**Informe Desafío 2 (Informática 2)**

**Estudiantes:**

Jose Eduardo Valverde Alvarez ([josee.valverdea@udea.edu.co](mailto:josee.valverdea@udea.edu.co))

Andrés Felipe Lafaurie Rincon ([andres.lafaurie@udea.edu.co](mailto:andres.lafaurie@udea.edu.co))

**Profesor:**

Anibal Jose Guerra Soler

Universidad de Antioquia

Facultad de Ingeniería

Ingeniería en Telecomunicaciones e Ingeniería electrónica

Medellín

17 de Octubre de 2025

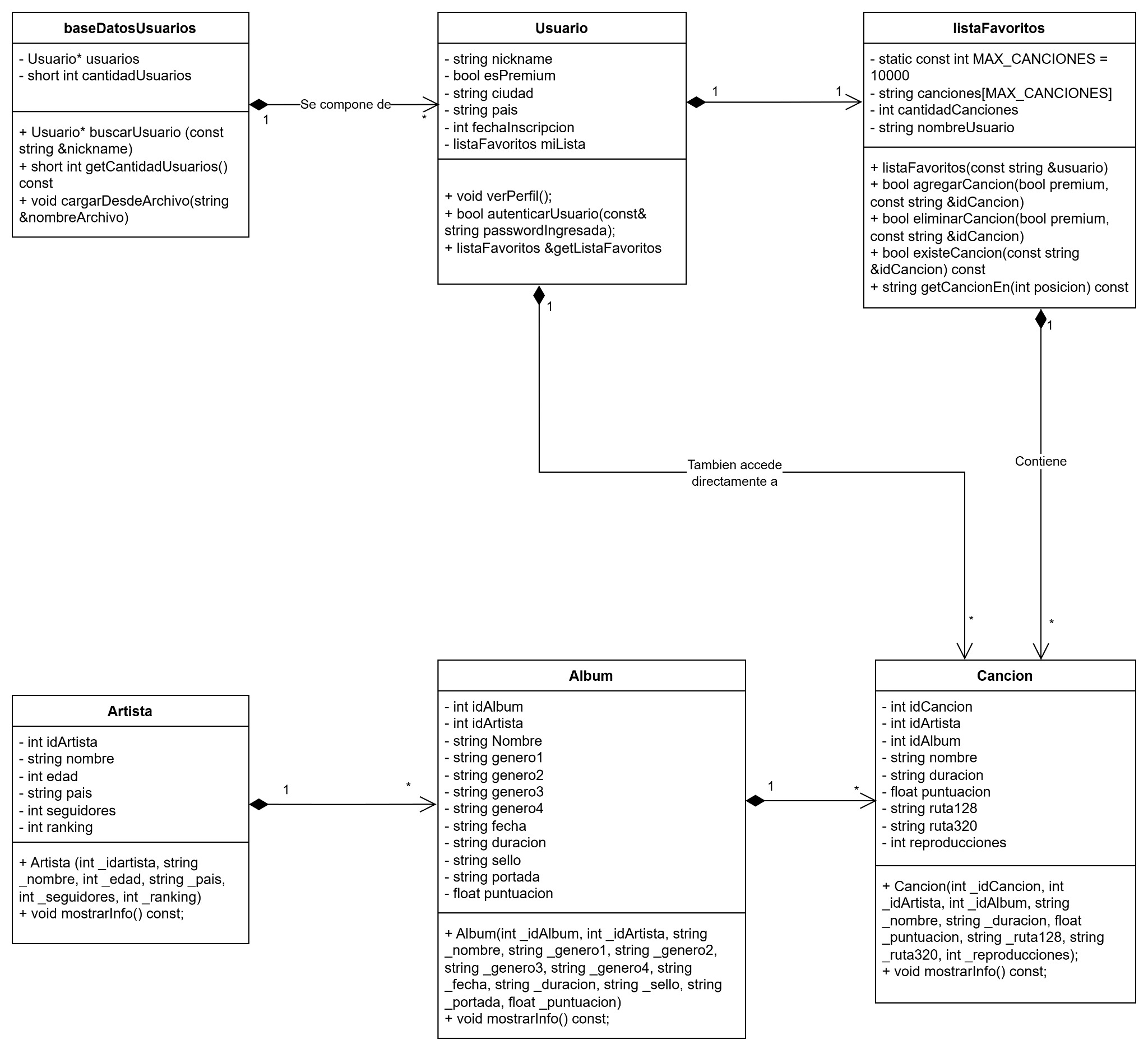
1. **Análisis y diseño de la solución**

1.1. Contextualización

UdeATunes es un servicio de streaming musical cuyo despliegue de operación será en forma de una plataforma en línea que ofrezca su servicio a nivel global. Cada usuario en la plataforma está registrado con un nickname o nombre de usuario, tipo de membresía, país y la fecha de inscripción a la plataforma, los usuarios de la plataforma que están pagando premium pueden tener acceso a funcionalidades adicionales que tiene la plataforma, una de ellas es poseer una lista de hasta 10000 canciones, estos usuarios también pueden editar sus listas e incluso pueden seguir las listas de otros usuarios y reproducir cualquier canción sin interrupciones publicitarias.

En el caso de los usuarios que no pagan premium, al reproducir dos canciones, aparecerá en pantalla un mensaje publicitario de máximo 500 caracteres, la publicidad en la plataforma no estará conformado por un solo mensaje que se mostrará consecutivamente sino de hasta 50 mensajes de diferentes categorías A, B, y C, la primera siendo de mayor prioridad y la última de menor prioridad, estos usuarios no tienen acceso a las listas de reproducción ya que es una función exclusiva para los usuarios que están pagando la versión premium de la aplicación.

1.2 Solución Planteada



Bajo este diagrama UML que también está adjunto en el repositorio, el codigo se trabaja en 6 clases las siguientes son:

* **baseDatosUsuarios**

Tal y como está descrito en el nombre la clase se encargará de cargar los datos que están guardados en una base de datos .txt

**Atributos:**

* Usuario\* usuarios
* short int cantidadUsuarios

**Métodos:**

* short int getCantidadUsuario()
* void cargarDesdeArchivo(string& nombreArchivo)
* **Usuario**

Pertenece a: baseDatosUsuarios (1 -> n)

Esta clase se encargará de reconocer e inicializar los datos del usuario activo que se reciben de baseDatosUsuarios

**Atributos:**

* string nickname
* bool esPremium
* string ciudad
* string pais
* int fechaInscripcion
* listaFavoritos miLista (clase listaFavoritos)

**Métodos:**

* void verPerfil()
* bool autenticarUsuario(const& string passwordIngresada)
* listaFavoritos &getListaFavoritos (Consulta a clase listaFavoritos para obtener la lista del usuario)
* **listaFavoritos**

Pertenece a: Usuario (1 -> 1)

La clase permite administrar y obtener datos sobre las listas de reproducción de los usuarios en la plataforma, mientras el usuario esté registrado en la plataforma como usuario premium.

**Atributos:**

* static const int MAX\_CANCIONES = 10000
* string canciones[MAX\_CANCIONES]
* int cantidadCanciones
* string nombreUsuario

**Métodos:**

* listaFavoritos(const string& usuario) (\*Sobrecarga con parametros)
* bool agregarCancion(bool premium, const string &idCancion)
* bool eliminarCancion(bool premium, const string &idCancion)
* bool existeCancion(const string &idCancion) const
* string getCancionEn(int posicion) const

Los métodos como agregarCancion y eliminarCancion tal y como su nombre indica, permiten agregar y eliminar canciones de la lista, los métodos también incluyen la actualización en tiempo real de las listas (es decir, adicion y eliminacion de las canciones que el usuario propietario indique) en caso de que sean seguidas por otros usuarios.

* **Canción**

Pertenece a: Usuario (1 -> n), listaFavoritos (1 -> n), Album (1 -> n)

Un usuario puede acceder directamente a esta clase a través de una reproducción aleatoria a cierto número de todas las canciones disponibles en la base de datos, o también, indirectamente a través de listaFavoritos haciendo reproducción aleatoria a cierta cantidad de canciones en su lista en tal caso de que tenga una.

Esto es a través de una librería de funciones que fueron hechas para la reproducción de canciones que será explicada en el punto de los algoritmos que se le aplicaron a esta función (reproducción.cpp).

Los datos de esta clase se inicializan a través de otra librería de funciones (GestorArchivos.cpp) que se encarga de leer el archivo en donde están alojados los datos de las canciones, y de paso los diccionarios que permiten conocer más detalles sobre otros datos clave sobre la canción (el album al que pertenece y su artista)

**Atributos**

* int idCancion
* int idArtista
* int idAlbum
* string nombre
* string duracion
* float puntuacion
* string ruta128
* string ruta320
* int reproducciones

**Metodos**

* Cancion(int \_idCancion, int \_idArtista, int \_idAlbum, string \_nombre, string \_duracion, float \_puntuacion, string \_ruta128, string \_ruta320, int \_reproducciones) (\*Sobrecarga de Parametros)
* **Album**

Pertenece a: Artista (1 -> \*)

**Atributos:**

* int idAlbum
* int idArtista
* string Nombre
* string genero1
* string genero2
* string genero3
* string genero4
* string fecha
* string duracion
* string sello
* string portada
* float puntuacion

**Métodos**

* Album(int \_idAlbum, int \_idArtista, string \_nombre, string \_genero1, string \_genero2, string \_genero3, string \_genero4, string \_fecha, string \_duracion, string \_sello, string \_portada, float \_puntuacion) (Sobrecarga de Parametros)
* **Artista**

**Atributos:**

* int idArtista
* string nombre
* int edad
* string pais
* int seguidores
* int ranking

**Métodos:**

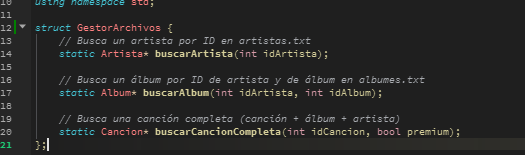
* Artista (int \_idartista, string \_nombre, int \_edad, string \_pais, int \_seguidores, int \_ranking) (Sobrecarga de parametros)

El consumo de recursos y la cantidad de iteraciones que se hacen en los métodos de estas clases (también las funciones secundarias que complementan estas) pueden variar según la cantidad de datos que se manejan en la base de datos, que por lo tanto se imprimirán en consola, excepto por las listas de reproducción que su tamaño fijo es de **32000 bytes**

1. **Otros algoritmos implementados**

En este punto vamos a enfocarnos en la parte que consideramos menos trivial del código que son las librerias GestorArchivos y reproduccion.cpp (que se hicieron como funciones) en la implementación de este codigo

* **GestorArchivos**

****

La lógica de este conjunto de funciones es la siguiente: La base de datos de las canciones está dividida en tres partes: las canciones y un diccionario en donde están los datos de los álbumes y otra en donde están los datos de los artistas.

Como los identificadores de las canciones están conformados por 9 dígitos en cada canción, están divididos en tres partes: los 4 primeros dígitos corresponden al identificador del artista, los siguientes 2 dígitos corresponden a los del álbum y los 2 últimos dígitos corresponden por supuesto a los del artista, entonces, cada vez que una clase necesite los datos de una canción, tome un identificador y luego pase a la función de **buscarCancionCompleta** que toma los datos de la canción que está en el archivo, y por consiguiente, toma los datos del álbum y del artista a través de sus respectivas funciones siguiendo el orden de la jerarquía de los datos principales de la canción y de paso inicializando los datos secundarios de cada dato principal en sus clases.

Las tres funciones fueron declaradas en un struct pensando en el manejo de este tipo de datos

* **reproduccion.cpp**

****

Este módulo fue pensado para almacenar todas las funciones que después usamos en el momento de la reproducción de la música en el programa, almacena toda clase de funciones que se usan para imprimir algo en pantalla o que funcionan como opciones durante la reproducción de canciones según sus modos.

Funciones para la reproducción de canciones.

Las funciones **Totalcanciones, reproduccionAleatoria** y **reproduccionLista** son funciones que se utilizan directamente para elegir canciones segun los criterios necesitados, estas funciones reciben las rutas a los archivos donde se almacenan las canciones disponibles y eligen cuales van a imprimir.

Totalcanciones, esta primera función se encarga de leer todas las líneas del archivo que se le haya indicado, con esto permitira saber cual es el numero maximo de canciones que se pueden leer del archivo y es necesario para las otras funcionalidades. ReprouccionAleatoria por su lado recibe varios parametros con los que se realizan dos acciones, cargar la publicidad e imprimirla en caso de los usuarios no premium y recibir una ruta de archivo junto con el número total de canciones de este, lo que le permitirá escoger un número aleatorio entre 0 y la cantidad de canciones, después imprimirá la información de la canción escogida. ReproduccionLista, esta función por su parte es muy parecida a la anterior, sin embargo esta se encarga de reproducir el archivo que reciba en orden, una por una y cuando llega al final vuelve a empezar.

Las funciones **detenerReprouccion, continuarReprouccion, retrocederCancion y repetirCancion** son funciones relacionadas a las opciones que tiene el usuario para escoger tras reproducir canciones, detenerReproduccion y continuarReproduccion imprimen por pantalla el estado de la reproduccion. Las funciones repetirCancion y retrocederCancion, junto con las funciones **inicializarHistorial, agregarCancion** y **liberarHistorial** se utilizan para crear o para utilizar el historial. InicializarHistorial, inicializa las variables del historial y crea el arreglo dinámico donde se guardaran. agregarCancion, agrega un id de una canción al historial de canciones y aumenta sus variables a necesidad de la lógica. liberarHistorial, borra el arreglo dinámico y reinicia las variables del historial. repetirCancion, accede al historial y reproduce la última cancion que aparece. retrocederCancion, recibe cuantas canciones desea retroceder el usuario luego accede a historial y vuelve sobre sus pasos ese número de canciones.

Las funciones **CargarAnuncios, SeleccionarAnuncioAleatorio, mostrarPublicidad** estan relacionadas al sistema de anuncios para los usuarios no premium, estas funciones se encargan de cargar, buscar o imprimir la publicidad, todas ellas comprueban que el usuario sea de categoria premium para funcionar o no.

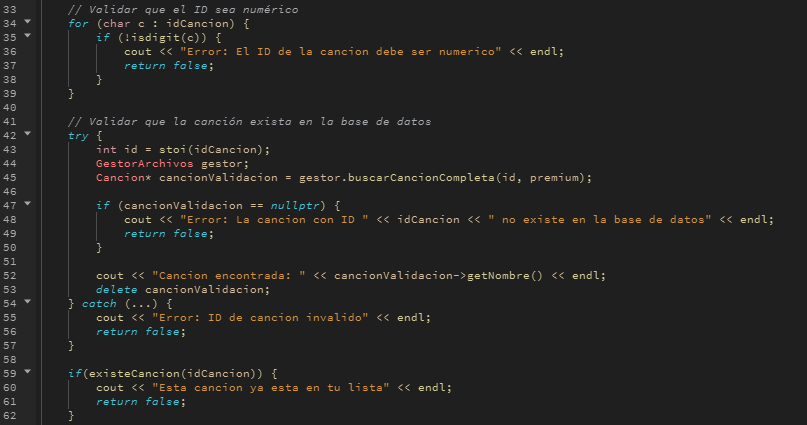
CargarAnuncios será la encargada de inicializar los anuncios que están alojados en un archivo .txt en donde cada linea esta estructurada por: la prioridad y el mensaje que corresponde, la segunda funcion mencionada se encargará de medir la cantidad de veces en las que un anunció se imprimirá al reproducir las canciones y por ultimo la funcion mostrarPublicidad será el despliegue del anuncio a la consola

1. **Problemas durante el desarrollo**

Tuvimos al menos dos problemas durante el desarrollo del código, uno en un atributo de la clase listaFavoritos agregarCancion, y otra en donde el programa se colapsaba durante la reproducción aleatoria en donde al parecer el código no sabía cómo responder en caso de que no se encontraban más canciones en la lista, en ambos problemas se implementaron manejo de excepciones los cuales se explicaran en los siguientes detalles de la solución a estos problemas.

1.1 Cualquier arreglo de strings se permitia agregar como “canción” en la playlist

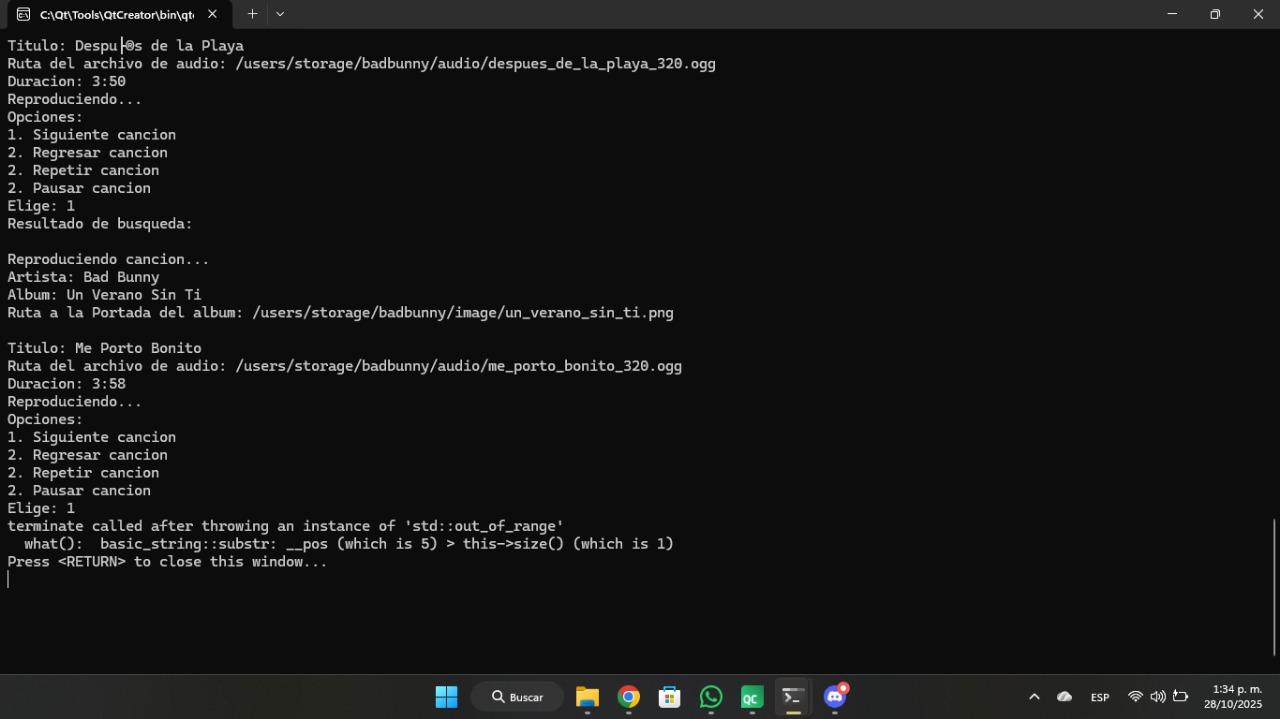
Durante la ejecución del método agregarCancion había un problema en el codigo que permitía que cualquier string que se reciba en la consola al pedir el identificador de la canción a agregar se permitía agregar (incl. guardar en el archivo .txt) cualquier cadena de caracteres como una canción sin previa identificación que verifique si la canción pertenece a la base de datos.



La solución al problema fue primero: agregar un condicional que verifique si el identificador recibido en consola es una cadena numérica, y luego implementamos una excepción pensando en un posible caso de que la cadena recibida si sea un caracter numérico de 9 dígitos pero no está alojado en la base de datos en donde están guardadas todas las canciones de la plataforma.

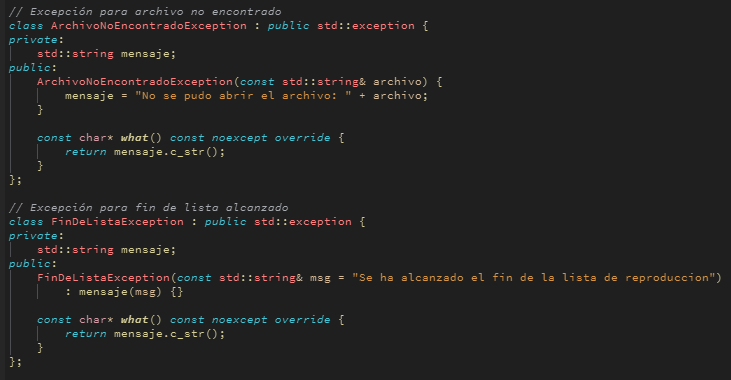
1.2 Colapso del programa al no encontrar más canciones en la lista de canciones favoritas o la biblioteca de canciones de la plataforma.

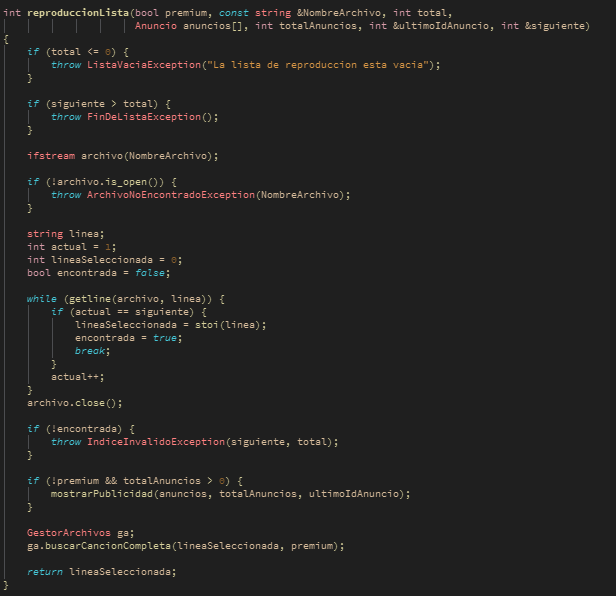
Otro problema que tuvimos en las últimas horas del desarrollo fue al probar el reproductor, que al reproducir aleatoriamente en las listas listas, el programa se crasheaba debido a que no encontraba más canciones en la lista de reproducción que se estaba ejecutando en ese momento, retornando en consola este mensaje



La solución al problema fue declarar excepciones a través de una librería que se creó llamada excepciones.h y finalmente implementarlas a las funciones de reproducción aleatoria sobre todo a las de las listas que es en donde se presentaba el problema que hacía que el programa se cerrará abruptamente.







1. **Evolución de la solución y consideraciones a tener en cuenta**

* El análisis del problema inicial que nos llevó a idear un modelo basado completamente en archivos y carpetas, donde un artista sería una carpeta, con carpetas adentro para el álbum y dentro de estos archivos con los nombres de la canción y dentro de estos archivos la información. Después de la contextualización en clase, pasamos a una nueva corriente, el uso de clases, se pensó entonces en un sistema completamente generado por clases, donde para cada artista, álbum y canción, se crearía un objeto de la clase correspondiente, cuyos atributos serian la información de cada objeto según lo pedía el documento, tras una de las clases donde hubo espacio para preguntas, le planteamos este modelo al profesor que nos advirtió sobre el uso de memoria y los inconvenientes respecto a este modelo, como el tener que generar todos los objetos cada vez que se ejecutara el programa o la cantidad de variables que se usaron. Gracias a esta observación se llegó a la conclusión de que habría que combinar ambos modelos, el uso de archivos y el uso de clases, con este modelo planteado se diseñó el diagrama de clases, donde se definieron las clases basedatosusuarios, usuario, listaReproduccion, reproductor, publicidad, artista, álbum y canción, pero que en el transcurso del desarrollo aparecieron nuevas y desaparecieron algunas. Para el modelo final existen las clases Album, Artista, Cancion, usuario, Basedatosusuarios, listafavoritos y exepciones, mientras que clases pasaron a ser funciones como la clase reproductor que pasó a ser un modulo de funciones junto con gestorarchivos. En este nuevo modelo existirian archivos que contendrian la información de usuarios, artistas, álbumes, canciones y listas de reproducción disponibles en la plataforma, archivos que usando las funciones de gestorarchivos se extraería la información y se crearán objetos de las clases pertinentes para la información que se necesitara, así se optimizará memoria ya que solo crearía los objetos que fueran necesarios y ahorra el inicializar cada conjunto de información manualmente.
* Como conclusión para este trabajo y siguientes, se recomienda evaluar varios modelos antes de empezar a desarrollar tu programa y consultar con tiempo a tu tutor para que te pueda brindar indicaciones antes de empezar a trabajar en un sistema que tal vez quedará obsoleto.