Android Development – The First App

Introduction.

Android is a popular operating system based on the Linux platform. The most popular flavors of Android run on x86 and ARM based processors. With integrated motion, position and environment sensors, it is an ideal platform for application development today. The Android SDK is Java based and builds on top of some of the core Java libraries. With over 1 million applications on the Play Store on Android, help and resources on Android Developement is easily available. This is a chronicle of our first Android application. In this guide, we will create a simple workout app targetting users who are interested in training for the military or otherwise. We will incorporate three types of exercise and a full training more comprising of all of the three exercises. Our application will not transmit or capture any data related to the user. It will be simple, readible, and practical in its application. We will use Android phone sensors to add simple features such as step count, and proximity detection. In the end, we will publish this app to the Play Store as a developer.

Prerequisites.

- A Linux, Windows, or MacOS machine with JDK.
- 4 GB RAM or more preferably (Emulators are resource intensive)
- 1 GB or more free disk space.
- Android SDK (Version 17 and up)
- Android Studio (Alternatively, use Eclipse ADT)
- Android Virtual Device or an Android Phone (Rooted + Developer Mode on)

First Steps.

Now that we have downloaded the JDK, Android SDK, and an IDE (Android Studio in our case), we are ready to create our first app. One further optional step would be to install Android SDK Manager to manage the features provided by the Android SDK.

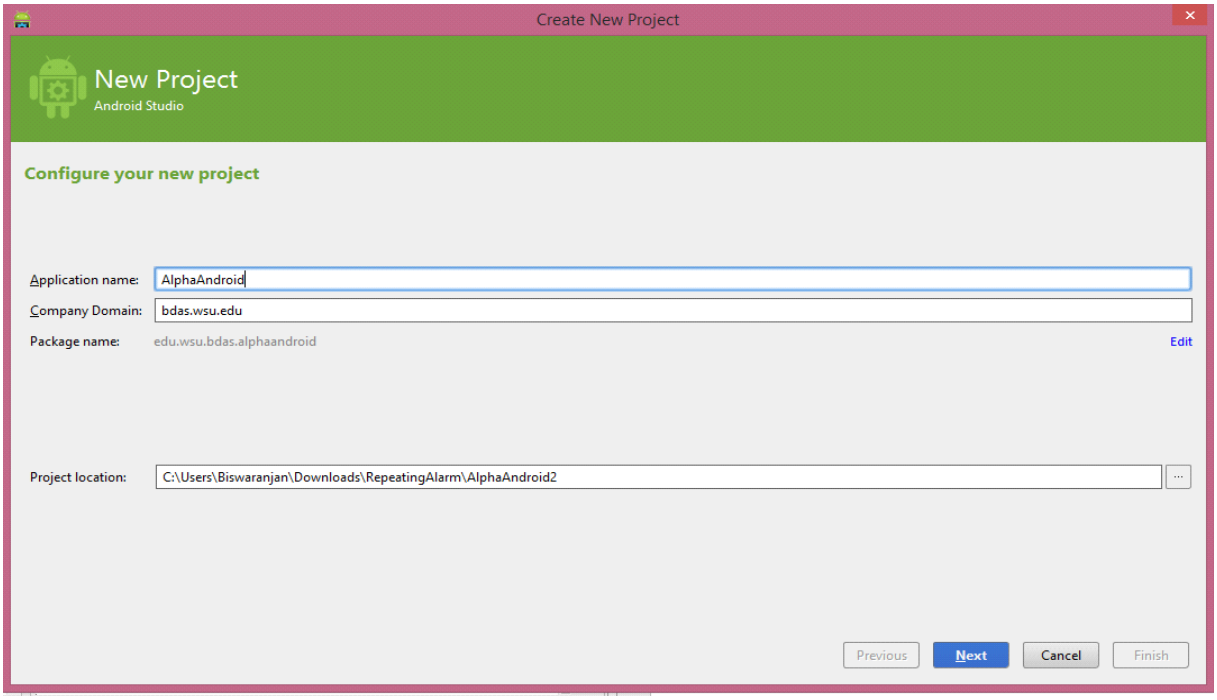
Our application will have 3 types of workout training. We will provide 'Cadence Pushups', 'Situp Training' and a 'Runner's Log'. As an automated option, we will provide a default mode where all the exercises are performed in sequence according to user defined settings. Aside from the graphics, and the multimedia to be used, we will need to use very simple programming mechaisms. The core concepts required to develop an application such as this, we will need to use the following namespaces from JAVA and Android SDK:

- java.util.timer (For timer services, we will need this for logging running times, and situp times.)
- android.os.countdowntime (for countdown timer, we will use this for cadence pushups, and situps)
- android.hardware.* (for sensors management, events and event handlers)
- android.widget.* (for Buttons, TextViews, ImageViews, and other inbuilt UI components)

- android.view.*
- com.google.android.gms.maps.*
- com.google.android.gms.maps.model.* (for Google Maps integration; we will cover it later)

Android Studio.

We will start Android Studio and create a new project **(File → New → Project.)**



We will name our application and give it a namespace and choose the Android SDK we want to target, which is SDK 15+. We will then add an activity to our application. This will be our splash screen. This 'Fullscreen' activity will be displayed while our resources are loading. This will ensure a consistent experience.

A basic android applications lifecycle is managed by the OS itself . An application consists of various “activities”. An activity can run in the background, as a regular application activity or as a fullscreen activity. The capabilities of an application, its accompanying activities, the types of sensors used by the application, etc are all stored in the android_manifest file (XML). The layouts of the activities, the global and local styles and local resources used are stored in XML based files as well. Here is a general directory structure of an Android project:

<ANDROID APP>
- /src
- /res
  - /drawable
  - /menu
  - /values
    - strings.xml
    - arrays.xml
  - /xml
    - preference.xml

SplashScreen Activity

- Create a blank activity with name SplashScreen.
- This will create two files that we will edit.
- activity_splashscreen.xml (/res/layout)
- SplashScreen.java (/src)
- Since this is only our splash screen, we will remove/hide the action bar from the screen.
- Add an ImageView widget to the activity.
- You can do this either in the design view or in the text view.
- If you have a splash image already, use it as a drawble but if not, skip this part for now
- The xml file for this activity should look something like this.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#0099cc"
    tools:context=".SplashScreen">

    <ImageView android:id="@+id/splashImage"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:src="@drawable/splash"
        android:scaleType="centerCrop" />

    <!-- The primary full-screen view. This can be replaced with whatever view
         is needed to present your content, e.g. VideoView, SurfaceView,
         TextureView, etc. -->
    <TextView android:id="@+id/fullscreen_content"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center|bottom"
        android:keepScreenOn="true"
        android:textColor="#FFFFFF"
        android:textSize="40sp"
        android:gravity="fill_horizontal"
        android:text= "@string/dummy_content" />
```

```
    <!-- This FrameLayout insets its children based on system windows using
         android:fitsSystemWindows. -->
    <FrameLayout android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:fitsSystemWindows="true">
    </FrameLayout>

</FrameLayout>
```

- After our data has loaded, we need to go to our main screen.
- Our main screen is just a full page menu.
- It has 2 small icons on the top right for information and exit!
- 5 menu items: Cadence Pushups, Situp Training, Runner's Log, Complete Training and Settings.

Styles and Strings.

We will use some custom styles to make our buttons and background look more appealing. A good reference on style.xml can be found here.

https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/res/res/values/styles.xml

Our style sheet is much simpler with only 1 addition button style and a background style. Note: It is possible to remove action bar by using the xml style sheet. We have however done this programatically. We will have all menu items, option items and setting items as string in the string.xml.

Drawables and Other Resources.

In our application we use about a dozen icons, splash screen image and variety of audio files for output. Since we will need to have custom resources that are original and royalty free, we will need to add these resources to our “/res” directory. MP3 files and other Raw resources go to "/res/raw" folder.

Cadence Pushups.

We will create another activity called “Pushups”. In this activity we provide cadence pushup training to our users (by default or custom settings). The basic requirement to have a pushup counter is to have a TextView widget (for display), a button to quit and a button to pause. In our initial app, we will use the Proximity Sensor inbuilt in some android phones and the SDK to detect if a user has completed a pushup cycle. This mechanism will allow us to count the number of pushups. A pre-workout tone of (ready/set/go) is played before the actual Down/Up Cycle. Each cycle is associated with stored voices to assist the users. At the end of the activity, we play another audio reassuring the user that the activity has indeed finished.

Situp Training.

Our application will start a counter after a defined time and count situps using the same technique that we used in our cadence pushups for counting. Instead of voice training, we let the user pick the time and display the statistics of the session after the sit ups.

Runners' Log.

Our running log book will provide a step counter, a gps mapper and will cast a live visually appealing video stream to the nearest chromecast available. For older phones, we will provide the basic option to just have a timer activated. The user must self-deactivate the timer in case they do not have gps or they are using a treadmill to run. We will store logs from our activities on local machine only to provide users with useful data and statistics.

Settings.

This will provide option to the user regarding the application and the training itself. We will need to save the state of the user options. We will have to do this in a way that the data is secure, persistent and not intrusive. To do this, we will use the Android 'Key-Value Pair' save mechanism.

Java

Now that we have the barebone structure of our application set up and some design tools in place, we can start to write the code. Firstly, we will remove the action bar from the Splash Screen.

Removing Action Bar

```
ActionBar actionBar = getSupportActionBar();
actionBar.hide();
```

We will now add an option so that our activity_splashscreen switches to our activity_homescreen once we load our whole project. To do this we will use the following API calls:

. display Splash screen onPreExecute()
. We will do this Asynchronously.
. We will handle Homescreen loading at onPostExecute()

To associate variables in Java with Android components, we use the standard API call :

```
(TypeOfComponent) findViewById.R.id.idname …
```

Note: Components in Android are called 'View' and a group of 'Views' is called 'ViewGroup'.

Once we have set up the variables to match our 'Views', we will declare an handler for the click events on these 'Views'. On click, we finish our current activity and start the required activity. We will not handle explicit closing of activities since Android manages the lifecycle itself. However, since

we need to save our 'Settings', we will save key-value pairs on Exit and load these values when 'Settings' is loaded. Baring this exception, we are not saving any other activity state. This application may save some other statistics as far as the workouts go but no UI element states are saved. A sample Button Click event is :

```
final Button button = (Button) findViewById(R.id.button_id);
            button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
      // Perform action on click
            }
});
```

Pushups.java

In this java file, we associate a variable with our TextView field, a variable for button to exit/pause out of order, an info button to show the instructions. Our logic is very simple.

Pseudo:

OnCreate, wait for X seconds. (User defined X)
Audio . ReadyAudio followed by SetAudio followed by BeginAudio.
Start audio DownAudio. (at T0)
If Proximity detected, assume its the users face. Start Audio UpAudio. (at T1)
If Paused, save state. Save any time diffentials. Proceed with resume.
If no face detected, after (T1-T0), play DownAudio.
After N seconds, we finish activity. (User defined N)

[Attach PUSHUPS.JAVA]

Situps.java

Similar to our pushups.java, we will be associating our fields with some local variables.

Pseudo:

OnCreate, wait for X(2) seconds. (User defined X(2))
Audio, WhistleAudio
If Proximity detected, assume its users head. Start Audio UpAudio. (at 0)
On first trial, play DownAudio after a default interval. (T0)
On proximity detected again, if interval passed is greater than minimum interval, play UpAudio, save T1-T0.
After T1-T0, play DownAudio.. loop.
If interval N(2) reached, finish activity. (User definied N(2))

Runners.java

We find our views by their id.
We update our timer TextView.
If (user has GPS):

MAP on GMAPS
SAVE MAP on return HOME.
SAVE highest altitude,
SAVE lowest altitude,
SAVE speed, distance, time, etc.
Save Log.
Else if (user on Treadmill):
Speed, Altitude.
Save Log.
Case to Chrome. Scenery!!

Settings.java

Find 'key-value' pairs stored already.
Restore values.
Record User changes as they happen with event handler.
If (any value changes): change key-value for the persistent data.

Info.java

Infomational session will in this activity. This activity will have 3 fragments.I> How to Use? II> Help and Customer Support! III> About US.

Switch depending on Context.

GPS and Google Maps

/* insert Code
*/

SensorManager, SensorEvent, SensorEventManager

We set up a count as a variable to count our footsteps. It will be displayed on a TextView object.

```
count = (TextView) findViewById(R.id.count);
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

Add Event Listener, and set up resume to update the steps.

```
@Override
protected void onResume() {
    super.onResume();
    activityRunning = true;
    Sensor countSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
    if (countSensor != null) {
        sensorManager.registerListener(mEventListenerStep , countSensor, SensorManager.SENSOR_DELAY_FASTEST);
```

```
} else {
            Toast.makeText(this, "Count sensor not available!", Toast.LENGTH_LONG).show();
        }

    }

    @Override
    protected void onPause() {
        super.onPause();
        activityRunning = false;
        // if you unregister the last listener, the hardware will stop detecting step events
// sensorManager.unregisterListener(this);
    }



  SensorEventListener  mEventListenerStep = new SensorEventListener() {

        public void onSensorChanged(SensorEvent event) {
            if (activityRunning) {
                count.setText(String.valueOf(event.values[0]) + " Steps Taken Today!");
            }
        }

    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }
};
```