Media Player and Preferences

Prelab:

- Import MP3 music files onto your Android device to be used in this lab **Mac Users:** Make sure to download the Android File Transfer Application
- Bring headphones to this lab in order to test your application without distracting others
- Start thinking about what preferences you'd like to implement in your Music Player

Lab Objectives:

- Find audio files on the device using an Android Content URI
- Create a basic functioning media player to play audio files
- Use a ListView to allow the user to choose which audio file to play
- Use Preferences to implement basic app preferences relevant to a media player

What to Turn in:

- The lab evaluation form checked off and signed by your TA

Importing the Lab Template Project:

- Open Android Studio and import the Android Application project given for this lab.

Discovering Audio Content on an Android Device:

On the Android Platform audio files can be obtained either by including them in your application's resources, retrieving them from a web request or by querying a content provider that hosts audio files. The approach we'll be taking for this lab is the later and the content provider we'll be querying is **`MediaStore.Audio.Media`** which hosts the user's as well as the system's audio files. Querying a content provider is much the same as querying an SQL database; both even return the results of a query via a Cursor object. The code snippet below demonstrates how such a query to the **`MediaStore.Audio.Media`** content provider is formed using its external (user) content URI and the context's content resolver.

```
// Queries the External storage audio files and returns results
Cursor mCursor = getContentResolver().query(
     MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,     // The content URI of Audio
files
      mProjection,              // The list of columns to return for each row
      mSelectionClause,                    // Selection criteria
      mSelectionArgs, // Arguments of the Selection criteria mSortOrder);; // The
      sort order for the returned rows
```

See the Android Development API on `ContentResolver.query` method for more information.

Note: **`MediaStore.Audio.Media`** *has both an internal content URI and an external content URI. If you have imported MP3 music files onto your device then use the external URI to retrieve and play*

those audio files. If you don't have any imported music you can use the internal URI and play ringtone and notification sounds that are already on the device.

Populating `songsList` with Audio File Information:

Next you will use this information about Content URIs to create a method that populates **`songsList`** in **MyMediaPlayerActivity.java** with audio information from the device's **`MediaStore`**. Perform the following steps in the **`populateSongsList()`** method.

1. Create the arguments for the ContentResolver query:

    a. URI: use **`MediaStore.Audio.Media.EXTERNAL_CONTENT_URI`** if you have imported music, otherwise use **`MediaStore.Audio.Media.INTERNAL_CONTENT_URI`**.

    b. Create a String[] of the columns to select, at a minimum you will need **`MediaStore.Audio.Media.TITLE`** (audio file title), and **`MediaStore.Audio.Media.DATA`** (path to audio file)

    c. Choose the query selection arguments (similar to the "WHERE" clause in an SQL query except you exclude the word "WHERE"). At this phase in the lab you can ignore this argument by passing null which will cause all the audio files for the given Content URI to be returned. Later you could filter the audio files based on the value of a column such as the **`IS_MUSIC`** or **`IS_RINGTONE`** column.

```
// Example Query to get only the music files.
mSelectionClause = MediaStore.Audio.Media.IS_MUSIC + " = 1";
```

    d. Add the selection criteria arguments. You may include "?"s in selection, which will be replaced by the values from selectionArgs, in the order that they appear in the selection. The values will be bound as Strings.

    e. Add the sort order arguments. This should be formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing **`null`** will use the default sort order, which may be unordered.

2. Use the cursor returned by the **`ContentResolver.query(...)`** function to iterate through the returned results and populate the ArrayList given in the **MyMediaPlayerActivity.java** class called **`songsList`**. Create a **`SongObject`** representing each audio file and add the title of the song and the file path for the song. Feel free to add more variables to the **`SongObject`** class if you wish to store more information about the audio file.

3. There is already a static method in **MyMediaPlayerActivity.java** that allows **SongList.java** to get the `songsList` and display it in a ListView using the **SongListAdapter** class similarly to the way it was done in the JSON Twitter lab. The template app provided also already has an options menu with the entries "Choose A Song" and "Preferences". We will implement the Preferences page in a later part of this lab. The only thing you need to do to make SongList activity work is to launch the SongList activity when the options menu has the item with the text "Choose a Song" chosen. In the `onOptionsItemSelected(MenuItem item)` method given in the template, create a new intent and launch the **SongList** activity.

4. Demo the **SongList** activity appearing correctly and populated with a list of audio files to your TA.

Using android.media.MediaPlayer to Control Audio Playback:

The basic UI design for the media player has been provided to you for this lab. The template also gives you a method called `playSong(int songIndex)` that will play an audio file that is stored at the songIndex using the **MediaPlayer** class. Your job is to finish implementing the Java code that will be used as the Model and the Controller in this MVC application.

1. Use the MediaPlayer object in **MyMediaPlayerActivity.java** to implement the media player controls `play, pause, forward` and `backward`. Also read the documentation on `android.media.MediaPlayer` to understand how to use it. The methods you will most likely need to have in your final application are: `stop()`, `pause()`, `isPlaying()`, `start()`, `setDataSource(...)`, `prepare()`, and `setOnCompletionListener(...)`.

   http://developer.android.com/reference/android/media/MediaPlayer.html

2. Notice that the Play button is also to be used as the pause button. When the play button is pressed and the audio starts to play, the background source of the play/pause button should be changed to the `btn_pause` selector which is implemented for you in **res/drawable/btn_pause.xml**. Conversely, when the pause button is pressed change the button background back to the `btn_play` selector implemented for you in **res/drawable/btn_play.xml**.

   Note: *Selectors have been used so that when the button is pressed it will change the background image to have the appearance that it has been pressed.*

3. Create button listeners for the Play/Pause, Forward, and Back buttons that are in the layout **media_player_main.xml**. In your button listeners you should implement the basic functions of the media player to be able to play, pause, go forward a track, and go back a track. You may accomplish this task however you please. The `playSong(...)` method should be used to play a specific audio file. Remember to implement the Play and Pause feature using the same button as explained in step 2.

4. Next we need to handle what happens when the MediaPlayer finishes playing an audio file. Implement an **`OnCompletionListener`** that will play the next audio file when the media player finishes playing a file.

Using the ListView to Choose a Song:

At this point the **SongList** activity should be working so the user can view songs that have been obtained from the **`MediaStore`** Content URI in a ListView. The next step is to allow the user to select a song in the ListView and then return to the main media player view and play that song.

1. Make sure that when you launched the SongList activity that you did so using the method **`startActivityForResult(...)`**, and gave the request code a unique integer value. A request code will be associated with the result data when it is returned. The activity you are "starting for result" can not modify this value, allowing you to identify incoming results. Common practice for request codes is to declare a static final variable that corresponds to the request code for the activity you are launching.

2. Next you need to override the method in the Activity called **`onActivityResult(...)`**. This method will be executed when any activity that you "started for result" is finished and the application returns to this activity. The bare minimum implementation of **`onActivityResult(...)`** is given below.

```
@Override
protected void onActivityResult(int requestCode,int resultCode, Intent data) {
      super.onActivityResult(requestCode, resultCode, data);

      // See if the requestCode matches the SongList requestCode you used
      // Then play the song that is returned by the SongList activity
}
```

3. Look at the code in **SongList.java** that handles a list item selection. The key line is this line of code when the result is set:

```
setResult(RESULT_OK,  intent);;
```

4. This line sets the **`resultCode`** and the **`data`** that are going to be the arguments of **`onActivityResult()`**. Use this information and the method **`data.getExtras().getIntExtra(...)`** in order to get the index of the song that is to be played in **`onActivityResult()`**.

5. Make the media player play the song that the user chooses in the **SongList** activity.

6. You should now have both audio files to play and have a working media player to play them with. Please be courteous to others and keep your device at a reasonable volume. Many students will be playing audio files during this lab. Demo your working media player buttons and SongList activity to your TA.

Creating and Using User Preferences:

1. Navigate to File > New > Android XML File (might be under Other…)

2. Change the Resource Type to Preference, call the file **media_preferences.xml**, select PreferenceScreen from the list and click finish.

3. To get you started, add this sample **`CheckboxPreference`** to the **media_preferences.xml** file.

```
<?xml version="1.0" encoding="utf--8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >

        <PreferenceCategory android:title="Song Order Preferences" >
                <CheckBoxPreference
                        android:defaultValue="false"
                        android:key="@string/mp_shuffle_pref"
                        android:summary="play songs in random order"
                        android:title="Shuffle" />
        </PreferenceCategory>
</PreferenceScreen>
```

Note: *The key for each preference should be unique. This will be used by your application to store this preference and to allow you to be able to recall it later.*

In order to figure out whether this setting is currently set to true or false use the following code snippet:

```
// Setup preferences and resources
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
Resources res = getResources();

boolean shuffle;
shuffle = prefs.getBoolean(res.getString(R.string.mp_shuffle_pref), false);
```

4. Create a new Activity that extends **PreferenceActivity** and call it **MediaPreferences**. Also, dont forget to add this activity to the **AndroidManifest.xml** file if you didn't use the New Android Activity Wizard.

5. Inside **MediaPreferences.java** add this **PreferenceFragment** class below. This will set the default values of the preferences if they have not been set already and adds the preferences from **media_preferences.xml**.

```
/** This fragment shows the preferences for the media player */
public static class MediaPreferencesFragment extends PreferenceFragment {
      @Override
      public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);;

            // Make sure default values are applied.
            PreferenceManager.setDefaultValues(getActivity(),
                        R.xml.media_preferences, false);;

            // Load the preferences from an XML resource
           addPreferencesFromResource(R.xml.media_preferences);;
      }
}
```

6. In the **`onCreate()`** method of **MediaPreferences.java** set this PreferenceFragment as the main content using the below code snippet:

```
// Display the fragment as the main content.
getFragmentManager().beginTransaction()
            .replace(android.R.id.content, new MediaPreferencesFragment())
            .commit();
```

Note: *This style of using fragments only works and API 11+ which is Honeycomb and Ice Cream Sandwich. Fragments are meant to simplify design and allow larger screens to have multiple fragments displayed at the same time.*

7. Implement the second part of the **`onOptionsItemSelected(...)`** method in **MyMediaPlayerActivity.java** to launch the MediaPreferences activity.

8. Create and implement at least 3 distinct preferences in your media player. You must use at least 2 different types of preferences (example: **`CheckBoxPreference`**, **`EditTextPreference`**, or **`ListPreference`**. It can be whatever you want). What these preferences are and do is up to you. You may use the shuffle preference given above as 1 of your 3 preferences. All 3 of your preferences should be accessible from the "Preferences" menu item and change the functionality or design of the media player in a significant way.

   Be creative. What are some preferences that you think should be in a media player?

   Check out the documentation on **`PreferenceFragments`**:
   http://developer.android.com/reference/android/preference/PreferenceFragment.html

   Hint: *One thing you could do is to have a preference for the types of audio media you retrieve and play from the Content URI (for example: only retrieve music, and not notifications or ringtones). Do this by changing the Selection Clause of your query to the ContentResolver.*

Note: *There's more to shuffling than just randomly traversing forward to a song. It's expected behavior for a shuffling implementation to be able to keep track of the songs it has selected. This enables the user to re--navigate to previously selected songs when the back button is pressed.*

9. Demo your application to your TA with your three newly added preferences and have them fill out the lab evaluation form.

Lab Evaluation Form --- Media Player and Preferences

Student's Name(s):_____

Lab Section/Time:_____

Evaluation:

_____ SongList activity showing list of audio files on device (15 points)

_____ Basic media player functionality (play / pause / skip / back) (15 points)

_____ Three custom preferences that persist and change media player functionality (20 points)

Turn in:

- Evaluation form signed by TA

TA Signature:_____

Date:_____