WIA2007 Mobile Application Development
Semester 1, Session 2022/2023
Practical 12 (Sound and Audio)

Today, we will develop a simple one-activity mobile application to play music stored in the raw resource directory and observe the status of the playback.

In Android Studio, create a new Android Studio Project with an Empty Activity (MainActivity.java and activity_main.xml). The **MainActivity** will :

- Creates UI that enables the user to play, pause and reset media playback
- Creates a MediaPlayerHolder object.
- Implement the PlaybackInfoListener to allow the MediaPlayerHolder object to update media duration and progress changes.

Apart from the Main Activity, you also need to create the following three Java classes (without the layout XML file):

- **PlayerAdapter** – interface class
- **MediaPlayerHolder** – implement PlayerAdapter to allow MainActivity to control playback functions.
- **PlaybackInfoListener** – abstract class

Task 1: Resource Preparation

We will play the music from the raw resource directory, hence, you need to choose any .mp3 file that you like and store it inside the res/raw resource directory.

Besides, we need 4 icons for “play”, “pause”, “reset”, and “seek bar”. These icons will be stored in the drawable folder. *Remember how we create a vector asset?*

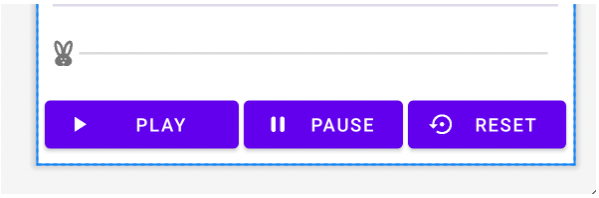

Figure 1: The four icons.

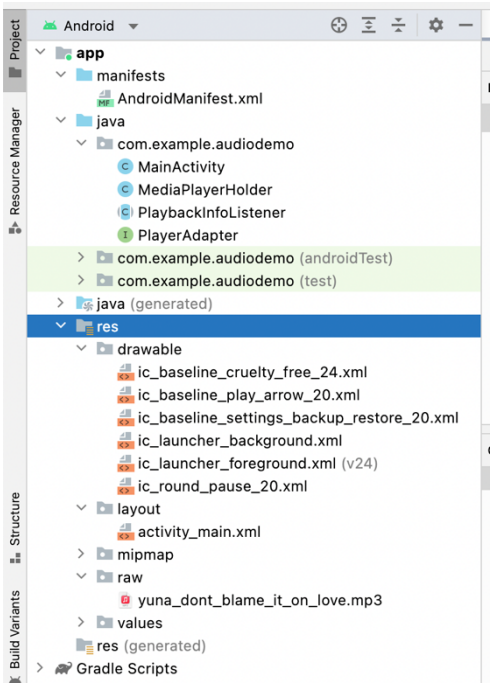

Figure 2: The project directory.

After you have added the vector assets (viz., the four icons) and the music (viz., .mp3 file), your project directory should look like Figure 2 above.

Task 2: The Layout XML File

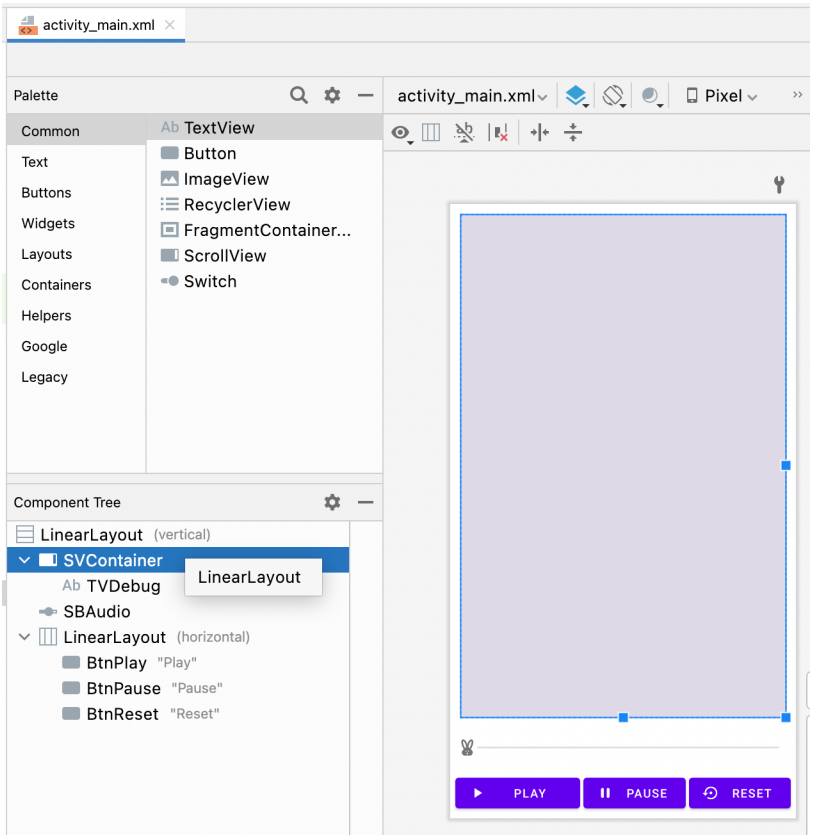

Figure 3: The layout XML file for MainActivity.

Use Linear Layout (or any layout of your choice), prepare a:

- ScrollView (ID: **SVContainer**) that contains a TextView (ID: **TVDebug**) (to display the state of the player and allow it to be scrollable).
- Seekbar (ID: **SBAudio**) (to display current play time and set playtime).
- Nested Linear Layout which contains THREE buttons for Play (ID: **BtnPlay**), Pause (ID: **BtnPause**), and Reset (**BtnReset**).

Code 1: The sample codes for activity_main.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="4dp"
    tools:context=".MainActivity">

    <ScrollView
        android:id="@+id/SVContainer"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="8dp"
        android:layout_weight="1"
        android:background="#DFD9E8"
        android:padding="4dp" >

        <TextView
            android:id="@+id/TVDebug"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="8dp"
            android:background="#DFD9E8"
            android:padding="4dp"
            android:textColor="#3F51B5"
            android:textSize="12sp"
            android:typeface="monospace" />
    </ScrollView>

    <SeekBar
        android:id="@+id/SBAudio"
        style="@style/Widget.AppCompat.SeekBar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginBottom="16dp"
        android:thumb="@drawable/ic_baseline_cruelty_free_24" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:orientation="horizontal">

        <Button
            android:id="@+id/BtnPlay"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
```

The ScrollView and the TextView to display the status of the playback.

The SeekBar that has a thumb icon that will move to indicate the progress of the playback.

The LinearLayout (with horizontal orientation) that contains Play, Pause, and Reset buttons.

```
            android:layout_weight="4"
            android:text="Play"
            app:icon="@drawable/ic_baseline_play_arrow_20" />

        <Button
            android:id="@+id/BtnPause"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
            android:layout_weight="1"
            android:text="Pause"
            app:icon="@drawable/ic_round_pause_20" />

        <Button
            android:id="@+id/BtnReset"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
            android:layout_weight="1"
            android:text="Reset"
            app:icon="@drawable/ic_baseline_settings_backup_restore_20"
/>
    </LinearLayout>
</LinearLayout>
```

Task 3: MediaPlayer

In this example, we will prepare the codes for the three classes (PlayerAdapter.java, PlaybackInfoListener.java, and MediaPlayerHolder.

First, let's prepare the PlayerAdapter interface as shown in Code 2. This PlayerAdapter will act as the abstraction/bridge of communication between the client (MainActivity) and the backend (MediaPlayerHolder).

Code 2: The codes for PlayerAdapter.java.

```
package com.example.audiodemo;

public interface PlayerAdapter {
    void loadMedia(int resourceId);
    void release();
    boolean isPlaying();
    void play();
    void reset();
    void pause();
    void initializeProgressCallback();
    void seekTo(int position);
}
```

Next, we will prepare the Abstract class PlaybackInfoListener specifically for our media player to list out the possible states and changes (methods) during the playback.

Code 3: The codes for PlaybackInfoListener.java.

```
package com.example.audiodemo;

import androidx.annotation.IntDef;
```

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

//abstract PlaybackInfoListener to list out the possible states and
changes (methods) during the playback
public abstract class PlaybackInfoListener {

    //Retention indicates how long annotations with the annotated type
are to be retained
    //Set RetentionPolicy.SOURCE - Annotations are to be discarded by the
compiler.
    @Retention(RetentionPolicy.SOURCE)
    @IntDef({State.INVALID, State.PLAYING, State.PAUSED, State.RESET,
State.COMPLETED})
    //@interface == annotation
    @interface State{
        int INVALID = -1;
        int PLAYING = 0;
        int PAUSED = 1;
        int RESET = 2;
        int COMPLETED = 3;
    }

    //Returns the string of the annotated state
    public static String convertStateToString(@State int state){
        String stateString;
        switch(state){
            case State.INVALID:
                stateString = "INVALID";
                break;
            case State.PLAYING:
                stateString = "PLAYING";
                break;
            case State.PAUSED:
                stateString = "PAUSED";
                break;
            case State.RESET:
                stateString = "RESET";
                break;
            case State.COMPLETED:
                stateString = "COMPLETED";
                break;
            default:
                stateString = "UNKNOWN STATE";
        }
        return stateString;
    }

    //to be implemented later
    void onLogUpdated(String formattedMessage){}
    void onDurationChanged(int duration){}
    void onPositionChanged(int position){}
    void onStateChanged(@State int state){}
    void onPlaybackCompleted(){}
}
```

Define the state of the player.
https://developer.android.com/reference/androidx/annotation/IntDef
https://developer.android.com/reference/java/lang/annotation/Retention

Convert State to String.

Methods that will be implemented in MainActivity class later.

Lastly, the MediaPlayerHolder class (the backend player implementation):

Code 4: The codes for MediaPlayerHolder.java.

```
package com.example.audiodemo;

import android.content.Context;
import android.content.res.AssetFileDescriptor;
import android.media.MediaPlayer;
import android.os.Build;
import androidx.annotation.RequiresApi;
import java.io.IOException;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

//actual implementation of media player
public class MediaPlayerHolder implements PlayerAdapter {

    //the time duration (in MS) to update the seekbar playback position
    public static final int PLAYBACK_POSITION_REFRESH_INTERVAL_MS = 1000;

    //Context - our application environment to access resources and start
process
    private final Context mContext;
    private MediaPlayer mMediaPlayer;
    private int mResourceId;
    private PlaybackInfoListener mPlaybackInfoListener;
    //ScheduledExercuterService schedules command to run on particular
time / interval
    private ScheduledExecutorService mExecutor;
    //Runnable to run the codes on thread
    private Runnable mSeekbarPositionUpdateTask;

    //get the calling environment and roll the tasks!
    public MediaPlayerHolder(Context context){
        mContext = context.getApplicationContext();
    }

    //Should only be called on the given API level or higher
    @RequiresApi(api = Build.VERSION_CODES.N)
    public void loadMedia(int resourceId){
        mResourceId = resourceId;
        initializeMediaPlayer();
        AssetFileDescriptor assetFileDescriptor =
mContext.getResources().openRawResourceFd(mResourceId);
        try{
            logToUI("> Trying to load the mp3. [03:loadMedia()]");
            mMediaPlayer.setDataSource(assetFileDescriptor);
        } catch (IOException e) {
            logToUI(e.toString());
        }

        //Transfer MediaPlayer object to the Prepared State before it can
be Started. Refer to lifecycle of MediaPlayer
        try{
            logToUI("> Prepare the media player. [03:loadMedia()]");
            mMediaPlayer.prepare();
        } catch (IOException e) {
            logToUI(e.toString());
```

Implementing PlayerAdapter.

Declaring variables and creating object.

The constructors and getting the application context (UI).

Loading the media file (pass using resourceId) by using setDataSource method.

After setDataSource(), the MediaPlayer object will be at *Initialized* state, and prepare() can be called. Once prepare() is returned, it will moved to the Prepared state. →(You may refer to the state diagram in Lecture 11).

```
        }


        initializeProgressCallback();
        //logToUI("> Initialize ProgressCallback()");
    }

    //Create Media Player object
    private void initializeMediaPlayer() {
        if(mMediaPlayer == null) {
            mMediaPlayer = new MediaPlayer();
            mMediaPlayer.setOnCompletionListener(new
MediaPlayer.OnCompletionListener() {
                //When the playback reaches the end of the stream
                //Display the completion information and update the
state.
                @Override
                public void onCompletion(MediaPlayer mediaPlayer){
                    stopUpdatingCallbackWithPosition(true);
                    logToUI("> MediaPlayer playback completed");
                    if (mPlaybackInfoListener != null) {

mPlaybackInfoListener.onStateChanged(PlaybackInfoListener.State.COMPLETED
);
                        mPlaybackInfoListener.onPlaybackCompleted();
                    }
                }
            });
            logToUI("> MediaPlayer object created and
onCompletionListener setup ready. " +
                    "[03:initializeMediaPlayer()]");
        }
    }

    private void logToUI(String s) {
        if(mPlaybackInfoListener != null){
            mPlaybackInfoListener.onLogUpdated(s);
        }
    }

    @Override
    public void initializeProgressCallback() {
        //get the duration of the media file
        final int duration = mMediaPlayer.getDuration();
        if(mPlaybackInfoListener != null) {
            mPlaybackInfoListener.onDurationChanged(duration);
            mPlaybackInfoListener.onPositionChanged(0);
            logToUI(String.format("Setting the playback duration (%d
sec). [03:initializeProgressCallback()]",
                    TimeUnit.MILLISECONDS.toSeconds(duration)));
            logToUI("> Setting playback position to 0.
[03:initializeProgressCallback()]");
        }
    }

    public void setPlaybackInfoListener(PlaybackInfoListener listener){
        mPlaybackInfoListener = listener;
    }
```

Create MediaPlayer object.

Action taken when it completed its playback.

Display progress on the UI.

Get the duration and set the playback position.

```
    @Override
    public void release() {
        if(mMediaPlayer != null) {
            logToUI("> Release and set MediaPlayer to null.
[03:release()]");
            mMediaPlayer.release();
            mMediaPlayer = null;
        }
    }

    @Override
    public boolean isPlaying() {
        if(mMediaPlayer != null){
            logToUI("> Media player is playing. [03:isPlaying()]");
            return mMediaPlayer.isPlaying();
        }
        logToUI("> Media player is not playing. [03:isPlaying()]");
        return false;
    }

    @Override
    public void play() {
        if(mMediaPlayer != null && !mMediaPlayer.isPlaying()) {
            logToUI(String.format("> Starting to play my favorite song
- %s. [03:play()]",

mContext.getResources().getResourceEntryName(mResourceId)));
            mMediaPlayer.start();
            if(mPlaybackInfoListener != null){
                logToUI("> Media player is currently playing.
[03:play()]");

mPlaybackInfoListener.onStateChanged(PlaybackInfoListener.State.PLAYING);
            }
            startUpdatingCallbackWithPosition();
        }
    }
    @RequiresApi(api = Build.VERSION_CODES.N)
    @Override
    public void reset() {
        if(mMediaPlayer != null) {
            logToUI("> Resetting media player. [03:reset()]");
            mMediaPlayer.reset();
            loadMedia(mResourceId);
            if(mPlaybackInfoListener != null){

mPlaybackInfoListener.onStateChanged(PlaybackInfoListener.State.RESET);
            }
            stopUpdatingCallbackWithPosition(true);
        }
    }

    private void stopUpdatingCallbackWithPosition(boolean
resetUIPlaybackPosition) {
        logToUI("> Resetting executor.
[03:stopUpdatingCallbackWithPosition()]");
        if(mExecutor != null){
            mExecutor.shutdownNow();
            mExecutor = null;
            mSeekbarPositionUpdateTask = null;
```

Release the player.

Check if the player is playing.

Start playing the music.

Reset the music.

Release the executor and reset SeekBar Position.

```
            if(resetUIPlaybackPosition && mPlaybackInfoListener != null){
                mPlaybackInfoListener.onPositionChanged(0);
            }
        }
    }

    private void startUpdatingCallbackWithPosition() {
        logToUI("> Starting executor.
[03:startUpdatingCallbackWithPosition()]");
        if (mExecutor == null) {
            mExecutor = Executors.newSingleThreadScheduledExecutor();
        }
        if(mSeekbarPositionUpdateTask == null) {
            mSeekbarPositionUpdateTask = new Runnable() {
                @Override
                public void run() {
                    updateProgressCallbackTask();
                }
            };
        }
        mExecutor.scheduleAtFixedRate(
                mSeekbarPositionUpdateTask, 0,
                PLAYBACK_POSITION_REFRESH_INTERVAL_MS,
TimeUnit.MILLISECONDS
        );
    }

    //Updating the progress of media player
    private void updateProgressCallbackTask() {
        if(mMediaPlayer != null && mMediaPlayer.isPlaying()){
            int currentPosition = mMediaPlayer.getCurrentPosition();
            if(mPlaybackInfoListener != null){
                mPlaybackInfoListener.onPositionChanged(currentPosition);
            }
        }
    }

    @Override
    public void pause() {
        if(mMediaPlayer != null && mMediaPlayer.isPlaying()){
            mMediaPlayer.pause();
            if(mPlaybackInfoListener != null){

mPlaybackInfoListener.onStateChanged(PlaybackInfoListener.State.PAUSED);
            }
        }
        logToUI("> Pausing the playback. [03:pause()]");
    }

    @Override
    public void seekTo(int position) {
        if (mMediaPlayer != null){
            logToUI(String.format("> Changing position to %d ms. [03:
seekTo()]", position));
            mMediaPlayer.seekTo(position);
        }
    }
}
```

Start the executor and SeekBar update.

Update the SeekBar position.

Pause the playback.

Go to specific position of the play.

Task 4: The MainActivity class

Now, let's tie the user action to the actual implementation:

Code 5: The codes for MainActivity.java.

```
package com.example.audiodemo;

import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AppCompatActivity;

import android.os.Build;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ScrollView;
import android.widget.SeekBar;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    public static final String TAG = "MainActivity";
    public static final int MEDIA_RES_ID =
R.raw.yuna_dont_blame_it_on_love;

    private TextView mTextDebug;
    private SeekBar mSeekbarAudio;
    private ScrollView mScrollContainer;
    private PlayerAdapter mPlayerAdapter;
    private boolean mUserIsSeeking = false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initializeUI();
        initializeSeekbar();
        initializePlaybackController();
        Log.d(TAG, ">> Completed activity initialization.
[04:onCreate()]");
    }

    private void initializeSeekbar() {
        mSeekbarAudio.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
            int userSelectedPosition = 0;
            @Override
            public void onProgressChanged(SeekBar seekBar, int progress,
boolean fromUser) {
                if(fromUser){
                    userSelectedPosition = progress;
                }
            }

            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {
                mUserIsSeeking = true;
            }

            @Override
```

Setup all the components when the activity started.

Setting listener for any SeekBar changes.

```
            public void onStopTrackingTouch(SeekBar seekBar) {
                mUserIsSeeking = false;
                mPlayerAdapter.seekTo(userSelectedPosition);
            }
        });
    }

    private void initializePlaybackController() {
        MediaPlayerHolder mMediaPlayerHolder = new
MediaPlayerHolder(this);
        Log.d(TAG, ">> Created MediaPlayerHolder
[04:initializePlaybackController()]");
        mMediaPlayerHolder.setPlaybackInfoListener(new
PlaybackListener());
        mPlayerAdapter = mMediaPlayerHolder;
        Log.d(TAG, ">> MediaPlayerHolder progress callback set.
[04:initializePlaybackController()]");
    }

    private void initializeUI() {
        mTextDebug = findViewById(R.id.TVDebug);
        Button mPlayButton = findViewById(R.id.BtnPlay);
        Button mPauseButton = findViewById(R.id.BtnPause);
        Button mResetButton = findViewById(R.id.BtnReset);
        mSeekbarAudio = findViewById(R.id.SBAudio);
        mScrollContainer = findViewById(R.id.SVContainer);

        mPauseButton.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View view) { mPlayerAdapter.pause(); }
        });
        mPlayButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) { mPlayerAdapter.play(); }
        });
        mResetButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) { mPlayerAdapter.reset(); }
        });
    }

    @Override
    protected void onStart() {
        super.onStart();
        mPlayerAdapter.loadMedia(MEDIA_RES_ID);
        Log.d(TAG, ">> Create MediaPlayer. [04:onStart()]");
    }

    @Override
    protected void onStop() {
        super.onStop();
        if(isChangingConfigurations() && mPlayerAdapter.isPlaying()){
            Log.d(TAG, ">> Don't release MediaPlayer as screen is
rotating and playing. [04:onStop()]");
        }
        else {
            mPlayerAdapter.release();
            Log.d(TAG, ">> Release MediaPlayer. [04:onStop()]");
        }
    }
```

Initialize the Playback Controller (including the PlaybackInfoListener and PlayerAdapter).

Get handle of the UI widgets and set onClickListeners for the three buttons.

Load media during the onStart().

Release the resources during onStop().

```
    private class PlaybackListener extends PlaybackInfoListener {
        @Override
        void onDurationChanged(int duration) {
            mSeekbarAudio.setMax(duration);
            Log.d(TAG, String.format(">> Setting playback duration -
setMax(%d). " +
                    "[04sub:onDurationChanged()]", duration));
        }

        @RequiresApi(api = Build.VERSION_CODES.N)
        @Override
        void onPositionChanged(int position) {
            if(!mUserIsSeeking){
                mSeekbarAudio.setProgress(position, true);
                Log.d(TAG, String.format(">> Setting position changes -
setProgress(%d). " +
                        "[04sub:onPositionChanged()]", position));
            }
        }

        @Override
        void onStateChanged(int state) {
            String stateToString =
PlaybackInfoListener.convertStateToString(state);
            onLogUpdated(String.format(">> Changing state - (%s). " +
                    "[04sub:onStateChanged()]", stateToString));
        }

        @Override
        void onPlaybackCompleted() {
            onLogUpdated(">> Playback is completed.
[04sub:onPlaybackCompleted()]");
        }

        @Override
        void onLogUpdated(String formattedMessage) {
            if(mTextDebug != null){
                mTextDebug.append(formattedMessage);
                mTextDebug.append("\n");
                //Moves the scrollContainer focus to the end
                mScrollContainer.post(
                        new Runnable() {
                            @Override
                            public void run() {

mScrollContainer.fullScroll(ScrollView.FOCUS_DOWN);
                            }
                        }

                );
            }
        }
    }
}
```

Create inner class PlaybackListener to track any playback changes.

Display message in log.

Additional information (for reference only): Module Gradle File

Code 6: The sample Module Gradle File.

```
plugins {
    id 'com.android.application'
}

android {
    compileSdk 31

    defaultConfig {
        applicationId "com.example.audiodemo"
        minSdk 21
        targetSdk 31
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner
"androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-
optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}

dependencies {

    implementation 'androidx.appcompat:appcompat:1.4.0'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.2'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-
core:3.4.0'
}
```

Submission

You are required to complete the exercise and submit the Android Studio Project to Spectrum before the next Tutorial class.