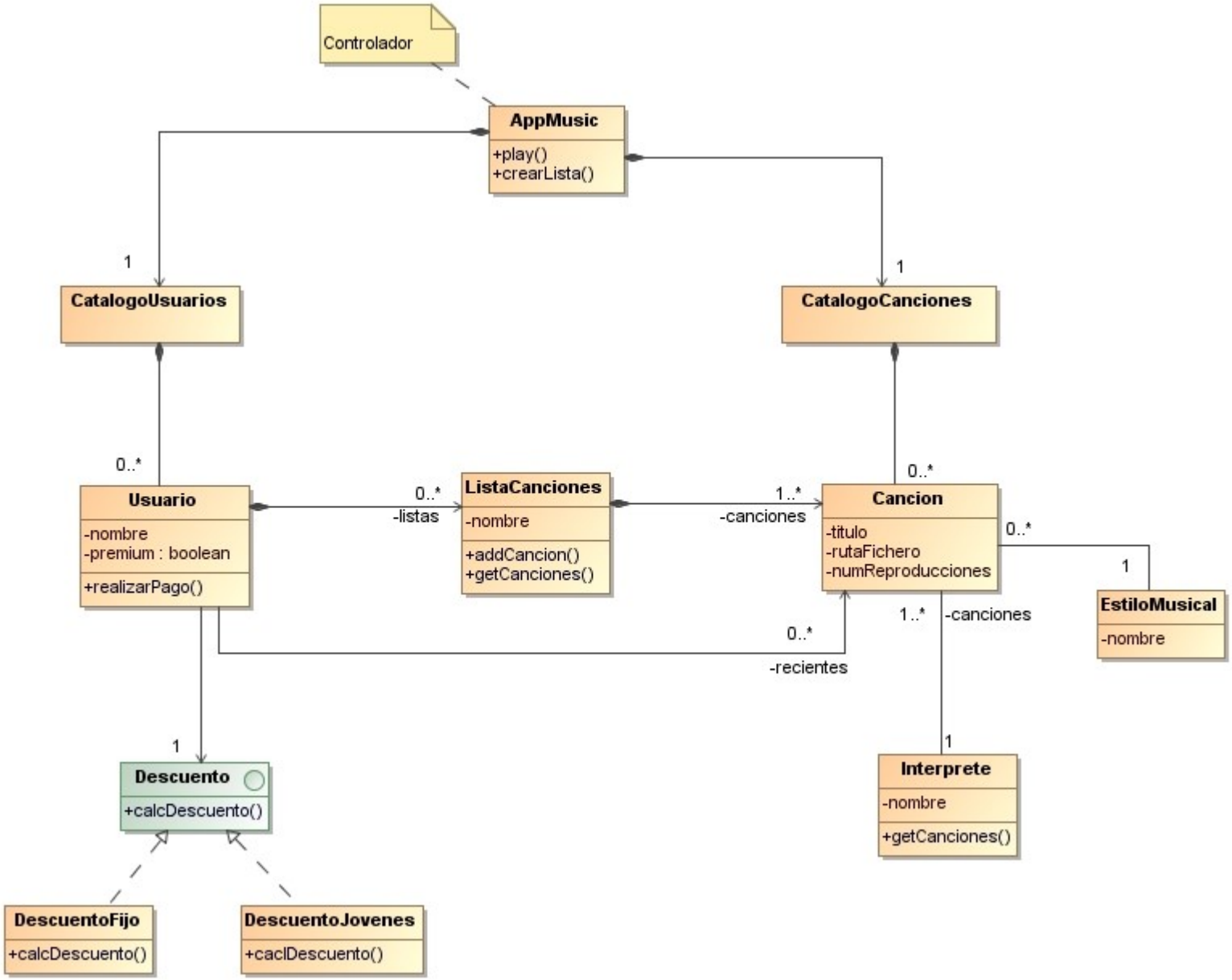
Grado en Ingeniería en Informática Tercer curso

Tecnologías de Desarrollo de Software

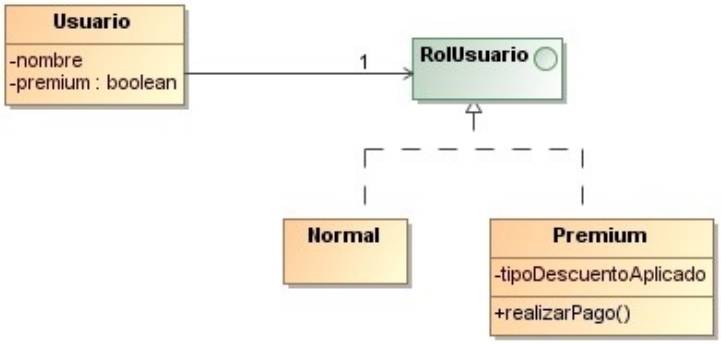
Diagrama de Clases del Dominio del Caso Práctico "AppMusic"

Murcia, 18 de octubre, 2020


Controlador
AppMusic
+play()
+crearLista()
1
1
CatalogoUsuarios
CatalogoCanciones
0..*
0..*
Usuario
-nombre
-premium : boolean
+realizarPago()
0..*
-listas
ListaCanciones
-nombre
+addCancion()
+getCanciones()
1..*
-canciones
Cancion
-titulo
-rutaFichero
-numReproducciones
0..*
1
EstiloMusical
-nombre
1..*
-canciones
0..*
-recientes
1
1
Descuento
+calcDescuento()
Interprete
-nombre
+getCanciones()
DescuentoFijo
+calcDescuento()
DescuentoJovenes
+caclDescuento()

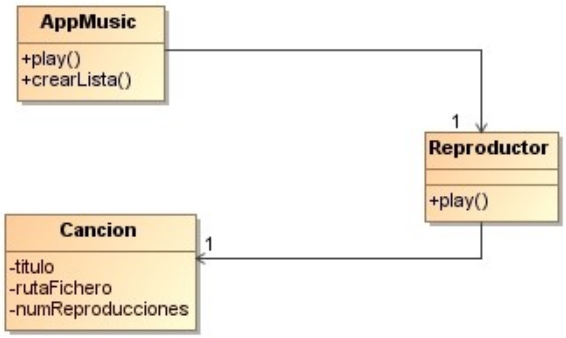
Comentarios sobre algunas decisiones en el diseño del diagrama:

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.

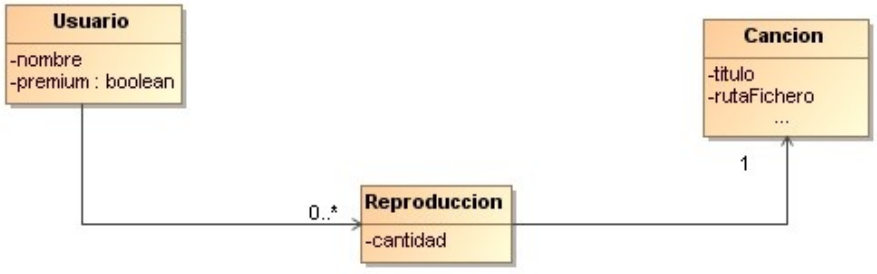


En este caso la funcionalidad que tendría la clase `Premium` y la existencia de un único rol nos ha motivado a considerar sólo una clase `Usuario,` con un atributo booleano que establece cuando un usuario es Premium.

2. Clase **`Reproductor`**. Se podría haber incluido la clase `Reproductor` que se muestra abajo. Esta clase podría ser considerada un ejemplo de clase *Servicio* de DDD. En este caso la operación es reproducir canciones que no debe estar en la clase `Canción` sino en una clase aparte. El hecho de que está clase utilizará el API de JavaFX para reproducir canciones nos ha llevado a no incluirla.



3. **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`. Si se pidiese mostrar las canciones que más veces ha reproducido **un único usuario** sería necesario añadir la clase `Reproducción` que se muestra abajo. **Esta segunda posibilidad será opcional en el caso práctico.**



4. **Canciones recientes**. Hemos considerado que no es una playlist predefinida dado que no se indica en los requisitos. Pero podría implementarse de ese modo.

5. **Intérprete**. La clase **`Interprete` no se tendrá en cuenta** y se registrará el intérprete de una canción en un campo `String` de la clase `Canción`, dado que no tiene funcionalidad asociada en nuestro caso. Se ha incluido en el diagrama de clases para mantener las canciones de un determinado intérprete y dado que se trata de un concepto esencial del dominio de la aplicación, pero no será necesario considerarla en la implementación de AppMusic.

6. **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 una clase o un enumerado. 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 `Cancion`.

7. **Catálogos**. Caso de existir las clases `Interprete` y `EstiloMusical`, se podría tener un catálogo 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.

8. Hemos 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 catálogos. 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.

9. Las clases persistentes serán `Usuario`, `Canción`, y `ListaCanciones`.

10. Como siempre sucede, se trata de un diagrama de clases inicial que podría evolucionar durante la implementación, entendemos que con pequeños cambios.