Informe desafío 2
UdeATunes

Michell Dayana Gaitán Gutiérrez
Manuel José Montoya Arboleda

2025-2
UdeA
Informática II
Profesor Aníbal José Guerra Soler

Análisis del problema y solución propuesta

Debemos implementar un programa que simula las funciones de una plataforma de streaming de música, cuyas tareas principales son gestionar usuarios según su clasificación, almacenar información sobre artistas, álbumes y canciones y manejar correctamente listas de reproducción.
Para el planteamiento del programa analizamos que la declaración de las clases se haría en base a las entidades participantes en la ejecución que tuvieran comportamientos definidos o información que deba conservarse para las bases de datos o que influyera en la ruta de desarrollo de los procesos durante la ejecución, teniendo cada una de estas entidades su propia clase.
Durante el desarrollo decidimos que los conjuntos de información que serían cargados en memoria durante la ejecución deberían ser guardados como objetos almacenados en punteros a arreglos de punteros para poder eliminarse cuando dejaran de prescindir en la ejecución.

A continuación, se describen las principales clases encargadas de gestionar la lógica y datos del sistema:

Sistema
Es la clase principal del programa ya que contiene los métodos de despliegue que permiten la interacción de la plataforma con el usuario.
Esta encargada, entre otras cosas, de cargar la información de los usuarios y las canciones desde la base de datos al inicio de la ejecución para poder accederla y modificarla a conveniencia y de gestionar e invocar los métodos de cada clase correspondiente a las entidades que participan del sistema.

ListaCanciones
Una de las clases más importantes del sistema debido a que proporciona los métodos necesarios para gestionar las canciones existentes en la base de datos y las listas de favoritos. Contiene como atributo principal un puntero a un arreglo de punteros de tipo Canción, lo que permite almacenar las canciones existentes y acceder a ellas a conveniencia.
Al inicio de la ejecución se hace necesario cargar en memoria la totalidad de las canciones disponibles en el sistema, ya que se debe acceder constantemente a la información y métodos de las mismas para poder reproducirse, agregarse y eliminarse de las listas de reproducción, y generar las listas de reproducción seguidas, de modo que cuando inicia el programa se carga como atributo de la clase Sistema un apuntador a un objeto de tipo ListaCanciones que las contiene.
También se utilizan los objetos tipo ListaCanciones en la clase ListaFavoritos, ya que dentro de la clase ListaFavoritos se necesitan los mismos métodos de acceso y edición de las canciones que en las canciones que maneja la clase Sistema anteriormente mencionadas. Es entonces cuando se necesita acceso a una lista de favoritos poseída por cierto usuario que se crea un objeto de la clase ListaCanciones para manejarla. Se decidió aprovechar la existencia de la clase ListaCanciones para así simplificar significativamente los métodos de la clase ListaFavoritos, que con esta implementación quedan reducidos a invocar los métodos de ListaCanciones, sin perder la estructura que representa de una manera cercana a la realidad a las entidades participantes del sistema.

Usuario

Encargada de almacenar la información disponible de los usuarios del sistema. Contiene los métodos que crean los objetos tipo ListaFavoritos de cada usuario, gestionan los usuarios seguidos por los miembros premium y unen las listas de favoritos utilizando a la clase ListaFavoritos.

Al inicio de la ejecución se carga como atributo de la clase Sistema un apuntador a un arreglo de apuntadores de tipo Usuario, que se usa para acceder de manera fácil a la información y método de cada usuario en cualquier momento de la ejecución. Además, cuando un usuario inicia sesión en la plataforma, se carga como atributo de la clase sistema un apuntador a tipo Usuario para guardar el objeto correspondiente al usuario actual.

SistemaAnuncios

Es la clase encargada de cargar desde la base de datos los anuncios existentes.

Contiene los métodos para aleatorizar los anuncios e imprimirlos según su tipo de prioridad. Sus principales atributos son punteros a arreglos de punteros de tipo anuncio que guardan los anuncios separados por tipo de prioridad.

Guarda en arreglos los objetos de la clase anuncio, los cuales contienen la información de cada anuncio individual.

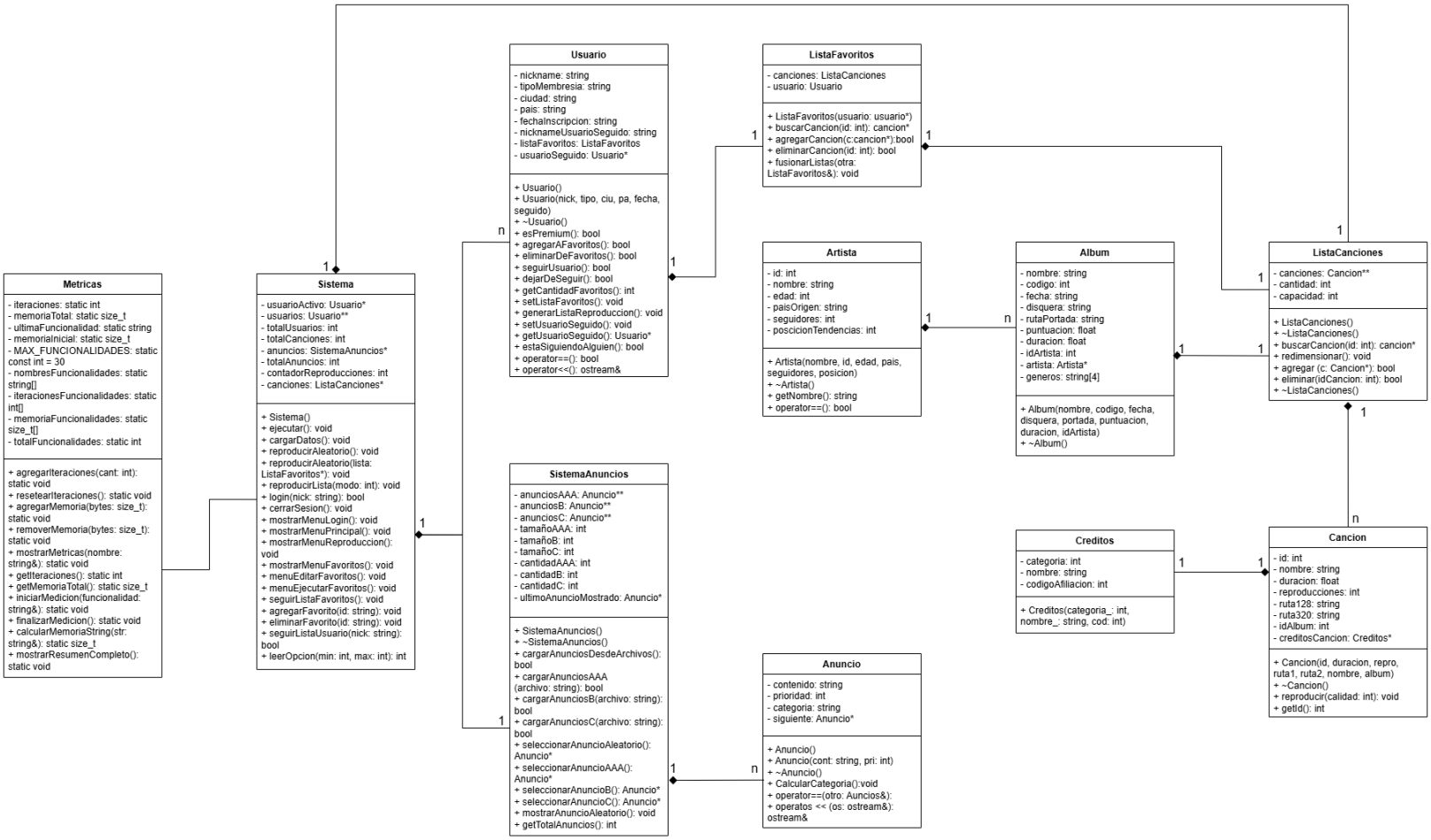
Metricas

Es la clase que gestiona los métodos que cuentan las iteraciones realizadas en cada proceso y la memoria utilizada. Sus métodos son invocados a lo largo de toda la lógica del programa y se imprimen los resultados obtenidos cuando se sale del sistema.

Otras clases

Las clases Artista, Album y Creditos son usadas únicamente como estructuras de datos para almacenar la información correspondiente a estas tres entidades. No contienen métodos que participen en la lógica principal del programa además de getters y setters.

Cabe mencionar que la clase Álbum crea los objetos Artista para cada álbum, ya que la manera en que las canciones muestran la información de álbumes y artistas cuando se reproducen es accediendo a los getters de los objetos correspondientes a dichas entidades, los cuales se cargan y eliminan al reproducir cada canción.



Estructura de los archivos de base de datos

Para hacer más legible y mantener la organización del programa evitando que los datos se confundan entre sí o sean leídos de una manera inadecuada, se decidió crear un archivo que almacene la información de cada entidad individual participante del programa de la siguiente manera:

albumes.txt
Cada línea, un álbum por línea:
código álbum; nombre álbum; código artista; nombre productora; duración; ruta a la portada; calificación

artistas.txt
Cada línea, un artista por línea:
código artista; nombre artista; edad; país; reproducciones; posición según su calificacion

canciones.txt
primera línea:

cantidad de canciones disponibles (Para poder reservar memoria cuando se guarden las canciones al inicio de la ejecución)

A partir de la segunda línea, una canción por línea:
código canción; nombre canción; código álbum; duración; ruta 128; ruta 320; cantidad de reproducciones

listas_favoritos.txt
Cada línea, una lista por línea:
nombre propietario; cantidad de canciones guardadas; código canciones guardadas...(separadas por ´;´)

usuarios.txt
primera línea:
cantidad de usuarios (para la reserva de memoria)

A partir de la segunda línea, un usuario por línea:
nickname; tipo de usuario; ciudad; país; fecha de inscripción; nickname usuario seguido

creditos.txt
Cada línea, una canción por línea:
código canción; codigo1; nombre1; afiliacion1; codigo2; nombre2; afiliacion2 ...

anunciosAAA.txt, anunciosB.txt, anunciosC.txt
primera línea:
cantidad de anuncios

A partir de la segunda línea, un anuncio por línea:
categoria; contenido

Todos los archivos se encuentran en la carpeta data.
La división de los anuncios en archivos diferentes según su prioridad se decidió para manejar de una manera correcta la aleatoriedad de las prioridades, accediendo y haciendo una lectura rápida del respectivo archivo cuando se obtenga el resultado de la se haya decidido qué tipo de anuncio se imprimirá.

Problemas presentados durante la implementación

Durante la implementación presentamos varios problemas de organización y lógica que pudimos resolver satisfactoriamente para el desafío.

Al principio del desarrollo la primera complejidad presentada fue la tarea de representar por medio de clases un problema tan complejo como el planteado en el enunciado, ya que se nos hizo difícil tener completamente claro desde un inicio qué clases serían necesarias y cuales podrían no utilizarse, debido a la cantidad de datos que debían ser guardados y el número de entidades que debían relacionarse entre sí. Fue en especial complejo identificar cómo debían relacionarse entre sí. Por ello nuestro diagrama de clases final es mucho más extenso que el inicialmente planteado, ya que durante el desarrollo identificamos que facilitaría la comunicación entre las clases poseer clases auxiliares como lo son ListaCanciones y SistemaAnuncios. También nos dimos cuenta de que varios métodos tendrían más coherencia y serían más fáciles de implementar si pertenecían a clases distintas a las definidas inicialmente, ejemplo de ello lo fue el cambio del método generarListaFavoritos (encargado de juntar las listas de favoritos de los usuarios que siguen a otros) que paso se hacer parte de la clase ListaFavoritos a la clase Usuario por la practicidad de poder invocarse con los datos del usuario cada que este se autenticara.

En etapas posteriores del desarrollo surgió confusión con los constructores de las clases y la cantidad de atributos de cada clase, ya que tuvimos que replantear algunos constructores debido a que algunos atributos que se necesitaban para su uso durante la ejecución no estaban disponibles o calculados al momento de crear el objeto, por lo que debimos agregar métodos set para dicha tarea, como en la clase ListaCanciones a la que se le tuvo que añadir el metodo setCapacidad. Además, tuvimos que analizar con mucha cautela el momento en que debían de invocarse dichos constructores, por ende, nació la idea de solo cargar al inicio del programa las canciones y los usuarios y construir los objetos tipo álbum, canción... solo en el momento en que se debían usar y liberarlos al poco tiempo.

Un elemento que fue difícil de controlar durante la implementación fueron las listas de favoritos, en especial la funcionalidad de juntar las listas de favoritos de dos usuarios. El problema fue causado por el mal manejo del atributo usuarioSeguido, ya que no se estaba cargando correctamente, lo que producía que no se conociese la información de dicho usuario y por ende la de su lista de favoritos. Para resolverlo se implementó en la clase Sistema que cuando un usuario iniciara sección se cargaran la lista de favoritos propia y la del usuario seguido consecutivamente, para así poseer la información de ambas para unirlas y reemplazar el resultado en los atributos del usuario autenticado en la plataforma.

Hacia la mitad del tiempo del desarrollo, cuando teníamos implementadas las clases necesarias para construir la lógica de funcionamiento, fue confuso realizar la comunicación entre las clases, porque, aunque ya existían los métodos de cada una, se debía generar otra lógica adicional que manejara de manera correcta las invocaciones a los métodos y los resultados de cada una, principalmente en la clase Sistema. Fue entonces la cantidad de atributos y métodos en cada clase y la cantidad de clases lo que produjo que fuera complejo identificar la manera en que las clases debían intercambiar datos entre ellas. Esto además

generó que creáramos bastantes métodos adicionales cuya implementación no teníamos contemplada inicialmente.