ÍNDICE

1-. Comienzo y funcionamiento

1. Abre el proyecto y sincroniza los Gradles.
2. Coloca las canciones en "/storage/emulated/0/Music" o similar, dependiendo del emulador que uses, pero en la memoria principal compartida del dispositivo, si quieres conocer la ruta especifica de tu dispositivo ejecuta Environment.getExternalStorageDirectory() para comprobar el resultado que obtienes.
    - Puedes colocar carpetas con canciones o cualquier jerarquía de carpetas personal, leerá todos los archivos, consejo encarecidamente colocar únicamente canciones y a ser posible con metadatos, ya que la información para cada objeto de tipo Cancion se recoge de los metadatos.
    - Ruego evitar usar la API 33, he encontrado varios problemas con la API 33 que incluyó hace poco Android Studio a su IDE, la 31 es mucho más estable.
3. Al ejecutar el programa aparece la carátula donde tendrás los controles básicos y una canción reproduciéndose al instante, esta carátula es bastante intuitiva, hay que mencionar que el menú de esta carátula es 100% compartido con el menú del MainActivity, entraremos en detalle más adelante.
4. Si pasas a actividad de la lista de canciones encontrarás la interfaz para la lista de canciones con un MediaController incorporado. Si tienes alguna duda sobre el funcionamiento puedes usar la función "Acerca De" del menú, aunque comentaré que el FAB tiene la función de mostrar y ocultar el MediaController así como de traer la actividad de la carátula al frente si mantienes pulsado el FAB.
5. También encontrarás un control sencillo para la reproducción sobre una notificación que cambia con cada canción, permitiendo interactuar con el reproductor desde cualquier otra aplicación.
6. Si recibes una llamada o comienzas a grabar un video la reproducción se pausará gracias al uso del BroadcastReceiver.

2-. Clases y Actividades

Hay tres actividades corriendo en la aplicación, la MainActivity corresponde con la lista de canciones, la CaratulaActivity se corresponde con la Carátula y AcercaDeActivity simplemente muestra información sobre la aplicación y su funcionamiento básico.

Actividades

1. MainActivity es la primera que se inicia, y es la encargada de levantar todo lo necesario para hacer que el programa funcione:
    - Aplicación: es la encargada de la intercomunicación con distintos componentes de la aplicación.
    - Lanza la actividad CaratulaActivity.
    - Lanza la actividad AcercaDeActivity cuando accedes a la correspondiente acción del menú.
2. CaratulaActivity es la segunda actividad que se inicia, y se lanza automáticamente cuando PlayerService se lanza. Esta actividad únicamente lanza AcercaDeActivity cuando se accede a la correspondiente opción del menú.
3. AcercaDeActivity solamente es una actividad de información.

Clases:

1. Aplicación: Esta clase extiende de Application, por lo que será nuestro núcleo de comunicación entre actividades y clases, jugando un papel fundamental en la implantación de todo el sistema. Se encarga de inicializar y gestionar el acceso diferentes partes del sistema.
2. Cancion: Clase que contiene los atributos y métodos necesarios para cada canción.
3. ListaCanciones: Clase que contiene como atributo la lista de las canciones, contiene métodos propios de gestión y también implementa la interfaz RepositorioCanciones con la funcionalidad accesible desde fuera.
4. RepositorioCanciones: Interfaz que únicamente proporciona las funciones que se quieren hacer sobre una única clase, que sería la clase ListaCanciones.
5. CancionAdapter: Es la clase encargada de construir el Recyclerview de la actividad principal a partir de la interfaz RepositorioCanciones.
6. PlayerService: Clase encargada de inicializar el MediaPlayer y gestionar parte de su funcionamiento con Intents y listeners, así como del control de las notificaciones.
7. IntentManager: Clase que contiene los métodos para acceder a las intenciones para llamar a las Actividades con diferentes tags de acción, así como los métodos para gestionar el PlayerService por comandos/acciones.
8. ControlReproductor: Clase que contiene atributos y métodos necesarios para la gestión del reproductor, creando una lógica específica para diversas funcionalidades.
9. UsosMediaController: Clase que implementa la interfaz "MediaController.MediaPlayerControl", encargada de toda la funcionalidad del MediaController. Esta clase actúa sobre el único MediaPlayer inicializado en el servicio de PlayerService y obtiene su objeto desde la clase Aplicación, lo que permite su interfaz gráfica en MainActivity.
10. ItemMenuObserver: Clase que sincroniza los menús. Se encarga de controlar los eventos de cambios en los ítems del menú, así como de registrar los oyentes a los cambios de los ítems del menú y notificar a estos oyentes cuando ocurre un cambio en un ítem de menú.
11. OnItemMenuChanged: Interfaz que se implementa en MainActivity y CaratulaActivity para estar a la escucha de los cambios en los ítems del menú para que, cuando se cambie en una actividad también se cambie en la otra.
12. AdapterDataObserver: Esta clase se complementa con "notifyDataSetChanged();" para CancionAdapter. Registra los oyentes y notifica a estos cuando hay un cambio en el Adaptador para llevar a cabo las funciones pertinentes.
13. OnAdapterDataChanged: Interfaz que se implementa en CancionesActivity para que cuando cambie el orden de la lista se generen la portada de la segunda canción y se coloque en CircularImageView (Información dentro de AcercaDe) de la derecha, ya que el de la izquierda es la canción actual, actualmente en reproducción.
14. SongChangedObserver: Esta clase registra los oyentes y contiene las funcionalidades para los cambios del estado de la reproducción y notificar a los oyentes para cada acción realizada
15. OnSongChanged: Interfaz que se implementa en CancionesActivity para que cuando cambie el estado de reproducción se ejecuten las acciones pertinentes dependiendo de la acción: "onSongChanged()", "onSongPaused()", "onSongPlayed()".
16. OnBroadcastReceiver: clase que extiende de BroadcastReceiver y que contiene los escuchadores de eventos y acciones del dispositivo para realizar las acciones necesarias.

17. BroadcastRegister: Clase que establece y asocia el filtro de escucha para OnBroadcastReceiver.

3-. Conexión

La interconexión de las clases puede ser bastante compleja, solo tienes que mirar la imagen del diagrama UML con todas las clases… Así que atenderé a lo principal, ya que sabemos lo que hace cada clase, el resto de clases son de apoyo y utilidades.

Aplicación

Aplicación inicializa ListaCanciones (RepositorioCanciones), CancionAdapter, ControlReproductor, ItemMenuObserver, AdapterDataObserver, SonChangedObserver y BroadcastRegister para su funcionamiento, establece los get y set pertinentes para cada objeto para que solamente sea inicializado una vez.

MainActivity

MainActivity inicializa PlayerService, CaratulaActivity, PlayerService, UsosMediaController y AcercaDeActivity si es necesario. Obtiene de Aplicación CancionAdapter, ControlReproductor, ItemMenuObserver y le manda el objeto MediaPlayer a Aplicación una vez arrancado el servicio y obtenido el objeto MediaPlayer de este

CaratulaActivity

CaratulaActivity no inicializa nada, pero obtiene de Aplicación ControlReproductor, MediaPlayer (para la Seekbar), ItemMenuObserver, AdapterDataObserver y getSongChangedObserver

PlayerService

PlayerService inicializa el MediaPlayer y obtiene de Aplicación el ControlReproductor y SongChangedObserver

UsosMediaController

UsosMediaController obtiene de MainActivity el MediaPlayer ya inicializado, ControlReproductor, el MediaController con el control ya colocado sobre MainActivity y asignado el objeto MediaPlayer, y de Aplicación el SongChangedObserver.

4-. Funcionamiento principal

Voy a explicar el funcionamiento más complejo para que sea más ameno, el resto es más intuitivo teniendo en cuenta que MediaPlayer solo se instancia una vez en PlayerService y se reparte donde haga falta, mientras que MediaController establece el controller en MainActivity pero se gestiona desde MediaController.

Cambiar de cancion

El proceso de cambiar de canción se puede hacer de dos maneras, dado la accesibilidad de cada forma, por ejemplo, si lo queremos hacer desde la notificación obviamente debemos hacerlo enviando PendingIntents al servicio, pero en la base hace exactamente lo mismo que los demás métodos: Llamar al méotododo “siguienteCancion();” o “anteriorCancion();” que obtiene la canción que toca en su caso y la manda mediante una intención al reproductor valiéndose del IntentManager.

Una vez PlayerService recibe la orden, resetea el MediaController, le establece la nueva canción y genera el evento OnPrepare con el método ".prepare():". El listener que tiene activo el MediaPlayer para el evento "OnPreparedListener" ejecuta el método ".changeSong(MediaPlayer mediaPlayer)" que comienza la reproducción y notifica a los listeners de "OnSongChanged" y "OnSongPlayed" para que controlen la generación de portadas para CancionActivity y las imágenes de los botones de de CancionActivity.

Las clases que pueden cambiar de canción son:

- MainActivity (desde el RecyclerView con el método OnClick)
- UsosMediaController
- CaratulaActivity
- PlayerService (Con la notificación y el OnCompletitionListener de MediaPlayer)

Parar y continuar con la reproducción

Play Pause

Hay dos formas de pausar y continuar con la reproducción, pero todas se hacen usando la clase SongChangedObserver, que recibe el MediaPlayer en el método "playSong" o "pauseSong" y hace la función que toque sobre el objeto mediaPlayer y tras eso notifica la acción por el OnSong…Listener pertinente.

La primera forma la realiza el MediaController, que lo único que hace es llamar a estos métodos citados. La segunda es mediante Intents con acciones de "PLAYPAUSE", "PAUSE" o "REANUDE" dependiendo de la situación y necesidades.

Las clases que pueden pausar o reanudar la canción son:

- MediaController
- CaratulaActivity
- PlayerService

Stop

La única clase que puede parar la canción es CaratulaActivity. Manda un Intent con la acción "STOP" a PlayerService, quien usando el método "stopSong(MediaPlayer mediaPlayer)" para el reproductor y a demás notifica al listener de OnSongPaused, para realizar el cambio del icono playPausa de CaratulaActivity. Tras eso, llama al método "setStop(true);" de ControlReproductor para controlar el estado de parado.

Parámetros de reproducción

La clase que se encarga de la lógica de la reproducción es ControlReproductor, pero quien implementa los botones para que los parámetros de reproducción se cambien son MainActivity y CaratulaActivity mediante los elementos del menú, que además son compartidos. Así que voy a comentar solo un ejemplo

Sincronización

Al pulsar sobre uno de los botones de random o repetir, ambos iconos deben de ser modificados para ambas actividades, estos se gestionan con la implementación de la interfaz OnItemMenuChangedListener, que se notifica cada vez que se pulsa sobre uno de estos

botones. Esto se consigue gracias a que la clase Observer de este listener almacena el ID del recurso que se ha utilizado para cambiar el icono, y además el ítem (icono) modificado. Por lo que, cuando se cambia un icono, la clase que lo cambia almacena ambos datos en el objeto Observer, que también es compartido, y cuando se produce el evento solamente tienen que coger el ítem de su menú y el recurso del objeto Observer y cambiarlo en su Menú.

Orden Aleatorio

Este método tiene una particularidad, pero se encuentra en la clase ControlReproductor, y es que el método setRandom() es get y set, ya que cada vez que se pulsa el botón tiene que cambiar si o si el estado independientemente desde donde se quiera cambiar el orden, por lo que se utiliza un condicional con la llamada el setRandom de ControlReproductor.

SetRandom o cambia el valor de random y ejecuta el método “ordenarLista” de la interfaz RepositorioCanciones pasándole como parámetro el valor de la variable random. Por su parte OrdenarLista ordena la lista dependiendo del valor obtenido.

Tras ordenar la lista, el método llama a Aplicación para ejecutar el método notifyDataChanged(), quien llama a notificarAdapter() de la clase AdapterDataObserver(CancionAdapter adapter), que ejecuta la notificación de que se han producido cambios en la lista que está usando el adaptador (“ adapter.notifyDataSetChanged();” y también notifica a los oyentes de OnAdapterDataChanged, que es el ControlReproductor solamente, quien va a generar la segunda carátula (ya que la primera está sonando). Este proceso lo dejamos para más adelante, pero se ve realicionado con este proceso, por lo que acabas de ver, por lo que es importante tenerlo en cuenta más adelante.

Tras haber hecho las notificaciones el método setRandom de ControlReproductor establece el orden a 0 y devuelve el estado de random, y dependiendo del resultado de Random ejecuta el proceso de sincronización anteriormente descrito.

Repetir

Esta opción de menú es más sencilla. Se repite la situación en la que el get y el set de repetir están en el mismo método de ControlReproductor.

Se llama a setRepetir, se cambia el valor y se devuelve el valor, después se realiza el proceso de sincronización anteriormente mencionado.

Caratulas de CaratulaActivity

Las caratulas de esta actividad se modifican cada vez que se cambia de canción o se realizan cambios en la lista de reproducción, como ya hemos visto anteriormente.

Una vez notificado uno de estos eventos por AdapterDataObserver o por SongChangedObserver, se realiza las siguientes acciones: Se llama al o los métodos (dependiendo del listener) generarCaratulaX() de ControlReproductor, quien obtiene la canción o canciones pertinentes para extraer su imagen embebida y guardarla como atributo o atributos de la clase, guardando “null” en caso de que la canción no tenga imagen embebida.

Luego, CaratulaActivity ejecuta la función de “ponerCaratula2()” o “ponerCaratulas()” dependiendo del listener, que obtiene la o las carátulas de ControlReproductor mediante la función “getCaratula1()” o “getCaratula2()” y comprueba que no sea null.

Si es null pone una carátula por defecto a la imagen concreta, y si no es null pone la carátula obtenida de ControlReproductor.

Dificultades encontradas

La principal dificultad es comprender como funciona Android, sus herramientas, que hace cada cosa, buscar soluciones que se ajusten a mi proyecto, aprender de verdad, vamos.

Voy a ser objetivo aquí, y además me vas a perdonar por el vocabulario, pero es que mis lugares java es una santa mierda de contenido para aprender. Donde realmente hemos aprendido en el tema3, Y NO POR MIS LUGARES JAVA, sino por los diferentes ejercicios que has ido mandando, sobre el resto no hemos aprendido absolutamente una bazofia.

Considero que este proyecto nos ha servido mucho más, o al menos desde mi postura, que todo el temario de mis lugares, nos hemos encontrado con información desorganizada y casi ya dada. Entiendo que la programación es información, leer y absorber herramientas, pero no se va a hacer latente si no se practica como hemos hecho con este proyecto, que además me ha servido para entender un poco mejor ciertas funcionalidades de java base, no solo de Android, pero es verdad que si lo realizas por tu cuenta hay que dedicarle mucho tiempo, porque te encuentras con cada problema que es la hostia con el nivel del que hemos partido en Android.

Voy a hacerte una sugerencia personal que puedes seguir o no:

- Si el año que viene vuelves a dar Android, el tema 1 deberías de prepararlo o por lo mejor organizarlo mejor de lo que estaba el PDF, porque todos gastamos más tiempo intentando seguir el PDF que desarrollando.
- A partir del tema 3 dar el tema 4 a toda hostia y que los alumnos empiecen con un proyecto, puedes quitar el examen, van a tener tiempo para desarrollar y poner en práctica nuevos conocimientos adquiridos, el tema 5 lo puedes olvidar. Yo a este proyecto le he metido más de 40 horas buscando soluciones que me cuadrasen y enterándome bien de qué hace cada cosa que hago. Puedes incluso quitar el tema 4 y dar herramientas con las que puedan trabajar sin dar el tema 4, porque luego llegamos al proyecto y tenemos que buscar que hace exactamente y cómo se comporta, en tanto qué opciones abarca lo que hemos hecho y dado en el tema 4.

Explicada la primera dificultad, que es entender qué se ha hecho y cómo funciona lo desarrollado en 4 meses de trabajo y más allá de lo dado en mis lugares java pues entender cada cosa que nos has mostrado que existe, como el MediaController o el MediaPlayer, cosa sencilla.

Lo más difícil ha sido el servicio de reproducción, sin duda alguna. Me ha supuesto más de 10 horas de buscar información de su uso y funcionamiento, así como de implantación y crear una lógica que sea posible en mi solución. Más tarde he entendido que podría haber utilizado el MediaPlayer incluso directamente en la clase de ControlReproductor y dejar el servicio de reproducción como un servicio de notificaciones normal y corriente manejando los PendingIntents directamente hacia ControlReproductor, pero bueno. El caso es que encontré la forma de levantar el servicio usando la clase ServiceBinder para luego en el MainActivity recuperar el objeto ServiceBinder usando la clase ServiceConnection y con eso hacer todo lo que he hecho.

Otra dificultad, por ejemplo, ha sido la sincronización de menús o la sincronización del MediaPlayer con las carátulas. Para esto me estuve informando y descubrí que podía hacer Listeners yo mismo para implementar avisadores donde yo considere oportuno. Esto me hizo un poco estallar la cabeza de cómo podía hacer la sincronización de todo a la vez, el menú con el reproductor, con las carátulas con el adaptador, pero lo implanté bastante rápido una vez conseguida la información teniendo en cuenta todas las variables lógicas de la aplicación. Para la sincronización los botones de Play/Pausa de la carátula ya fue un paseo.

Luego por comentar, algo más light fue el tema de los Intents fuera de actividades o para evitar que ejecuten el OnCreate de una actividad, pero nada, eso es un tag y para adelante.