

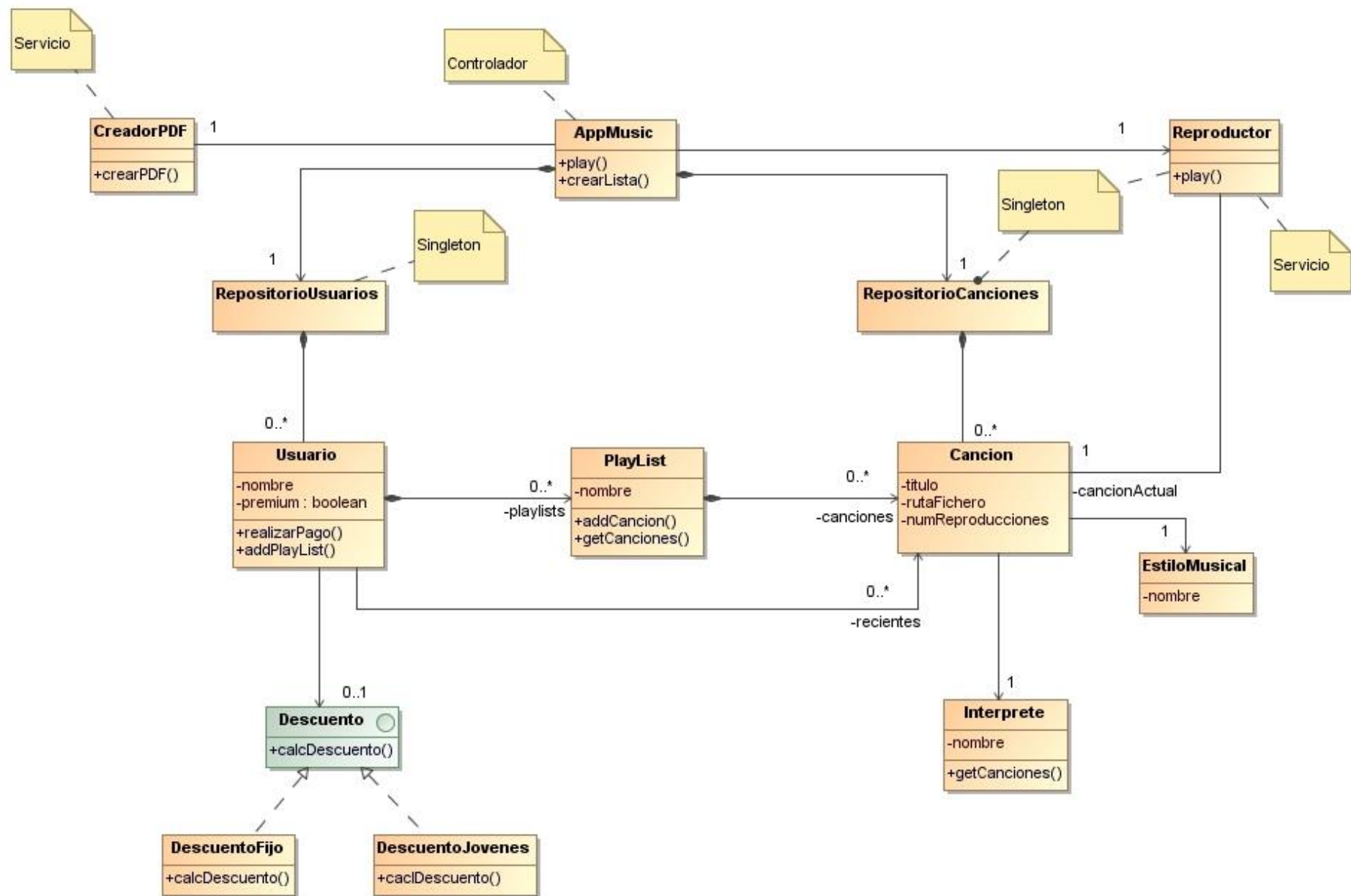
Grado en Ingeniería en Informática

Tercer curso

Tecnologías de Desarrollo de Software

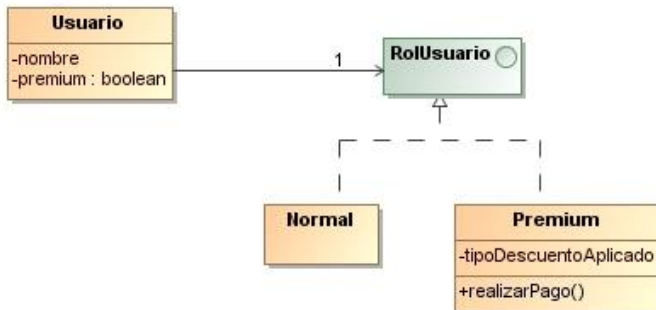
# Diagrama de Clases del Dominio del Caso Práctico “AppMusic”

Murcia, 9 de octubre, 2023



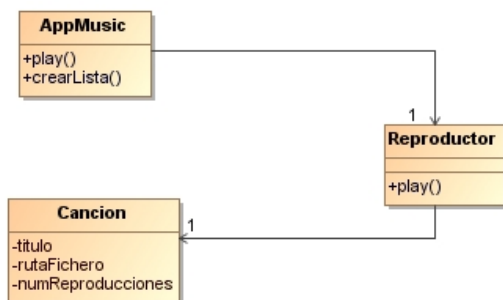
## Comentarios sobre algunas decisiones en el diseño del diagrama:

1. **Usuarios Premium.** Un usuario puede en cualquier momento convertirse en usuario Premium o dejar de serlo. Esta característica es un ejemplo de rol jugado por un usuario. La forma recomendada para representar roles de un usuario es asociando una jerarquía de roles a una clase como la que se muestra abajo.

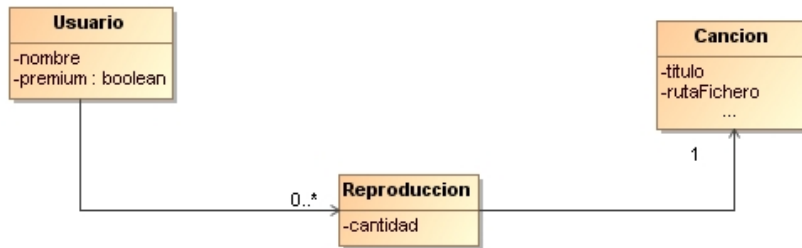


En este caso la funcionalidad que tendría la clase **Premium** y la existencia de un único rol nos ha motivado a considerar sólo una clase **Usuario**, con un atributo booleano que establece cuando un usuario es **Premium**.

2. **Clase Reproductor.** Se ha incluido la clase **Reproductor** que se muestra abajo. Esta clase sería un ejemplo de clase *Servicio* de DDD. En este caso la operación del negocio es reproducir canciones, la cual no debe ser parte de la clase **Canción** sino en una clase aparte que encapsula el acceso a la librería JavaFX.



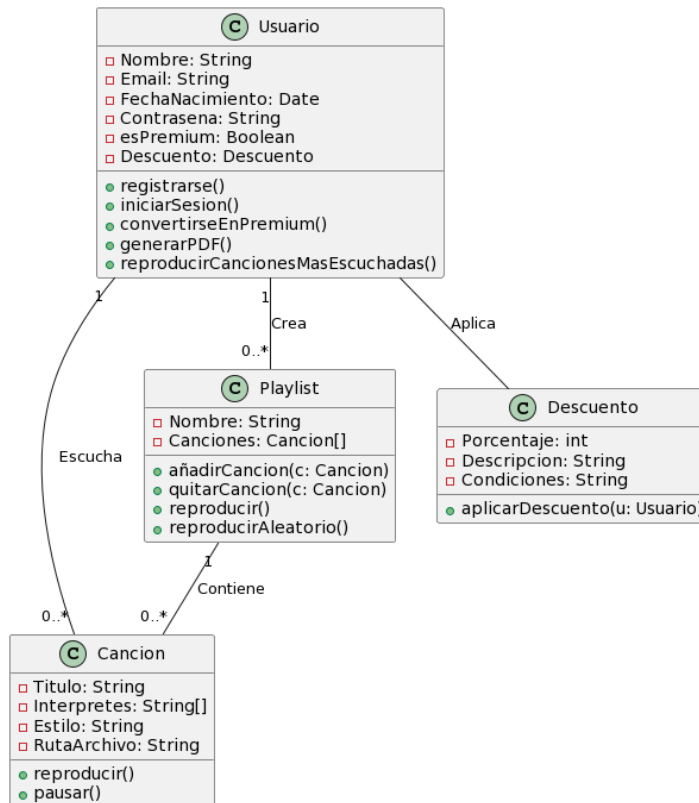
3. **Clase CreadorPDF.** Es otra clase *Servicio* encargada de crear un archivo PDF con información sobre las playlists de un usuario. En el diagrama, no se ha mostrado una asociación con **Playlist** para evitar cruzar líneas de asociación. Encapsula el acceso al API elegida para manejar archivos PDF.
4. **Canciones más reproducidas.** Un usuario Premium puede ver las canciones más reproducidas por **todos los usuarios**. Por ello, la clase **Cancion** incluye el campo `numReproducciones`.
5. Nótese que si se pidiese mostrar las canciones que más veces ha reproducido **un único usuario** sería necesario añadir una clase que registre ese valor. El diagrama de abajo, la clase **Reproducción** es la que juega ese papel. **Esta segunda posibilidad será opcional en el caso práctico.**



6. **Canciones recientes.** Hemos considerado que no es una playlist predefinida por una razón semántica, la lista de canciones recientes no es una playlist. Podría implementarse de ese modo, pero nos parece más apropiado que usuario las registre en una lista específica.
7. **Intérprete.** Se ha incluido ya que se trata de un concepto esencial del dominio de la aplicación, pero no será necesario considerarla en la implementación de AppMusic, dado que los requisitos no lo requieren. Sería necesaria si se debiese tener un registro de las canciones de un determinado intérprete. En AppMusic, el intérprete de una canción se registrará en un campo `String` de la clase Canción.
8. **Estilo musical.** En el diseño de clases del dominio de una aplicación es común tener que decidir si una información se representa como un atributo de tipo primitivo o enumerado o bien una clase. En este caso, deberíamos optar por representar los estilos musicales por medio de una clase `EstiloMusical` teniendo en cuenta que los estilos pueden variar y llevan asociados una lista de canciones de ese estilo. Pero, al igual que con `Interprete`, el estilo se registrará en un campo `String` de Canción.
9. **Repositorios.** Se mantendrán repositorios para los usuarios y las canciones (objetos raíz sobre los que se aplican operaciones de búsqueda, pero no para playlists. Caso de existir las clases `Interprete` y `EstiloMusical`, se podría tener un repositorio de objetos `Interprete` y otro de objetos `EstiloMusical` para realizar búsquedas más eficientes, por ejemplo, para encontrar las canciones de una categoría o de un intérprete.
10. Se han mostrado las **relaciones** más importantes; por ejemplo, `AppMusic` estará relacionada con casi todas las clases del dominio, pero sólo se han mostrado relaciones con los objetos servicio y repositorios. Del mismo modo, sólo se han incluido unos pocos **atributos y métodos** como es habitual en los diagramas de clase. Cada grupo deberá establecer cuidadosamente cuáles son los atributos y métodos de cada clase, y las relaciones entre clases.
11. Las clases persistentes serán `Usuario`, `Cancion`, y `PlayList`.
12. Como siempre sucede, se trata de un diagrama de clases inicial que podría evolucionar durante la implementación, entendemos que con pequeños cambios.

## Utilización de ChatGPT

Hemos pedido a ChatGPT lo siguiente: “Analiza la especificación de requisitos de abajo y crea el diagrama de clases del negocio. Dibuja el diagrama de clases en PlantUML” Y le hemos pasado el apartado “Requisitos” del documento de especificación del caso práctico. El diagrama de clases UML generado es mostrado abajo:



Y entre la información textual generada incluyó la siguiente lista de notas adicionales:

### Notas Adicionales

- La lista de canciones recientes puede implementarse como una Playlist con un comportamiento especial que maneje automáticamente su tamaño y las canciones que contiene basándose en la reproducción de canciones del usuario.
- La funcionalidad de login y registro podría requerir una clase adicional para gestionar la autenticación y la seguridad.
- La generación de PDF puede implicar una clase adicional que gestione la creación y formato del documento.
- Es posible que se requieran clases adicionales para gestionar la reproducción de música, como por ejemplo, un reproductor que maneje la lógica de reproducción, pausa, siguiente canción, etc.
- La funcionalidad para pagar y convertirse en usuario "Premium" puede requerir una clase adicional para gestionar pagos y verificar transacciones, aunque la especificación menciona que esto está fuera del alcance del proyecto.

Puede resultar extraño que se mantenga la lista de todas las canciones escuchadas por un usuario y se le puede preguntar por ello:

¿Por qué considerar que es necesario la relación "Escucha" entre Usuario y Cancion? ¿Qué funcionalidad la justifica?

**Respuesta de ChatGPT:**

Tienes toda la razón en cuestionar esa relación; siempre es crucial evaluar la necesidad y justificación de cada elemento en un diagrama de clases para evitar una sobrecarga de complejidad innecesaria. La relación "Escucha" entre **Usuario** y **Cancion** se incluyó pensando en las siguientes posibles funcionalidades o requisitos, aunque es correcto que no se mencionan explícitamente en la especificación de requisitos que proporcionaste:

1. **Historial de Escucha:** Para mantener un registro de las canciones que un usuario ha escuchado. Este historial podría utilizarse para implementar funcionalidades como:
  - *Recomendaciones:* Proponer canciones similares a las que el usuario ha escuchado previamente.
  - *Canciones Recientes:* Mostrar una lista de las últimas canciones que el usuario ha escuchado.
2. **Canciones Más Escuchadas:** Si se desea implementar una funcionalidad que muestre las canciones más escuchadas en la plataforma, se necesitará rastrear cada vez que un usuario escucha una canción.
3. **Análisis de Datos:** Para realizar análisis de los patrones de escucha de los usuarios, identificando tendencias y preferencias, lo que podría ser utilizado para mejorar la plataforma o para fines de marketing.

Entonces le preguntamos:

Las canciones más recientes podrán ser obtenidas si la canciones escuchadas se van añadiendo al final conforme se escuchan, pero las más escuchadas requeriría un cálculo que implicaría a todas las canciones de la lista para conocer el número de reproducciones. ¿Cómo lo harías realmente?

Y la **respuesta de ChatGPT** fue:

Tienes razón, obtener las canciones más escuchadas implicaría rastrear el número de reproducciones para cada canción. Aquí hay una forma en que podríamos modelar esto:

Y ofrece dos alternativas: el contador de número de reproducciones en la clase **Canción** o la clase **Reproducción** que hemos comentado arriba, discutiendo que la segunda sería más conveniente para una funcionalidad avanzada.

Nótese que en la primera pregunta, cuando pedimos el diagrama de clases del negocio, podríamos haber establecido un escenario más preciso, por ejemplo, haber indicado que se desea aplicar DDD y que se usará un controlador para la separación modelo-vista, y entonces la respuesta hubiese incluido objetos servicio y repositorio, entre otros cambios.