Universidad de Antioquia - informatica 2

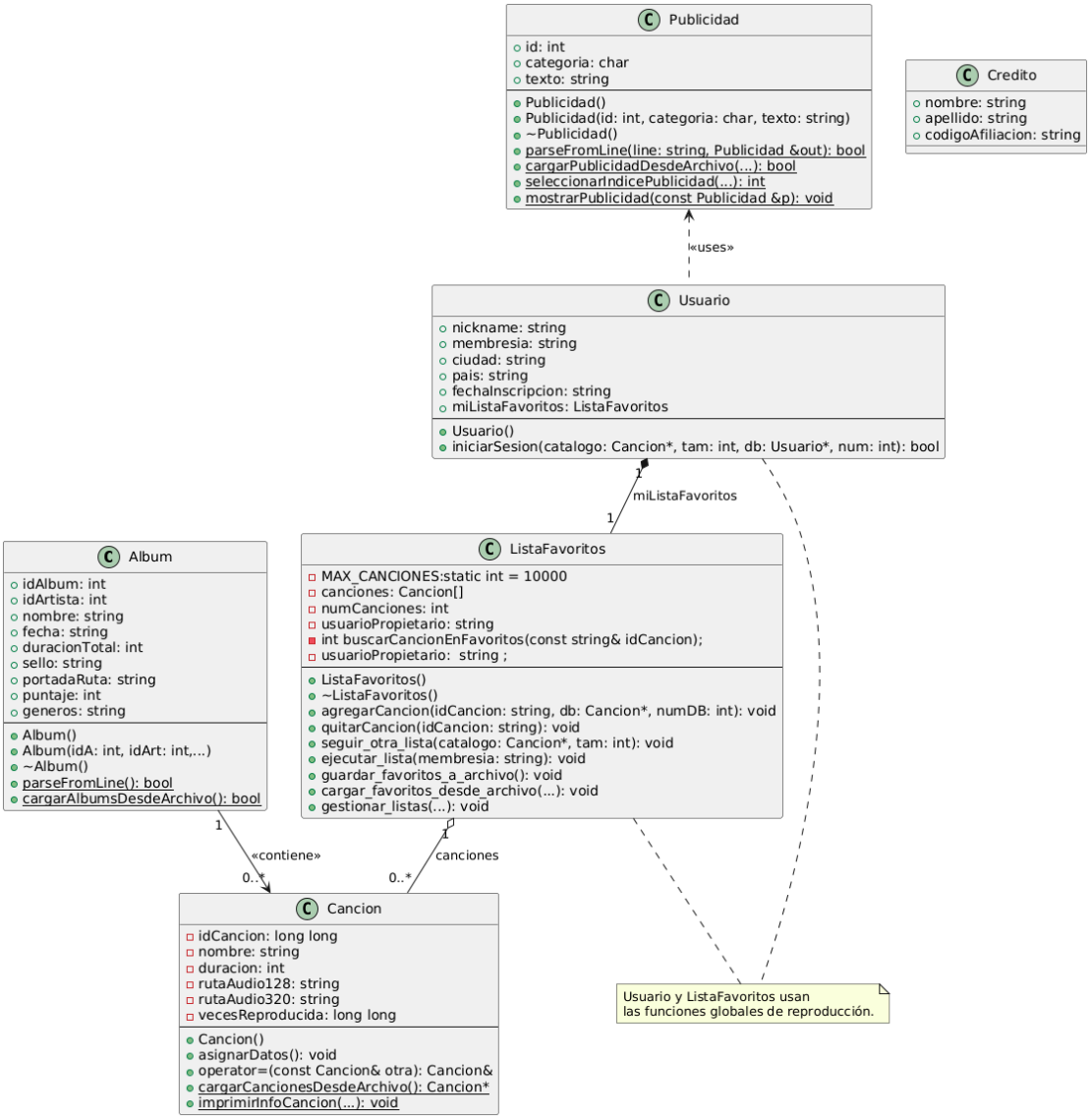
Estudiantes : Alexa Carolina Gamboa Mier y Manuela Arboleda Montoya

Semestre: 2025-2

Análisis del problema

El desafío planteado es crear un programa llamado UdeATunes el cual simula ser una app de reproduccion de musica el desafio nos pide implementar multiples funciones como reproducción de música en aleatorio , reproducción en orden secuencial, tener listas de canciones favoritas y poder modificarlas agregando o quitando canciones. Crear bases de datos organizadas que ayuden a la solución del desafío también es una parte esencial ya que las operaciones que realice nuestro reproductor como quitar o agregar una canción a favoritos tienen que guardarse.

Diagrama de clases final



Solución implementada

La solución planteada para este desafío es simple, crear una base de datos que contenga toda la información necesitada como información de los usuarios, datos de las canciones o las listas de favoritos de los usuarios, extraer la información de estas bases para almacenarla en arreglos dinámicos y facilitar su acceso, posterior a esto realizar funciones que operen con estos datos y estructurar el programa ,todo esto acompañado de una función que mide el consumo de recursos del programa el cual mostrará el número de iteraciones requeridas y el consumo de memoria.

Contenido general del código

El código implementa un Sistema de Gestión de Música simple, enfocado en la carga de datos (catálogo de canciones y usuarios), autenticación y gestión de listas de reproducción. Al inicio, carga todas las canciones y usuarios desde archivos, utilizando métricas internas para rastrear el consumo de memoria y el costo temporal (iteraciones). Una vez que el usuario se autentica con su nickname y fecha, el sistema carga su lista de favoritos (persistencia segmentada en archivo maestro) y presenta un menú principal donde puede reproducir el catálogo completo (secuencial o aleatorio, con la última siendo una función Premium) o gestionar su lista de favoritos (agregar, quitar, guardar y seguir listas de otros usuarios Premium). El programa finaliza liberando la memoria dinámica y presentando un informe final de métricas.

Base de datos

El siguiente esquema contiene la ruta relativa del archivo que contiene la base de datos , la estructura con la cual se denotan las credenciales de la base, un ejemplo de la sintaxis y el contenido general de la base de datos.

ruta relativa	sintaxis	ejemplo	contenido
\base_datos\canciones.txt	nombre, id , duracion , ruta Audio 128,ruta Audio 320, reproducciones	Hubiera,900111223,122,/users/storage/musica/ov7/crudo_128.ogg,/users/storage/musica/ov7/crudo_320.ogg,20030	canciones.txt contiene la información de cada canción , donde se especifican atributos como nombre, duración id, entre otras , cada línea del archivo contiene la información de una canción diferente
\base_datos\usuarios.txt	nombre, membresía , ciudad , país , fecha de inscripción	alexa,p,medellin,colombia,20-10-2025	usuarios.txt contiene toda la información de los usuarios del programa, el tipo de membresía se denota por “p” (premium) o “e” (estándar) , cada línea del archivo contiene la información de un usuario diferente
\base_datos\listas_favoritos.txt	[idCancion1,idCancion2,idCancion1].[idCancion1].[idCancion3]	[987650102,543210408,111223].[].[987650102,543210408,432100501,543210312]	listas_favortios.txt contiene todas las listas de favoritos de los usuarios cada lista se denota con [] y un punto para indicar el fin de la lista, los miembros estándar por defecto tiene una lista de favoritos vacía.
\base_datos\album.txt	idAlbum, idArtista, nombre,	01,Lugar	album.txt contiene los datos de cada

	duracion, fecha, sello, portada, puntaje, generos	Secreto,3600,2024-05-10,Sony Music,/users/storage/claudia_lorelle/image/lugar_secreto.png,9,Pop; Latina	álbum, cada línea representa un álbum con su identificador, artista, duración en segundos, fecha de lanzamiento, sello discográfico, ruta de portada, puntaje y géneros
\base_datos\artistas.txt	id, pais, seguidores, posición en tendencias	1,45,canada,15000000,2	artistas.txt registra la información de los artistas, contiene el id, edad, país, número de seguidores y posición en tendencias. El id permite relacionar artistas con albumes
\base_datos\publicidad.txt	idPublicidad, categoria, mensaje	4,C,"Prueba el nuevo sabor de café helado en tu cafetería favorita."	publicidad.txt lista de bloques de publicidad disponibles, cada línea tiene un id, categoria y un mensaje
\base_datos\creditos.txt	id, categoría(P/E), Nombre, Apellido, CodigoAfiliciacion	3,M,Carlos,Ruiz,C112233445	creditos.txt contiene la lista de las personas que participaron en la producción (Productores P, Musicos M, Compositores C). El codigo de afiliacion es alfanumérico de 10 caracteres y se valida durante la carga

Funciones principales y variables globales

El contenido siguiente es el nombre del archivo.cpp acompañado de sus funciones,variables globales y la explicación de estas

cancion.cpp

- **static string trim(const string& str):**str es la cadena de entrada. Retorna: string,función de utilidad que elimina los espacios en blanco iniciales y finales de la cadena dada. Utilizada para limpiar los datos leídos del archivo.
- **string Cancion::getIdentificador()** cons retorna string.Método getter que convierte y retorna el identificador numérico de la canción (idCancion, tipo long long) a una cadena de texto.
- **void Cancion::asignarDatos(nom, id, dur, r128, r320, rep.)**: Asigna todos los valores de los parámetros a los atributos internos de la instancia de la clase Cancion.
- **Cancion& Cancion::operator=(const Cancion& otra)**:Referencia constante al objeto Cancion a copiar. Retorna: Cancion&. Sobrecarga del operador de asignación (=). Copia los datos de la canción otra a la canción actual, manejando la auto-asignación.
- **void redimensionarArreglo(DatosCarga& datos)** : Referencia a la estructura DatosCarga que contiene el arreglo. Retorna: void. Duplica la capacidad del arreglo dinámico. Asigna nueva memoria, copia los datos antiguos y libera la memoria anterior.
- **Cancion* cargarCancionesDesdeArchivo(const string& nombreArchivo, int& tamano) :** Referencia de salida para el número de canciones cargadas. Retorna: Cancion*.Función principal de carga. Lee el archivo de canciones, crea objetos Cancion en un arreglo dinámico y gestiona el redimensionamiento. Retorna un puntero al arreglo cargado.

Listas_Favoritos.cpp

- **static string trim(const string& str)**: str es la cadena de entrada. Retorna: string. Función de utilidad que elimina los espacios en blanco iniciales y finales de la cadena dada. Utilizada para limpiar los datos leídos del archivo.

- **Cancion* buscarCancionEnDB(long long idCancion, Cancion* dbCanciones, int numDB)**: idCancion es el ID numérico de la canción a buscar; dbCanciones es el catálogo completo de canciones; numDB es el tamaño del catálogo. Retorna: Cancion*. Busca una canción por su ID dentro del catálogo principal (dbCanciones). Retorna un puntero a la canción si la encuentra, o nullptr si no.

- **int ListaFavoritos::buscarCancionEnFavoritos(const string& idCancion):** idCancion es el ID de la canción a buscar (en formato string). Retorna: int. Busca una canción en el arreglo interno de favoritos de la lista actual. Retorna el índice donde se encuentra la canción, o -1 si no está.

- **void ListaFavoritos::guardar_favoritos_a_archivo():** Retorna: void. Función Crítica que persiste la lista de favoritos actual del usuario en el archivo maestro (listas_favoritos.txt). Lee el archivo completo, encuentra el segmento del usuario propietario, actualiza ese segmento con la nueva lista de IDs, y reescribe todo el archivo.

- **void ListaFavoritos::agregarCancion(const string& idCancion, Cancion* dbCanciones, int numDB):** idCancion es el ID a agregar; dbCanciones es el catálogo principal; numDB es el tamaño del catálogo. Retorna: void. Intenta agregar una canción a la lista de favoritos. Verifica si ya existe y si el ID está en el catálogo. Si tiene éxito, llama a guardar_favoritos_a_archivo().

- **void ListaFavoritos::quitarCancion(const string& idCancion):** idCancion es el ID de la canción a eliminar. Retorna: void. Busca y elimina una canción de la lista de favoritos por su ID. Luego, reorganiza los punteros en el arreglo y llama a guardar_favoritos_a_archivo() para actualizar el archivo.

- **void ListaFavoritos::cargar_favoritos_desde_archivo(const std::string& nicknameUsuario, Cancion* catalogoCanciones, int tamanoCatalogo, int indiceUsuario):** nicknameUsuario es el dueño de la lista; catalogoCanciones es el catálogo principal; tamanoCatalogo es el tamaño del catálogo; indiceUsuario es la posición en la DB global. Retorna: void. Función Crítica que carga la lista de favoritos del usuario al iniciar sesión. Busca el segmento correspondiente al indiceUsuario en el archivo maestro, parsea los IDs y asigna los punteros de las canciones encontradas en el catálogo.

- **void ListaFavoritos::seguir_otra_lista(Cancion* catalogoCanciones, int tamanoCatalogo):** catalogoCanciones es el catálogo principal; tamanoCatalogo es el tamaño del catálogo. Retorna: void. Permite al usuario actual copiar las canciones de la lista de favoritos de otro usuario (Buscado por nickname). Busca la lista del usuario objetivo en el archivo maestro y agrega los IDs no repetidos a la lista actual.

- **void ListaFavoritos::editar_menu(Cancion* dbCanciones, int numDB, const string& membresia, int numUsuariosTotal):** dbCanciones es el catálogo principal; numDB es el

tamaño del catálogo; membresia es el tipo de membresía del usuario; numUsuariosTotal es el total de usuarios. Retorna: void. Presenta el submenú para Agregar o Quitar canciones de la lista actual.

- **void ListaFavoritos::ejecutar_lista(const string& membresia):** membresia es el tipo de membresía del usuario. Retorna: void. Muestra las opciones de reproducción (Secuencial/Aleatorio) y llama a las funciones globales de reproducción (reproducirEnOrdenSecuencial o reproducirAleatorioTemporizado).

- **void ListaFavoritos::gestionar_listas(Cancion* dbCanciones, int numDB, const string& membresia, int numUsuariosTotal):** dbCanciones es el catálogo principal; numDB es el tamaño del catálogo; membresia es el tipo de membresía del usuario; numUsuariosTotal es el total de usuarios. Retorna: void. Presenta el menú principal de gestión de listas, permitiendo reproducir, editar o seguir la lista de otro usuario.

metricas.cpp

- **void resetearContador():** Retorna: void. Pone el contador global de iteraciones (CONTADOR_ITERACIONES_GLOBAL) a cero, preparando la métrica para una nueva operación.

- **void incrementarContador(long long cantidad):** cantidad es el valor a sumar. Retorna: void. Aumenta el CONTADOR_ITERACIONES_GLOBAL en la cantidad especificada. Se utiliza para registrar el costo temporal de las operaciones de búsqueda y bucles.

- **size_t calcularMemoriaTotal(int numCancionesDB, int numUsuariosTotal, int numCancionesActuales)**: numCancionesDB es el total del catálogo; numUsuariosTotal es el total de usuarios; numCancionesActuales es el tamaño de la lista de favoritos. Retorna: size_t. Calcula la memoria estimada total ocupada por los datos principales del sistema (Catálogo + Usuarios + Punteros de Favoritos).

- **void mostrarMetricas(size_t memoriaTotalBytes, const string& funcionalidad):** memoriaTotalBytes es la memoria calculada; funcionalidad es el título del reporte. Retorna: void. Imprime en consola un resumen de recursos (Costo Temporal: Iteraciones y Memoria) para un punto específico de la ejecución del programa.

- **void mostrarMetricasFinales(size_t memoriaTotal):** memoriaTotal es la memoria final calculada. Retorna: void. Muestra el informe final y completo del programa, detallando el total acumulado de operaciones atómicas y el consumo final de memoria estimado.

reproductor.cpp

- **void imprimirInfoCancion(const Cancion& cancion, int indice, int total, const string& membresia):** cancion es el objeto Cancion a mostrar; indice es la posición actual en la lista (0-base); total es el número total de canciones; membresia es el tipo de membresía del usuario. Retorna: void. Imprime en consola la información detallada de la canción que se está

reproduciendo, incluyendo la ruta de audio de alta calidad (320 kbps) solo si el usuario es Premium.

- **void reproducirEnOrdenSecuencial(Cancion* lista, int numCanciones, const string& membresia, int numUsuariosTotal):** lista es el arreglo de canciones; numCanciones es el tamaño del arreglo; membresia es el tipo de membresía; numUsuariosTotal es el total de usuarios (no usado directamente). Retorna: void. Implementa la reproducción controlada canción por canción en el orden en que aparecen en la lista. Permite avanzar a la siguiente, retroceder (solo Premium) y repetir indefinidamente la canción actual (solo Premium).

- **void reproducirAleatorioTemporizado(Cancion* lista, int numCanciones, const string& membresia, int numUsuariosTotal):** lista es el arreglo de canciones; numCanciones es el tamaño del arreglo; membresia es el tipo de membresía; numUsuariosTotal es el total de usuarios (no usado directamente). Retorna: void. Implementa la reproducción aleatoria automática y temporizada. Reproduce un límite de K canciones (máximo 5) sin repetir, y luego ofrece un control manual de reproducción con funciones aleatorias y la opción de retroceder utilizando un historial limitado (solo Premium).

usuarios.cpp

- **Usuario::Usuario()**: Retorna: void. Es el constructor por defecto de la clase Usuario. Inicializa todos los atributos de cadena (nickname, membresia, etc.) a un valor vacío ("").

- **Usuario* cargarUsuariosDesdeArchivo(const std::string& ruta, int& numUsuarios):** ruta es la ruta del archivo CSV de usuarios; numUsuarios devuelve la cantidad de usuarios cargados. Retorna: Usuario* (puntero al arreglo dinámico de usuarios). Carga la base de datos completa de usuarios desde el archivo. Cuenta el total de líneas para dimensionar el arreglo, lo asigna dinámicamente y luego parsea cada línea usando stringstream para llenar los datos de cada objeto Usuario. Llama a incrementarContador(1) por cada línea leída.

- **bool Usuario::iniciarSesion(Cancion* catalogoCanciones, int tamanoCatalogo, Usuario* dbUsuarios, int numUsuariosTotal):** catalogoCanciones es la DB de canciones; tamanoCatalogo es el tamaño de la DB; dbUsuarios y numUsuariosTotal son la DB y el tamaño global de usuarios. Retorna: bool (true si el login fue exitoso, false si falló).

Publicidad.cpp

- **bool Publicidad::cargarPublicidadDesdeArchivo(const string &ruta, Publicidad *&outArray, int &outCount, long &iterations, size_t &memBytes, int maxMensajes):**
  La función carga y valida anuncios desde un archivo CSV ubicado en ruta y se construye en memoria un arreglo dinámico de objetos Publicidad. Crea un registro publicitario en el arreglo de salida, incrementa el contador de iteraciones y acumula una estimación de memoria usada

- **int Publicidad::seleccionarIndicePublicidad(Publicidad *arr, int cnt, int lastIndex):** arr es el arreglo de anuncios cargados; cnt es la cantidad de anuncios; lastIndex es el índice del anuncio mostrado previamente (para evitar repetición inmediata). Retorna: int (índice seleccionado o -1 si no es posible). Implementa selección aleatoria ponderada por categoría excluyendo preferentemente el lastIndex cuando hay alternativas; calcula el peso total

(sumando pesoPorCategoria), genera un número aleatorio en el rango de peso y devuelve el índice correspondiente; si no encuentra candidato válido devuelve un fallback distinto del anterior o -1

- **void Publicidad::mostrarPublicidad(const Publicidad &p):** p es el objeto Publicidad a mostrar. Retorna: void. Imprime en consola la categoría (AAA/B/C) y el texto del mensaje de forma legible, sirve como salida textual del anuncio para el reproductor, registra visualización en métricas a nivel de llamada externa si procede.

album.cpp

- **Album::Album(int idA, int idArt, const string& nombre, const string& fecha, int durTotal, const string& sello, const string& portada, int puntaje, const string& generos):** Constructor parametrizado. Parámetros: idAlbum, idArtista (0 si no aplica), nombre del álbum, fecha en formato YYYY-MM-DD, duración total en segundos, sello discográfico, ruta de portada, puntaje numérico y lista de géneros (separados por ';'). Retorna: instancia inicializada con los valores suministrados.
- **bool Album::parseFromLine(const std::string& line, Album &out):** soporta dos formatos de entrada: formato recomendado con idAlbum e idArtista explícitos (9 campos) y formato mínimo sin idArtista (8 campos, idArtista asumido 0). Realiza trimming de tokens, convierte campos numéricos con atoi, y construye la instancia out con valores normalizados. Ignora líneas que comienzan con '#'. Si faltan campos esenciales devuelve false.
- **bool Album::cargarAlbumsDesdeArchivo(const std::string& ruta, Album *&outArray, int &outCount, long &iterations, size_t &memBytes):** abre el archivo en ruta, lee línea a línea incrementando iterations; para cada línea válida llama parseFromLine y, si válida, redimensiona manualmente outArray (new[]/delete[]), copia elementos, añade el nuevo álbum y actualiza memBytes sumando sizeof(Album) y estimaciones de cadenas (nombre y portada). Ignora líneas vacías o comentadas y continúa tras errores de parseo. Al terminar cierra el archivo y deja outArray apuntando a un bloque contiguo con outCount elementos (o nullptr si no hay álbumes).

artistas.cpp

- **Artista::Artista(int id, int edad, const string& pais, long seguidores, int posicion, const string& nombre):** Constructor parametrizado. Parámetros: id numérico del artista, edad, país, número de seguidores, posición en tendencias y nombre. Inicializa álbumes a nullptr y albumCount a 0. Retorna: instancia Artista con los valores proporcionados.
- **bool Artista::addAlbum(Album* a)**: Parámetro a — puntero a Album (no nulo). Retorna: bool (true si se añadió correctamente). crea un nuevo arreglo de Album* con tamaño albumCount+1, copia los punteros existentes, añade al final, libera el arreglo anterior y actualiza albumCount. Maneja null-check y devuelve false ante entrada inválida.
- **bool Artista::cargarArtistasDesdeArchivo(const std::string& ruta, Artista *&outArray, int &outCount, long &iterations, size_t &memBytes):** abre el archivo, lee línea a línea incrementando iterations; por cada línea válida llama parseFromLine y, si válida, redimensiona manualmente outArray (new[]/delete[]), copia elementos, añade el nuevo Artista y actualiza memBytes sumando sizeof(Artista) y aproximación de memoria de cadenas (nombre). Ignora líneas vacías o comentadas y continúa tras errores de parseo. Al finalizar cierra el archivo y deja outArray apuntando a un bloque contiguo con outCount elementos (o nullptr si no hay artistas).

creditos.cpp

- **bool Credito::validarCodigoAfiliacion(const string& codigo):** Parámetro: codigo — cadena a validar. Retorna: bool — true si el código tiene exactamente 10 caracteres alfanuméricos, false en caso contrario. Uso: valida la integridad del código antes de crear registros de crédito.
- **bool Creditos::agregarProductor(Credito* c):** Parámetro: c - puntero a Credito no nulo. Retorna: bool true si se añadió correctamente. crea un nuevo arreglo de tamaño productoresCount+1, copia punteros existentes, añade c, libera el arreglo anterior y actualiza productoresCount; toma propiedad del puntero c.
- **bool Creditos::crearYAgregarProductor(const string& nombre, const string& apellido, const string& codigo):** Parámetros: nombre, apellido, codigo. Retorna: bool true si el código es válido y el productor fue agregado. Valida el código con validarCodigoAfiliacion, crea un nuevo Crédito y lo añade mediante agregarProductor.
- **bool Creditos::parsearLineaYAgregar(const std::string& linea):** Parámetro: linea cadena con una entrada del archivo creditos.txt. Retorna: bool true si la línea fue parseada y el registro agregado; false si está vacía, malformada o inválida. Tokeniza por comas aceptando formato con id opcional o sin id, aplica trim simple, normaliza categoría P/M/C, valida el código de afiliación y llama a la creación correspondiente para agregar el registro a la categoría adecuada. Rechaza entradas con categoría desconocida o código inválido.