Homework 3: Music Player

Submission:

All submissions should be done via git. Refer to the git setup, and submission documents for the correct procedure.

The root directory of your repository should contain your README file, and your Android Studio project directory.

Overview:

Without music to decorate it, time is just a bunch of boring production deadlines or dates by which bills must be paid.

—Frank Zappa

Sometimes you have to play a long time to be able to play like yourself.

—Miles Davis

There's always a dissonance between what you wish was happening and what is actually happening. That's the nature of creativity, that there's a certain level of disappointment in there.

—Jerry Garcia

I remember one time - it might have been a couple times - at the Fillmore East in 1970, I was opening for this sorry-ass cat named Steve Miller. Steve Miller didn't have his shit going for him, so I'm pissed because I got to open for this non-playing motherfucker just because he had one or two sorry-ass records out. So I would come late and he would have to go on first and then we got there we smoked the motherfucking place, everybody dug it.

—Miles Davis

For this assignment, you will be implementing a simple music player.

On the next page are some screenshots of an example app.
Specifications:

Elements

Your app should contain the following elements:

- A list view containing a list of songs.
  - When a song in this list is tapped by a user it should be played.
- Text showing the name of the current song that is playing.
- Text showing the next song that will be played.
  - If a song is selected from the list, the next song will change to the next one on the list.
- A slider that shows the progress through the song.
  - The slider should automatically be updated as the song plays
  - If the user drags the slider then the song should skip to the appropriate position
  - Hint: It can be implemented by SeekBar.
- Text showing how long a song has been playing in the format “MM:SS” (see convertTime)
- Text showing how long until a song finishes in the format “MM:SS”
- A play/pause button
  - Pressing this button plays/pauses the music
  - The button switches between the play and the pause symbol at the appropriate times
- Skip forward and skip backwards buttons.
  - Pressing skip forward causes the next song to be played
  - Pressing the skip backwards button the previous song to be played, even if the current song is well on its way. The previous button never restarts a song.
  - If music is currently paused, skip forward/backward unpause the player
  - If a song is selected from the list, skip forward goes to the next song in the list and skip backward plays previous song of the selected song.
- There should be a settings button in the action bar. When pressed this opens another activity.
  - The settings are for “loop” mode, and there are two buttons, cancel and ok.
  - The values displayed when this activity opens should mirror the current settings of the app
  - The Cancel button
    - Pressing this goes back to the main activity, discarding any modifications
  - The OK button
    - Pressing this goes back to the main activity and uses the provided modifications
    - If loop mode is enabled, press forward/backward button to play the next/previous song and then loop on that song.
- If a song is playing and the activity is paused/settings activity is invoked, you can either keep playing the song or pause the song and resume playing when the main activity comes back.
- You will need controls for your app. We recommend Android’s built in vector graphics, which are very versatile, look good at any resolution and are easy to add to your
project. You can see the course video on icons and images for more information. Once you add them to your project, they will have names like @drawable/ic_play_arrow_black_24dp and @drawable/ic_fast_forward_black_24dp.

- The music in the list will be downloaded from the internet the first time you play it. songs.xml in values directory contains two string arrays. The song_names array contains the name of the song and the urls array contains the corresponding URL. The downloaded music will be stored in app’s internal directory. When the song is played for the second and subsequent times, the player will use the local copy. If your network is bad or non-existent when you first download a song, you might experience a freeze.

**Initialization**

When the app initially starts it should automatically start playing the first song in the list.

**When a song is finished**

When a song finishes playing the next song in the list should be played (unless skip or loop is enabled). If the last song in the list is played then wrap around to the beginning and play the first song.

**Coroutines**

We haven’t talked about coroutines much in class, but they were present in your first two programming assignments, and they are present here. Coroutines are light-weight threads that are independent and can execute concurrently (at the same time). Coroutines are useful for when you need to do something like update the time display. Having code that just wakes up, sees what time it is and update the display is simple to write and to reason about.

There is another coroutine to download the song before you play it. Accessing a device like the network or persistent storage (via the file system) takes a lot longer than computing, so we create a new coroutine to “talk” to the device.

The problem with coroutines is when more than one of them accesses the same memory and at least one of those accesses is a write (e.g., updating or setting a value). When that happens, all sorts of things can go wrong. We’ll talk about what in class, but suffice to say if you have a variable, like a Boolean flag, and you read that flag in one thread and you write it in another, then you need an AtomicBoolean type for that variable. Read the documentation for AtomicBoolean.

We have included most of the code to launch and manage the coroutines in the starter repository, but you need to understand what is going on and you will need an AtomicBoolean.

**Hints**

If you would like you may use your own songs instead of the provided ones. Replace the name and the urls to your song. You can find public domain music on Internet Archive.

The MediaPlayer object has internal state and methods that can be called only in a specific state (https://developer.android.com/reference/android/media/MediaPlayer#state-diagram and https://developer.android.com/reference/android/media/MediaPlayer#valid-and-invalid-states
have more details). You can use an AtomicBoolean to keep track of when your MediaPlayer instance is in the state where it is valid to call a method. Using an atomic type prevents conflicts among threads that concurrently try to modify the variable. Notice that you only need this variable if you want to call a method while the MediaPlayer could be in invalid state. In my solution, I only use one AtomicBoolean.

Work on one feature at a time. There are a lot of tiny elements in this project, many of them do not depend on each other.

I will try to break your app by pressing buttons in orders you did not intend. Prepare for this.

This project will require a little bit of outside research. We do not cover SeekBars (for example) in class, nor do we cover MediaPlayers. But we cover the things you need to know to figure these things out on your own. Google, the Android documentation, Stack Overflow, and Piazza are your friends.

Modify the README we give you. It should be in plain text and named README (not README.txt). It should be in the root directory of your submitted files.