Informe de Análisis y Diseño del Sistema UdeATunes

Andrés Felipe Sierra Fernández

Octubre 2025

1. Análisis del problema

El proyecto **UdeATunes** busca desarrollar un sistema que simule el comportamiento básico de una plataforma de streaming musical. El sistema debe permitir la administración y reproducción de canciones, el manejo de usuarios (premium), artistas, álbumes, publicidad y listas de favoritos.

2. Objetivos del sistema

- Permitir el registro y autenticación de usuarios.
- Reproducir canciones de forma aleatoria o seleccionada(Esta ultima solo para usuarios Premium).
- Mostrar publicidad a usuarios estándar.
- Administrar artistas, álbumes y canciones.
- Permitir a usuarios premium crear y gestionar listas de favoritos.
- Medir el consumo de recursos del sistema (memoria e iteraciones).

3. Identificación de entidades principales

El sistema está compuesto por las siguientes clases principales:

- **Usuario**
- **Publicidad**
- **Artista**

- **Álbum**
- **Canción**
- **Sistema**

4. Diseño general

Cada clase se implementará de forma independiente, utilizando únicamente composición y punteros para la relación entre objetos. A continuación, se presenta una descripción resumida de cada clase:

- **Usuario**: contiene información como nombre, tipo de membresía, ciudad, país y fecha de registro.
- **Publicidad**: almacena la categoría y el mensaje asociado.
- **Artista**: contiene un código identificador, edad, país, número de seguidores y una lista de álbumes.
- **Álbum**: incluye código, nombre, géneros, fecha, duración total y canciones asociadas.
- **Canción**: posee un identificador único, nombre, duración, ruta del archivo y número de reproducciones.
- **Sistema**: administra las colecciones de usuarios, artistas, canciones y publicidades.

5. Aplicación de conceptos POO

El diseño propuesto incorpora los principios fundamentales de la programación orientada a objetos, adaptados a las restricciones del proyecto. A continuación, se describe cómo se aplican dichos conceptos dentro del modelo.

5.1. Constructores

Cada clase define al menos tres constructores:

- **Constructor por defecto:** inicializa los atributos básicos, asignando valores nulos o vacíos (`'\0'` en los arreglos de `char`).
- **Constructor parametrizado:** permite crear objetos con datos específicos (por ejemplo, un usuario con su nombre y tipo de membresía).
- **Constructor de copia:** asegura la duplicación correcta de objetos que manejan memoria dinámica (por ejemplo, duplicar la lista de canciones o álbumes de un artista).

5.2. Getters y Setters

Se implementan funciones *get* y *set* para acceder y modificar atributos privados, garantizando el principio de encapsulación. Por ejemplo:

```
void setDuracion(float d);
float getDuracion() const;
```

5.3. Sobrecarga de métodos

En clases como `Usuario` o `Sistema` se pueden definir métodos con el mismo nombre pero distintos parámetros. Ejemplo:

```
void agregarFavorito(Cancion* c);
void agregarFavorito(int idCancion);
```

5.4. Sobrecarga de operadores

Para facilitar la interacción entre objetos, se propone sobrecargar al menos dos operadores:

- **Operador de asignación (=):** permite copiar correctamente objetos con memoria dinámica, asegurando una copia profunda de los datos.
- **Operador de comparación (==):** facilita comparar objetos (por ejemplo, verificar si dos canciones tienen el mismo ID o si dos usuarios son iguales por nombre de usuario).

5.5. Ejemplo de aplicación

```
bool Cancion::operator==(const Cancion& otra) const {
return this->id == otra.id;
}

Usuario& Usuario::operator=(const Usuario& otro) {
if (this != &otro) {
strcpy(this->nickname, otro.nickname);
this->tipoMembresia = otro.tipoMembresia;
// Copiar lista de favoritos si aplica
}
return *this;
}
```

Estos mecanismos garantizan un uso apropiado de la programación orientada a objetos, sin violar las restricciones del proyecto y manteniendo un diseño modular, reutilizable y seguro en el manejo de memoria.

6. Diagrama de clases (UML simplificado)

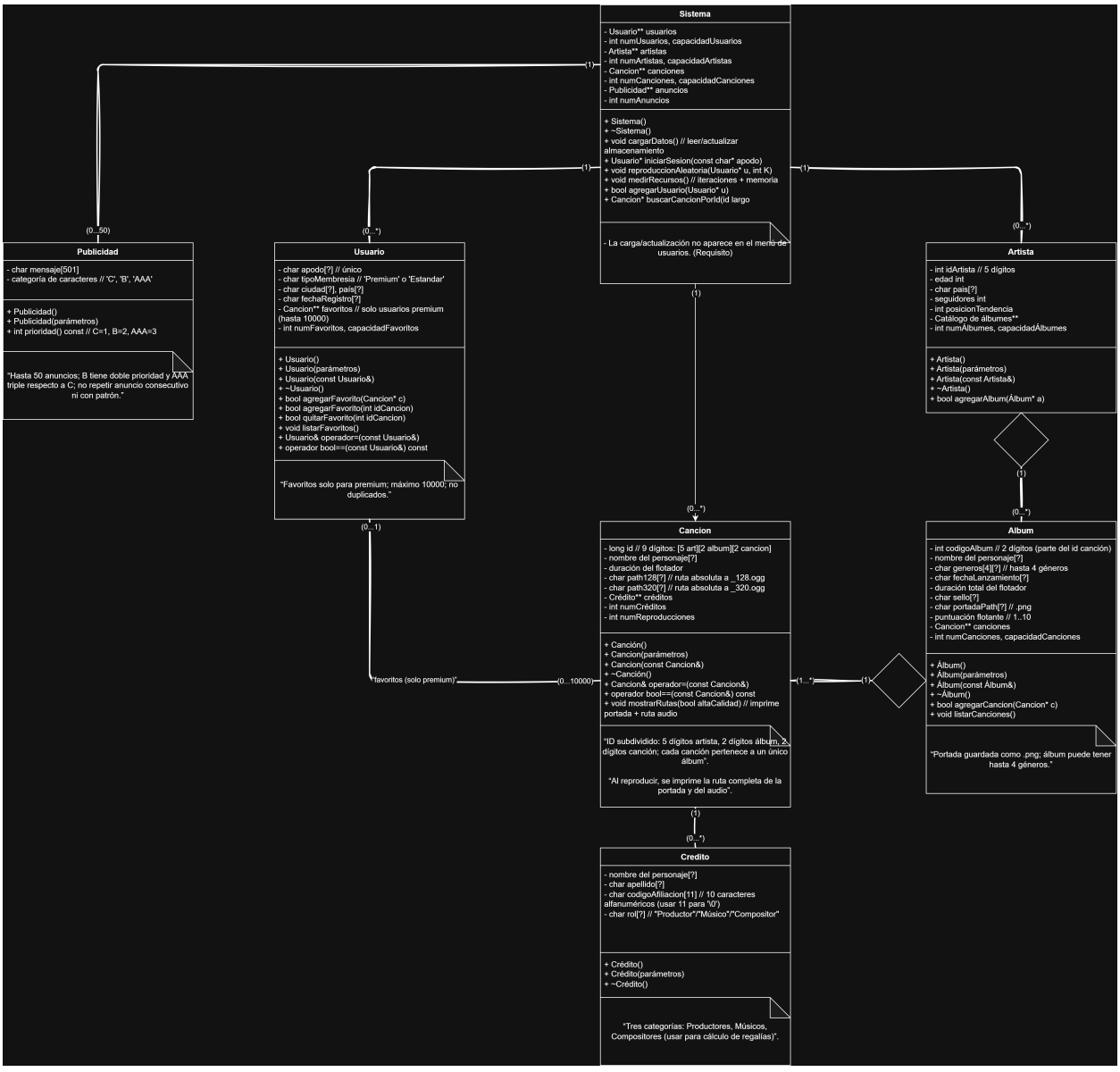

Figura 1: Diagrama de clases UML del sistema *UdeATunes*.

7. Lógica de subprogramas principales

Sistema::cargarDatos() - Carga la información desde archivos de texto usando arreglos de char.
Sistema::iniciarSesion() - Solicita nombre de usuario y tipo, retorna puntero al usuario.
Sistema::reproduccionAleatoria() - Reproduce canciones al azar y muestra publicidad si aplica.
Usuario::agregarFavorito() - Agrega una canción a la lista de favoritos.
Sistema::medirRecursos() - Mide iteraciones y memoria usada.

8. Conclusión

El diseño propuesto cumple las condiciones del desafío, garantizando la modularidad del sistema y la interacción entre clases sin recurrir al uso de STL().