienta de documentación:** Word.

4. Diseño:

Para este apartado, he usado los paquetes como "módulos" para que cada paquete contuviera las clases relacionadas entre sí y no hubiera mucha dependencia entre ellas. He intentado que cada clase tenga una sola responsabilidad, así como que en el proyecto haya una fuerte modularidad, cohesión y reducir la interdependencia entre paquetes.

Nuestro proyecto podría ser interpretado siguiendo el diseño de la arquitectura hexagonal, donde tenemos completamente aislados de nuestros datos y de los accesos a Spotify, la interfaz de usuario y las bases de datos. Bajo ningún concepto la clase Interfaz o DataBase llaman a Spotify o crean objetos de las clases Artist, Album y

Track, si no que invocan a los paquetes de dichas clases que actúan como conexión con Spotify o crean los objetos.

He decidido usar clases accessor exclusivamente para cada paquete porque la clase Track recibe como parámetro en el constructor una lista de objetos AlbumList y evita tener que volver a acceder a la Api de Spotify. Esto supone una diferencia con los paquetes Artist y Album, que reciben como parámetro la lista que contiene los ids de los artistas disponibles. Esta decisión fue el motivo por el que creé clases similares en distintos paquetes y así separar responsabilidades.

Por último, decidí añadir por mi cuenta en la interfaz una opción para elegir el artista. De esta manera, se puede simplificar la carga de trabajo con la base de datos y reducir el tiempo de ejecución del programa, si no se desean guardar todos los artistas.

5. Conclusiones:

Estoy satisfecho con que el proyecto funcione, sin embargo, creo que tal vez podría haber diseñado los paquetes o clases en torno, a su funcionalidad en lugar de en torno a los objetos de Spotify. También me hubiera gustado implementar algún mecanismo para controlar las excepciones sobre la opción que añadí, sin embargo, como no encontré forma, decidí indicar con un mensaje al usuario que si el nombre no estaba en la lista el programa se detendrá. No estoy del todo contento con el tamaño de la clase MainScreen y creo que añadir un controlador de excepción a esta hubiera supuesto un aumento en las líneas de código.

Otra de las cosas en las que me he fijado, es en lo importante que es optimizar el tiempo de acceso o el número de accesos a la base de datos, ya que el tiempo varía considerablemente entre las pruebas que he hecho comparando un disco hdd con ssd.

Por último, he aprendido la importancia que tiene la documentación cuando se trabaja con una fuente externa ya que la referencia de Spotify web sirvió como guía al desarrollar cada una de las clases y los accesos.

6. Líneas Futuras:

Para desarrollar un producto comercializable sería conveniente aumentar el rango de alcance del programa, ya que cinco artistas parece limitado. Otro de los temas a tratar sería el tiempo de ejecución que puede ser un factor que aleje a ciertos usuarios. La estructura del programa podría ser dividida en módulos que puedan ser referenciados por futuros proyectos contribuyendo a la solución de bugs y actualizaciones. La interfaz es bastante mejorable y necesitaría de un controlador de excepciones, además, debería ser gráfica para ser accesible a un mayor número de usuarios. Por último, se podría barajar la opción de obtener más datos de Spotify como audiolibros, podcasts y playlists.

7. Bibliografía:

- Como pasar de json a lista de objetos: https://howtodoinjava.com/gson/gson-parse-json-array/
- Tutorial Sqlite: https://www.sqlitetutorial.net
- Referencia Spotify api: https://developer.spotify.com/documentation/web-api/

8. Notas:

- El proyecto en el repositorio de Github no contiene los valores de ClientId ni ClientSecret por lo que será preciso que se introduzca en la clase SpotifyAuthorization.
- Las imágenes no se veían muy bien en el documento, por lo que las he subido al repositorio de Github en la carpeta diagramas.

Alejandro Jesús González Santana