COMP4521 EMBEDDED SYSTEMS SOFTWARE

LAB WEEK # 7: USING AUDIO FUNCTIONS IN ANDROID

INTRODUCTION

In this lab we will learn audio related functions of Android. We will construct a simple audio player using the APIs offered by Android Thereafter that we will learn how to start a background service that will keep on playing music even if the main activity is closed. Finally we will see an example of using SoundPool.

OBJECTIVES

- Learn to use audio related APIs.
- Learn to use background service.
- Learn to use SoundPool.

A SIMPLE AUDIO PLAYER

1. Locate the SDK of Android for Eclipse (if necessary).
2. Create an Android Virtual Device (AVD) (if necessary). Make sure to turn on Audio Playback support in the AVD. Edit the AVD if necessary.
3. Create a new project. The project name is **BasicMusicPlayer**, target version is 15, package name is **hkust.comp4521.audio**.
4. Import necessary files. Please put the .png files into **/res/drawable**. They are icon files for the application. And put the .m4a file under a new folder **/res/raw**. This audio file will be played in the application.
5. Add necessary strings. 6 strings are used in our application. Most of them serve as prompting information.

String Name	playername	app_name	isPlaying	stopped	paused	reset
Content	Basic Music Player	BasicMusicPlayer	Song is Playing ...	Song is stopped ...	Song is paused ...	Player is reset ...

6. Create the main layout. It's the only layout of this application. Notice the usage of a new type of widget: ImageButton.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
     android:orientation="vertical" android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:gravity="center">
     <TextView android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:gravity="center"
          android:textSize="14pt"
          android:layout_gravity="center"
          android:text="@string/playername" />
    <LinearLayout
          android:orientation="horizontal"
               android:gravity="center"
          android:layout_height="wrap_content"
          android:layout_width="fill_parent"
          >
       <ImageButton android:id="@+id/play"
                  android:layout_height="wrap_content"
                  android:layout_width="wrap_content"
                  android:src="@drawable/play"/>
       <ImageButton android:id="@+id/reset"
                  android:layout_height="wrap_content"
                  android:layout_width="wrap_content"
                  android:src="@drawable/reset"/>
       <ImageButton android:id="@+id/stop"
                  android:layout_height="wrap_content"
                  android:layout_width="wrap_content"
                  android:src="@drawable/stop"/>
     </LinearLayout>
</LinearLayout>
```

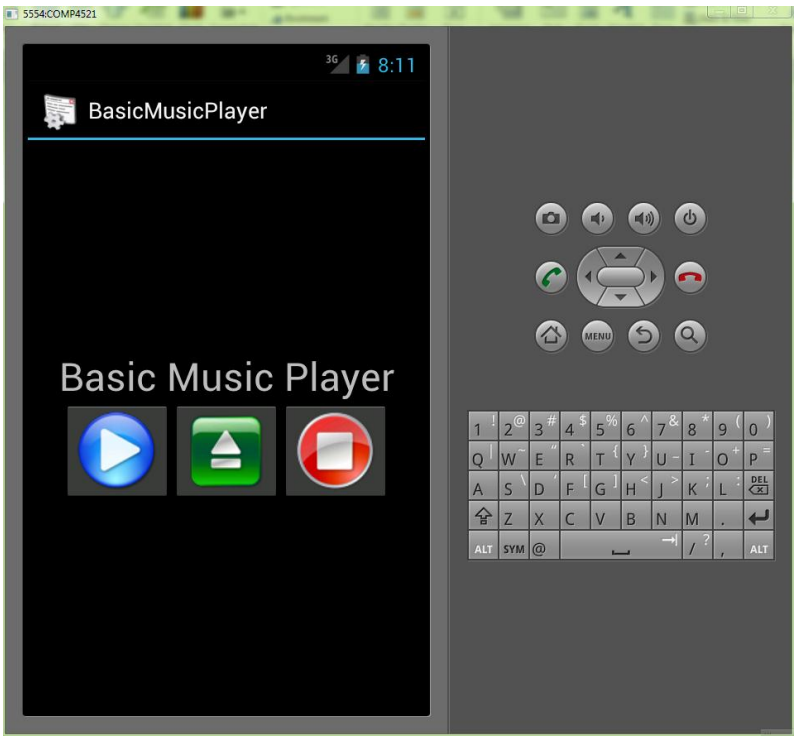
7. Create source code of the main layout. All functions of the main layout are triggered by the buttons, so the structure is relatively simple. Read the comments carefully to understand how to use the rich multimedia APIs offered by Android. Especially, notice how the application updates the main layout dynamically according to the current state.

```
public class BasicMusicPlayer extends Activity implements OnClickListener {

     MediaPlayer player;
     ImageButton playerButton, stopButton, resetButton;
     boolean play_reset = true;

     /** Called when the activity is first created. */
     @Override
     public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.main);

          // Get the button from the view
          playerButton = (ImageButton) this.findViewById(R.id.play);
          playerButton.setOnClickListener(this);

          stopButton = (ImageButton) this.findViewById(R.id.stop);
          stopButton.setOnClickListener(this);

          resetButton = (ImageButton) this.findViewById(R.id.reset);
```

```
        resetButton.setOnClickListener(this);


    }

    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.play:
            // If the user presses the play/pause button
            // if no player instance is available, then create a media player first
            if (play_reset) {
                play_reset = false;
                player = MediaPlayer.create(this, R.raw.braincandy);
                player.setLooping(false); // Set looping
            }
            playPause();
            break;
        case R.id.stop:
            if (!play_reset){
                // If the user presses the stop button
                player.stop();
                // change the image of the play button to play
                playerButton.setImageResource(R.drawable.play);
                Toast.makeText(this, R.string.stopped, Toast.LENGTH_SHORT).show();
                try {
                    player.prepare();
                } catch (IllegalStateException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
            break;
        case R.id.reset:
            if (!play_reset){
                // If the user presses the reset button
                player.reset();
                // change the image of the play button to play
                playerButton.setImageResource(R.drawable.play);
                Toast.makeText(this, R.string.reset, Toast.LENGTH_SHORT).show();

                // Release media instance to system
                player.release();
                play_reset = true;
                break;
            }
        }

    }

    // when the activity is paused, then pause the music playback
    @Override
    public void onPause() {
        super.onPause();

        player.reset();
        // change the image of the play button to play
        playerButton.setImageResource(R.drawable.play);
        Toast.makeText(this, R.string.reset, Toast.LENGTH_SHORT).show();

        // Release media instance to system
        player.release();
        play_reset = true;
    }

    // Toggle between the play and pause
    private void playPause() {
```

```
            // if the music is playing then pause the music playback
            if(player.isPlaying()) {
                  player.pause();
                  // change the image of the play button to play
                  playerButton.setImageResource(R.drawable.play);
                  Toast.makeText(this, R.string.paused, Toast.LENGTH_SHORT).show();
            }
            // Music is paused, start, or resume playback
            else {
                  // change the image of the play button to pause
                  playerButton.setImageResource(R.drawable.pause);
                  player.start();
                  Toast.makeText(this, R.string.isPlaying, Toast.LENGTH_SHORT).show();
            }
      }
}
```

8. Run the application. Test what will happen when you press each of the button, and go back to the source code to see how are these functions defined.



9. *A question left for you:* Can you modify the source code so that it can play any audio files on the SD card of the device? Can you modify it to enable you to play streaming audio?

AN AUDIO PLAYER RUNNING IN THE BACKGROUND

1. In practice, users often expect the audio player to keep on playing even if the main layout is minimized. This can be implemented by running a background service on the device.
2. Modify the source code. We do not need extra data files or strings. When clicking on the buttons, the main thread of application will not do any real operation; instead, it will pass intents to the background service to trigger related actions.

```
    private static final String TAG = "MyPlayer";
    private static ImageButton playerButton, stopButton, resetButton;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
    Log.d(TAG, "after layout");

    // Get the button from the view
    playerButton = (ImageButton) findViewById(R.id.play);
    playerButton.setOnClickListener(this);

    stopButton = (ImageButton) findViewById(R.id.stop);
    stopButton.setOnClickListener(this);

    resetButton = (ImageButton) findViewById(R.id.reset);
    resetButton.setOnClickListener(this);

}

public void onClick(View v) {
  switch (v.getId()) {
  case R.id.play:
    startService(new Intent(this, MusicService.class));
        playerButton.setImageResource(R.drawable.pause);
    break;
  case R.id.stop:
    stopService(new Intent(this, MusicService.class));
        playerButton.setImageResource(R.drawable.play);
    break;
  case R.id.reset:
    stopService(new Intent(this, MusicService.class));
        playerButton.setImageResource(R.drawable.play);
      break;
  default:
    break;
  }
}
```

3. Create the background service called MusicService.java. Its superclass is android.app.Service.

```
public class MusicService extends Service {
    private static final String TAG = "MyService";
    MediaPlayer player;

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
```

```
@Override
public void onCreate() {
      Toast.makeText(this, "Music Service Created", Toast.LENGTH_SHORT).show();
      Log.d(TAG, "onCreate");

      player = MediaPlayer.create(this, R.raw.braincandy);
      player.setLooping(false); // Set looping
}

@Override
public void onDestroy() {
      Toast.makeText(this, "My Service Stopped", Toast.LENGTH_SHORT).show();
      Log.d(TAG, "onDestroy");
      player.stop();
      player.reset();
      // Release media instance to system
      player.release();
}

@Override
public void onStart(Intent intent, int startid) {
      Toast.makeText(this, "My Service Started", Toast.LENGTH_SHORT).show();
      Log.d(TAG, "onStart");
      player.start();
}
   }
```

4. Add a declaration for the service in <application> of **AndroidManifest.xml**. This is necessary for any newly created background services.

```
<service android:enabled="true" android:name="MusicService" />
```

5. Run the application. Now the player will keep on playing music even if the main activity is minimized, because the background service, which is responsible for playing music, is still running.
6. *A similar question left for you:* Can you modify the source code so that it can play any audio files on the device?

USING SOUNDPOOL

1. Create a new project. The project name is **SoundPoolTutorial**, the package name is **hkust.comp4521.audio**.
2. Add resource audio files. Put the 4 audio files to **/res/raw**.
3. Modify the main layout. We only need to add two buttons in the main.xml. Their ids are Button01 and Button02, and texts are “Play Music” and “Shoot” respectively.
4. Create source code for the buttons. SoundPool is used here to mix the source audio files. Android offers rich functions that can manipulate the output sound stream.

```
public class Tutorial extends Activity implements OnClickListener {
     private  SoundPool mSoundPool;
     private  HashMap<Integer, Integer> mSoundPoolMap;
     private  AudioManager  mAudioManager;
     private  Context mContext;
     int streamVolume;

     /** Called when the activity is first created. */
     @Override
     public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.main);

          mContext = getBaseContext();
          mSoundPool = new SoundPool(6, AudioManager.STREAM_MUSIC, 0);
          mSoundPoolMap = new HashMap<Integer, Integer>();
          mAudioManager =
(AudioManager)mContext.getSystemService(Context.AUDIO_SERVICE);

          mSoundPoolMap.put(1, mSoundPool.load(mContext, R.raw.tranquil1, 1));
          mSoundPoolMap.put(2, mSoundPool.load(mContext, R.raw.riff, 1));
          mSoundPoolMap.put(3, mSoundPool.load(mContext, R.raw.drums, 1));
          mSoundPoolMap.put(4, mSoundPool.load(mContext, R.raw.explosion, 1));
          streamVolume = mAudioManager.getStreamVolume(AudioManager.STREAM_MUSIC);

          Button SoundButton = (Button)findViewById(R.id.Button1);
          SoundButton.setOnClickListener(this);

          Button GunButton = (Button)findViewById(R.id.Button2);
          GunButton.setOnClickListener(this);


     }

     public void onClick(View v) {
          switch (v.getId()) {
          case R.id.Button1:
               // set up for these two streams to loop for ever
               mSoundPool.play(mSoundPoolMap.get(1), streamVolume, streamVolume, 1, -1,
1f);
               mSoundPool.play(mSoundPoolMap.get(3), streamVolume, streamVolume, 1, -1,
1f);
               break;
          case R.id.Button2:
               // makes the explosion sound whenever the shoot button is pressed
               mSoundPool.play(mSoundPoolMap.get(4), streamVolume, streamVolume, 1, 0,
1f);
               break;
          }
```

```
    }
}
```

5. Run the application. Click the buttons to see what will happen, and go back to the source code to understand how the functions are implemented.