# ANÁLISIS DESAFIO 2

Sergio Andres Chaves Roa, Oscar Alexander Sepulveda

Resumen—El proyecto UdeATunes es un prototipo educativo de servicio de streaming musical desarrollado en C++ aplicando principios de POO. El informe presenta el diseño del sistema y su propósito: gestionar de forma eficiente usuarios (con membresías estándar/premium), artistas, álbumes, canciones y publicidad en la plataforma. Se destacan como elementos clave la clase Aplicacion como núcleo organizador del sistema, el empleo de punteros para manejo dinámico de memoria (sin uso de la STL), y la implementación de relaciones UML (composición, agregación, asociación) entre clases. El alcance incluye la organización jerárquica del sistema mediante clases gestoras (GestorUsuarios, GestorCanciones, etc.), la separación modular, y la integración funcional del Reproductor. El diseño favorece un código claro y reutilizable, con encapsulamiento que oculta detalles internos

Palabras Claves— Programación orientada a objetos, jerárquica, plataforma, punteros.

## I. Introducción

En el contexto académico de *Programación Orientada a Objetos (POO)*, UdeATunes ejemplifica el modelado de un problema real: un servicio de música en línea. La POO permite agrupar datos con sus operaciones y ocultar detalles internos de los objetos, lo cual mejora la modularidad del software. A través de la implementación de clases y herencia, se logra además reutilización y escalabilidad.

El sistema UdeATunes, con requisitos de membresías, listas de favoritos y publicidad personalizada, justifica una estructura clara y bien definida. La solución propuesta se apoya en clases con interfaces públicas estables (encapsulamiento) y en la comunicación eficiente entre ellas mediante punteros (referencias directas) para optimizar el manejo de datos. De este modo se integra el concepto teórico de la POO a la problemática real del streaming musical, destacando la importancia de principios como encapsulamiento, modularidad y composición.

# II. OBJETIVO GENERAL

Diseñar el modelo de clases del sistema **UdeATunes** de forma que permita gestionar usuarios, canciones, álbumes, artistas y publicidad de manera jerárquica y modular, usando estructuras propias en C++ (con memoria dinámica y punteros).

## III. OBJETIVOS ESPECÍFICOS

Diseñar el diagrama de clases UML que refleje correctamente las entidades del dominio (Usuario, Cancion, Artista, Album, MensajePublicitario, etc.) y sus relaciones (composición, agregación, asociación, dependencia).

Organizar el sistema jerárquicamente con la clase *Aplicacion* como núcleo central del flujo de ejecución.

Implementar clases gestoras (como *GestorUsuarios*, *GestorCanciones*, *GestorArtistas*, *GestorAlbumes*, *GestorPublicidades*) que manejen colecciones de objetos mediante punteros, evitando la duplicación de datos.

Asegurar la gestión de la memoria de forma dinámica sin usar contenedores STL, garantizando eficiencia y control de recursos.

Fomentar la encapsulación y modularidad: cada clase debe proteger su estado interno y exponer sólo las operaciones necesarias.

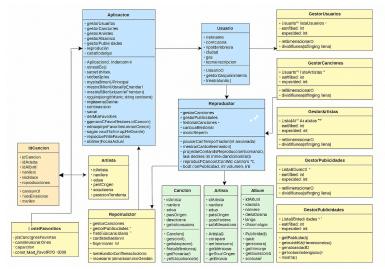


Fig. 1. Diagrama de clases análisis del desafío.

## IV. ANÁLISIS DEL PROBLEMA

El desafio plantea un servicio de streaming musical donde cada Usuario (identificado por nickname, tipo de membresía, ciudad, país, fecha de inscripción) tiene privilegios distintos según su categoría (estándar o premium). Entre las funcionalidades a atender están la reproducción aleatoria de Canciones, la gestión de listas de favoritos (solo premium), y la exhibición controlada de MensajesPublicitarios para usuarios estándar. Por ejemplo, por cada dos canciones reproducidas a un usuario estándar se debe mostrar un anuncio publicitario, el cual se selecciona de hasta 50 disponibles con prioridad ponderada según su categoría.

Además, deben almacenarse datos complejos: Artistas con su catálogo de Álbumes (cada uno con hasta cuatro géneros, fecha, duración, portada PNG, puntuación) y Canciones con identificadores estructurados por artista y álbum. El problema requiere además restricciones estrictas: no puede haber usuarios, canciones, ni álbumes duplicados; cada canción pertenece a un solo álbum; y no se permite el uso de la librería STL en la implementación.

Estas exigencias plantean la necesidad de un manejo eficiente de la memoria (con punteros y estructuras propias) y de un diseño cohesionado y escalable.

```
______
       UdeATunes
        USUARIO PREMIUM
._____
Bienvenido: checho
Ciudad: Medellin, Colombia
Calidad de audio: 320 kbps (HD)
_____
--- REPRODUCCION ---
1. Reproduccion aleatoria
  (Sin anuncios, controles avanzados)
--- MIS FAVORITOS ---
2. Ver mi lista de favoritos
3. Agregar cancion a favoritos
4. Eliminar cancion de favoritos
5. Seguir lista de otro usuario
6. Reproducir mis favoritos
--- BUSQUEDA ---
7. Buscar cancion por ID
8. Ver todos los artistas
9. Ver todos los albumes
10. Ver todas las canciones
--- MI CUENTA ---
11. Ver mi informacion
12. Cerrar sesion
0. Salir de la aplicacion
```

Fig. 2. Menú de usuario Premium..

# V. PLANTEAMIENTO DE LA SOLUCIÓN

La solución propuesta modela el sistema mediante clases especializadas y relaciones claras. La clase **Aplicacion** actúa como núcleo organizador: contiene punteros (composición UML) a los cinco gestores principales (*GestorUsuarios*, *GestorCanciones*, *GestorArtistas*, *GestorAlbumes*, *GestorPublicidades*). Cada *GestorX* gestiona dinámicamente sus objetos asociados.

Por ejemplo, *GestorUsuarios* crea y almacena objetos **Usuario**; *GestorCanciones* almacena objetos **Cancion**; y así sucesivamente. Estas relaciones «todo-parte» se implementan con composición o agregación: cuando el gestor *posee* objetos que él mismo crea, la relación es composición fuerte (los objetos se destruyen junto con el gestor); si el gestor solo referencia objetos creados externamente, se emplea agregación vía punteros.

En el diseño se emplean punteros en lugar de contenedores STL, lo que refleja agregación: por ejemplo, un Usuario mantiene punteros a sus Canciones favoritas, o un Album conoce punteros a sus Canciones. El Reproductor es un

componente funcional que interactúa con varios módulos: utiliza el *GestorPublicidades* para obtener anuncios, el *GestorCanciones* para acceder a las pistas de audio, y mantiene punteros al usuario activo.

La encapsulación se asegura declarando atributos privados y ofreciendo métodos públicos (getters/setters) para acceder a ellos. Gracias a esta separación por clases, el sistema evita redundancias: por ejemplo, el uso de punteros en agregación permite referenciar el mismo objeto **Artista** desde múltiples **Canciones** sin duplicarlo

#### VI RESULTADOS

El diseño resultante cumple con los objetivos: la organización modular del código facilita su entendimiento y mantenimiento. Cada gestor de clases implementa operaciones (carga, búsqueda, modificación) sin interferir con otros, respetando la ocultación de información. La interfaz pública de cada clase ofrece funciones claras para el usuario o la aplicación, logrando reutilización de código: las clases se pueden usar en contextos distintos sin cambios internos. La solución basa la robustez en la composición: por ejemplo, al destruir un *GestorCanciones* se libera toda la memoria de sus **Canciones** asociadas (garantizando manejo consistente de la memoria). En términos de eficiencia, el uso de punteros y estructuras propias permitió medir y controlar el consumo de recursos según lo requerido (sin sobrecarga de contenedores genéricos).

```
_____
      UdeATunes
     USUARIO ESTANDAR (GRATIS)
_____
Bienvenido: anibal
Ciudad: medellin, colombia
Calidad de audio: 128 kbps
_____
--- REPRODUCCION ---
1. Reproduccion aleatoria
  (Con publicidad cada 2 canciones)
--- BUSOUEDA ---
2. Buscar cancion por ID
3. Ver todos los artistas
4. Ver todos los albumes
5. Ver todas las canciones
--- MI CUENTA ---
6. Ver mi informacion
7. Hazte Premium! ($19,900/mes)
Cerrar sesion
0. Salir de la aplicacion
_____
Opcion:
```

Fig. 3. Menú de usuario Estándar.

## VII. ANÁLISIS DE LOS RESULTADOS

El empleo de punteros para implementar las relaciones tuvo impacto positivo en la robustez y escalabilidad. La agregación (uso de punteros a objetos existentes) evita copias innecesarias y garantiza referencias compartidas: como señala la documentación de POO, "la clase A incluye punteros a objetos de clase B creados fuera de A, objetos cuyo ciclo de vida no es gestionado por A". Esto reduce la duplicación de memoria y permite que varias clases accedan a un mismo objeto de modelo (por ejemplo, varias listas de reproducción referencian las mismas instancias de Cancion sin copias). La modularidad logra asimismo aislar fallos: debido al encapsulamiento de cada clase, cambios en la implementación interna (por ejemplo, en la estrategia de selección de publicidad) no afectan a otras (como los Gestores de usuarios o artista), preservando la integridad del estado global. Además, al controlar estrictamente el acceso a los datos internos (con atributos privados), se protege el sistema contra modificaciones indebidas. Como menciona la literatura, este control mejora la integridad y seguridad del sistema: un objeto sólo puede alterarse a través de sus métodos públicos, evitando estados inválidos. En conjunto, la separación en módulos independientes y el uso de composición refuerza la escalabilidad: se pueden agregar nuevas funciones (por ejemplo, nuevos tipos de listas de reproducción) sin modificar las clases ya establecidas.

### VIII. DISCUSIÓN

La implementación actual del sistema UdeATunes incorpora decisiones de diseño explícitas que merecen análisis. En particular, se eligió manejar dinámicamente los datos mediante **punteros crudos** y gestionar manualmente la memoria, renunciando al uso de la STL de C++. Esto otorga un control muy fino de los recursos, permitiendo optimizaciones específicas (por ejemplo, ajustar exactamente la asignación de memoria) en contextos donde el desempeño es crítico.. Sin embargo, esta libertad conlleva riesgos importantes: se incrementa la complejidad y la carga de trabajo del desarrollador, quien debe gestionar explícitamente la liberación de memoria para evitar fugas o punteros colgantes.

Adicionalmente, la jerarquía de clases está centrada en una clase "Aplicación" que coordina los subsistemas. Si bien esta organización facilita un punto central de inicialización y control, concentra múltiples responsabilidades en un solo componente. Como advierte la literatura de arquitectura, un diseño que concentra demasiada lógica en una clase puede degenerar en el antipatrón del "Objeto Dios", dificultando el mantenimiento y la escalabilidad del sistema.

En resumen, las decisiones de usar punteros crudos y una clase Aplicación como núcleo proveen flexibilidad y coherencia estructural, pero exigen cuidadosa ingeniería para mitigar sus inconvenientes.

## IX. Conclusión

En conclusión, el diseño del sistema UdeATunes logró organizar jerárquicamente las responsabilidades, con la clase Aplicacion como núcleo coordinador, y clases gestoras especializadas que interactúan mediante punteros. Se comprobaron los principios fundamentales de la POO: encapsulamiento para ocultar detalles internos, modularidad en la división del problema y **composición** para las relaciones todo-parte. El uso de punteros en C++ (en lugar de contenedores STL) permitió un control exhaustivo de la memoria dinámica tal como lo exigía el enunciado. Como resultado, el sistema es claro y extensible, con código reutilizable en distintos módulos. Entre las limitaciones identificadas figuran la complejidad añadida de gestionar manualmente la memoria (requisito educativo) y la falta de interfaz gráfica (se implementó en consola). En términos de aprendizaje, este proyecto reafirma la importancia de diseñar clases coherentes y de explotar mecanismos como la composición y la abstracción para desarrollar software robusto y eficiente.

## X. Referencias

- Especificación del desafío UdeATunes (Desafío II 2025-2), Departamento de Informática, UdeA.
- Wikipedia *Encapsulamiento (informática)* (consultado 2025)<u>es.wikipedia.org</u>.
- C. López, ¿Qué es la Programación Orientada a Objetos y cuáles son sus principios fundamentales?, CodersLink (2023) CODERSLINK, COMCODERSLINK, COM.
- J. M. Megino, Agregación vs Composición en Diagramas de clases UML, Blog SEAS (2013) SEAS. ESSEAS. ES.
- Curso de Programación 2 (UdeA): Notas sobre relaciones UML y punteros en C++pertusa, gitbooks, io.
- E. Gamma et al., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley (1994).
- G. Booch, The Unified Modeling Language User Guide, Pearson (2005).
- L. J. AGUILAR, PROGRAMACIÓN ORIENTADA A OBJETOS CON C++, DÍAZ DE SANTOS (2010).