G53MDP Coursework 1-2 MP3Player

Summary

In this exercise you are required to build an Android MP3 player application. This is an assessed exercise and will account for **10% of your final module mark**. This is an individual coursework, and your submission must be entirely your own work – please pay particular attention to the section of this document regarding plagiarism. This document sets out general requirements and broad instructions for developing the application.

Your application should be submitted no later than:

- **3pm on Monday the 19th of November 2018**

Submissions should be made electronically via Moodle. Standard penalties of 5% per working day will be applied to late submissions.

Your application should be submitted as a .zip or .tar.gz file containing all relevant source code, configuration and related files, and a compiled .apk file – i.e. the contents of the directory containing your Android Studio project. Do not submit RAR files.

Specification

You should create an application with the functionality of a simple music player for Android, which allows users to select from a number of music files stored on the SD card storage of the device to be played, and allows the music to continue to play in the background while the user performs other tasks.

Your application should consist of:

- An *Activity* presenting an interface for the user that:
    - Displays and allows the user to select from and play music files from the /sdcard/Music folder
    - Allows the user to stop or pause playback
    - Displays the current progress of the playback (i.e. the elapsed time)
- A *Service* that provides continued playback in the background
- A *Notification* that allows the user to return to the *Activity* while the music is playing

You must implement a **Service** to handle the music-playing element of the application, as this is a long-running task and the user can be expected to leave the initial activity to perform other tasks. You should think carefully about the relationship between the Activity and Service in your application, and how these should be used appropriately to perform the task. There is **no requirement** that your service will be used remotely, i.e. you do not need to use *AIDL*.

A simple MP3Player class is provided that wraps a basic MediaPlayer object for loading and playing an MP3 file. It is left up to you to decide how best to design and implement Activities for selecting and controlling the music playback.

~~Your application must be written in Java and make use of the Android SDK. There are no requirements to target a specific Android API version, however you should assume that your application would be tested on an emulated Nexus 6 device (1440x2560 560dpi) running Android API version 23 (Android 6.0 Marshmallow).~~

You should consider the following when implementing your application:

- Appropriate use of Activities, Intents and appreciation of the Activity life- cycle
- Appropriate use of Widgets and ViewGroups for layouts that support devices of differing screen sizes and resolutions
- Appropriate use of Services, Notifications and appreciation of the Service life-cycle
- Your application should have appropriate comments and variable / class names, so that a reader can easily understand how it works at the code level

Your application must be written in Java and make use of the Android SDK. There are no requirements to target a specific Android API version, however you can assume that your application will be tested on an emulated device (1080 x 1920 420dpi) running Android API version 28 (Android 9.0 Pie).

Assessment Criteria

As this is a constrained exercise marks are awarded for achieving specific functionality as follows. For all elements either 0 or full marks are awarded as appropriate. There are no additional marks available for additional functionality in this exercise:

	Marks
The application has an Activity that displays the contents of /sdcard/Music/	1
The Activity allows the user to select, play, pause and stop a track	2
The Activity displays the current progress of playback	3
The application displays a notification that returns the user to the Activity	2
The application has a Service that handles playing the track	4
The Service has an interface that can be bound to that controls playback	4
The Service continues to play the track when the Activity is destroyed	2
The Service is stopped when it is no longer needed	2
Total	20

Plagiarism

N.B. Use of third party assets (tutorials, images, example code, libraries etc.) MUST be credited and referenced, and you MUST be able to demonstrate that they are available under a license that allows their reuse.

Making significant use of tutorial code while referencing it is poor academic practice, and will result in a lower mark that reflects the significance of your own original contribution.

Copying code from other students, from previous students, from any other source, or soliciting code from online sources and submitting it as your own is plagiarism and will be

penalized as such. FAILING TO ATTRIBUTE a source will result in a mark of zero – and can potentially result in failure of coursework, module or degree.

All submissions are checked using both plagiarism detection software and manually for signs of cheating. If you have any doubts, then please ask.

Instructions

Note that this coursework aims to serve as an assessment of the Services material covered in lectures and used in lab exercises 3 and 4 – as such try to think about how to make use of the concepts covered in those.

MP3Player

Begin by creating a new application in Android Studio as usual.

Add the class MP3Player.java to your app project. This class is available on Moodle or at the URL below:

https://github.com/mdf/g53mdp/blob/master/MP3Player/MP3Player.java

You can either copy the file directly into your project's source directory (*projectname/app/src/main/java/…*) or create a new MP3Player Java class in your project and copy / paste the code into it, updating the package qualifier accordingly.

MP3Player is a simplistic wrapper for an Android MediaPlayer object. MediaPlayer has its own internal thread for actually doing the work of playing an MP3, so there is no need to spawn a new thread to contain it. There is also no need to asynchronously load an MP3 in a separate thread. It is worth adding the MP3Player class directly to an Activity and controlling it directly with buttons to make sure you understand how it works before moving it into a Service.

The MP3Player has a *state* variable that reflects the stateful nature of the underlying MediaPlayer, and getState() will return one of the following:

```
public enum MP3PlayerState {
    ERROR,
    PLAYING,
    PAUSED,
    STOPPED
}
```

The MP3Player begins in the *STOPPED* state on instantiation.

The class has a few simple methods for loading and playing an MP3:

```
public void load(String filePath)
```

…attempts to load and play the file specified by *filePath*. If all goes well the music will start playing and the MP3Player will now be in the *PLAYING* state. If something went wrong – usually if the file is not found, or is not a playable type, the MP3Player will be in the *ERROR* state.

The music can be paused or unpaused when playing:

```
public void play()

public void pause()
```

Or finally stopped:

```
public void stop()
```

Note that stopping releases and cleans up the MediaPlayer, so the MP3 must be loaded again from the beginning if it needs to start playing again.

The duration and current position of the MP3 can be queried, in milliseconds, and so can the current filename:

```
public int getDuration()
public int getProgress()
public String getFilePath()
```

Files

To transfer mp3 files onto the **sdcard** of the emulator you can drag them from your computer onto the emulator display, and they will be copied to the Downloads folder. From there, move them into the Music folder either using the Files app, or using the command line via **adb shell, cd /sdcard/** to change to the external storage directory**.** Alternatively, from the command line using adb as in lab exercise 3:

adb push my_file.mp3 /sdcard/Music/my_file.mp3

If you don't have access to any music in mp3 format, a variety of royalty free / creative commons licensed files are available here http://freemusicarchive.org/

The Android permissions system by default prevents the application from reading from the sdcard, so you will need to add the READ_EXTERNAL_STORAGE permission to the manifest as shown below. Depending on the emulator API version you are using you will also need to enable this permission from within the phone settings via Settings->Apps->**My MP3Player**->Permissions->Storage

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
```

The code below handles much of the work of listing files in the /Music/ directory and populating a *ListView* with the resultant list. onItemClick is called when one of the entries in

the list is selected by the user. You are free to extend this code as you wish. Note that there is no requirement to recurse into directories or handle directories in any way.

```
import android.os.Environment;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import java.io.File;

final ListView lv = (ListView) findViewById(R.id.listView);

File musicDir = new File(
        Environment.getExternalStorageDirectory().getPath()+ "/Music/");
File list[] = musicDir.listFiles();

lv.setAdapter(new ArrayAdapter<File>(this,
        android.R.layout.simple_list_item_1, list));

lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> myAdapter,
                            View myView,
                            int myItemInt,
                            long mylng) {
        File selectedFromList =(File) (lv.getItemAtPosition(myItemInt));
        Log.d("g53mdp", selectedFromList.getAbsolutePath());
        // do something with selectedFromList...
    }
});
```

Note that ListView can be found in Legacy->ListView in the view design palette. It has largely been superseded by RecyclerView in newer Android API versions, however is used here for simplicity.

https://developer.android.com/guide/topics/ui/layout/recyclerview

Service and Notification

The main element of this coursework is to implement a Service component that contains an instance of the MP3Player class, and to control this Service from your main Activity component.

With the previous exercises in mind, you should think about:

- How should the Service be *started*
- How should the Activity tell the Service what to do (play, pause etc)
- How can the Activity know what the Service is doing (display progress etc)
- When should the Servce be *stopped*

The MP3Player class and the view code above should give you a starting point for developing a Service and an Activity respectively.

Note that you should explicitly design your Service so that it continues running when the Activity component has been *destroyed*, and when the Activity is recreated it should consistently regain control of the Service.

Your Service should maintain a *Notification* while it is running, to allow the user to return to the Activity even after the Activity has been destroyed, via a Pending Intent.

To create a notification, first a proxy for the Notification system service must be retrieved in the form of the NotificationManager. Next, if using a recent version of the API, a channel for the notification is created (this is mostly just boilerplate code).

```
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.os.Build;


private final String CHANNEL_ID = "100";

...

NotificationManager notificationManager = (NotificationManager)
        getSystemService(NOTIFICATION_SERVICE);

// Create the NotificationChannel, but only on API 26+ because
// the NotificationChannel class is new and not in the support library
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    CharSequence name = "channel name";
    String description = "channel description";
    int importance = NotificationManager.IMPORTANCE_DEFAULT;
    NotificationChannel channel = new NotificationChannel(CHANNEL_ID, name,
importance);
    channel.setDescription(description);
    // Register the channel with the system; you can't change the importance
    // or other notification behaviors after this
    notificationManager.createNotificationChannel(channel);
}
```

Each notification your application creates should have a unique ID, and an Intent to start the interface Activity is used to construct a *PendingIntent* that return the user to the Activity when the notification is tapped.

```
import android.support.v4.app.NotificationCompat;
import android.app.PendingIntent;

int NOTIFICATION_ID = 001;

...

intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);

NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this,
CHANNEL_ID)
        .setSmallIcon(R.drawable.ic_launcher_background)
        .setContentTitle("my notification title")
        .setContentText("my notification text")
        .setContentIntent(pendingIntent)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

Finally, the notification is displayed:

```
notificationManager.notify(NOTIFICATION_ID, mBuilder.build());
```

Or used to foreground the service. Note that foreground services require the application to specify the `android.permission.FOREGROUND_SERVICE` permission in the manifest.

```
startForeground(NOTIFICATION_ID, mBuilder.build());
```

References

https://developer.android.com/guide/components/services

https://developer.android.com/guide/topics/ui/notifiers/notifications