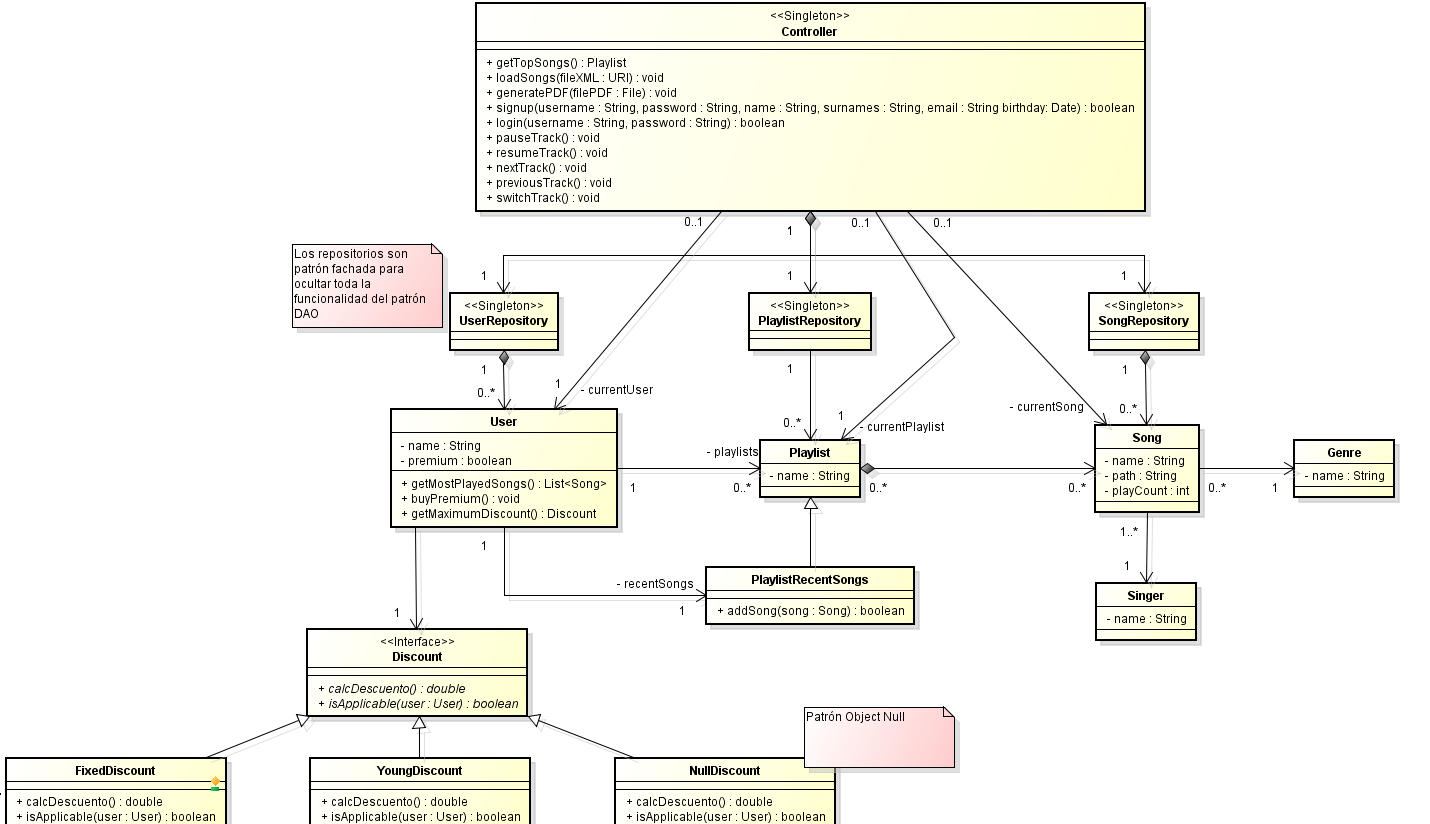
PORTADA

INDICE

# Diagrama de clases del dominio



Lo más destacable del diagrama de clases del dominio es lo siguiente:

* La clase Controller está conectado a tres repositorios (UserRepository, PlaylistRepository y SongRepository) que se encargan de abstraer la lógica de la persistencia.
* Controller, además, tiene una referencia al usuario conectado en ese momento y a la playlist y canción en ejercución.
* De la clase Playlist hereda otra llamada PlaylistRecentSongs que tiene un funcionamiento especial: solo permite 10 canciones y, en caso de estar llena, la playlist elimina la canción más antigua.
* La interfaz Discount es implementada por tres tipos posibles de descuentos: FixedDiscount, YoungDiscount y NullDiscount. Este último tipo de descuento representa la ausencia de descuento.

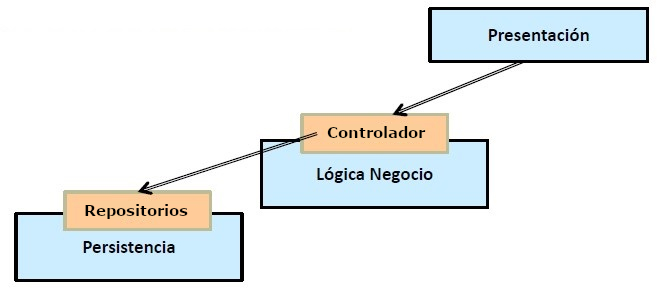
# Arquitectura de la aplicación

En el desarrollo de la práctica se ha aplicado una arquitectura de tres capas y el principio de la “**Separación modelo-vista**” (que es la base del patrón arquitectural “Modelo-Vista-Controlador”).

Se ha aplicado el principio “Separación modelo-vista” usando la clase Controller como una **Fachada** hacia las clases de la vista que conforman la interfaz de usuario (GUI). De esta forma, las clases que implementan la GUI únicamente interactúan con esta clase Controlador; sin embargo, esta no conoce ninguna de las clases de la vida. Permitiendo así el desacoplamiento entre las clases del dominio y de las interfaces de usuario. Estas últimas simplemente realizan llamadas a funciones del controlador tales como “loadSongs()”, “login()” o “buyPremium()”. De este modo, las clases de la interfaz solo se dedican a mostrar información y a capturar información del usuario, pero no implementan ninguna lógica del negocio, y las clases del modelo no tienen referencias a ninguna clase de la interfaz de usuario.

La arquitectura de la aplicación se organiza en tres capas:

* **Capa de vista o presentación**. Incluye las clases de la GUI que se encargan de mostrar la información y recopilar información del usuario. Estas clases están implementadas con la librería Swing de Java. Solo interactúa con la clase que actúa como Controlador.
* **Capa de negocio o dominio**. Esta capa contiene toda la lógica del negocio (correspondientes a las del diagrama mostrado arriba). La fachada de esta capa es la clase Controller (Controlador) que permite separar la interfaz de usuario de la lógica del dominio o de la aplicación.
* **Capa de persistencia**. Esta capa contiene las clases que se encargan del acceso al servicio de persistencia. Se ha utilizado el patrón DAO (Data Acces Objects) para conseguir que la aplicación sea independiente del servicio de persistencia usado. La fachada de esta capa son los distintos repositorios que serán usados por la clase Controller para realizar las operaciones CRUD (Create, Read, Update y Delete).



Las dependencias de librerías externas han sido manejadas con Maven y el proyecto ha sido almacenado en un repositorio privado de GitHub.

# Capa del dominio

La capa del dominio, negocio o modelo está formada por las clases que representan los conceptos del dominio (Playlists, User, Song…) y los repositorios (UserRepository, PlaylistRepository y SongRepository).

La clase **User** contiene una PlaylistRecentSongs

## Repositorios

SongCatalog y UserCatalog son Singleton.  
  
El adaptadorPlaylistDAO almacena si una playlist es normal o de recientes a través de un booleano "isRecent".  
  
Los repositorios tienen HashMap para realizar cachés. La política de escritura es escritura directa (cualquier cambio realizado en la  
caché será realizado inmediatamente en la base de datos). Los cambios primero se realizan en memoria y luego en el HashMap, de esta forma  
se pueden realizar los cambios a los códigos de las clases.

## Cargador de Canciones

Diagrama de componente

NullDiscount -> patrón Object Null

Repositorios, -> fachadas

Singleton ->

Discount -> Estrategia

Controlador -> Fachada para las vistas

CargadorCanciones -> Observer

Diagramas:

builder

# Controlador

La clase Controller es un Singleton.

# Repositorios o Catálogos

# Capa de persistencia o almacenamiento

Diagrama DAO

# Diagrama de colaboración para la operación de añadir una nueva playlist

# Indicación de los componentes utilizados y de su propósito y origen

safsdfsdf

# Tests unitarios implementados

Para garantizar el correcto funcionamiento de las distintas clases del programa, se definieron una serie de tests unitarios para controlar que la funcionalidad de ciertas clases y funciones esenciales no se viera afectada por los posibles cambios implementados en el futuro. Para generar los tests se utilizó la librería **JUnit 4**.

En primer lugar, se van a comentar los tests para las clases User, Playlist y Song. Todas las clases contienen tests para asegurar el correcto funcionamiento de los metodos "get", importantes para el correcto funcionamiento del resto del programa. Junto a estos tests se definieron pruebas específicas para cada clase: la clase User incluye tests para verificar el funcionamiento de la playlist "recentSongs", en concreto el tamaño máximo y orden. La clase Song comprueba que la funcionalidad de añadir reproducciones actualiza correctamente los valores internos, y la clase Playlist confirma que añadir y eliminar canciones funciona como corresponde.

La clase Discount incluye dos tests generales para todos los descuentos definidos en la aplicación. En primer lugar, se comprueba que la colección de descuentos devuelta por Discount.descuentos() no esté vacía y, a continuación, se comprueba que todos los descuentos existentes devuelven un precio final menor al precio original. La única excepción sería el descuento nulo, para el cual se define un segundo test (que comprueba que el precio no cambia).

Los DAOs de persistencia incluyen también una serie de tests, para comprobar que las funciones CRUD se ejecutan correctamente sobre el servidor de persistencia. Estas clases test tienen en común que requieren de métodos setUp y tearDown para preparar el servidor de persistencia antes de cada test. Para la preparación del servidor se utilizan métodos de la librería, y en los tests se utilizan los métodos de los objetos DAO para comprobar su efectividad. Cada clase comprueba el correcto funcionamiento de sus métodos store(), get(), set(), delete() y getAll(). Cabe mencionar que tanto los tests de AdaptadorPlaylist como AdaptadorUser incluyen varios tests del método delete(), dado que las relaciones que tienen con otros objetos almacenados en persistencia se deben eliminar con condiciones especiales (si se elimina un objeto User se eliminan los objetos Playlist asociados, pero no ocurre lo mismo con Playlist y Song).

# Manual de usuario

que incluya información sobre cómo se cargan e indique en quçe punto del código se establece el directorio para la carga de canciones locales.

Observaciones finales

sobre cualquier aspecto que los alumnos consideren relevante,  
como las tecnologías empleadas, el material disponible, etc, y deben incluir una estimación del tiempo dedicado.