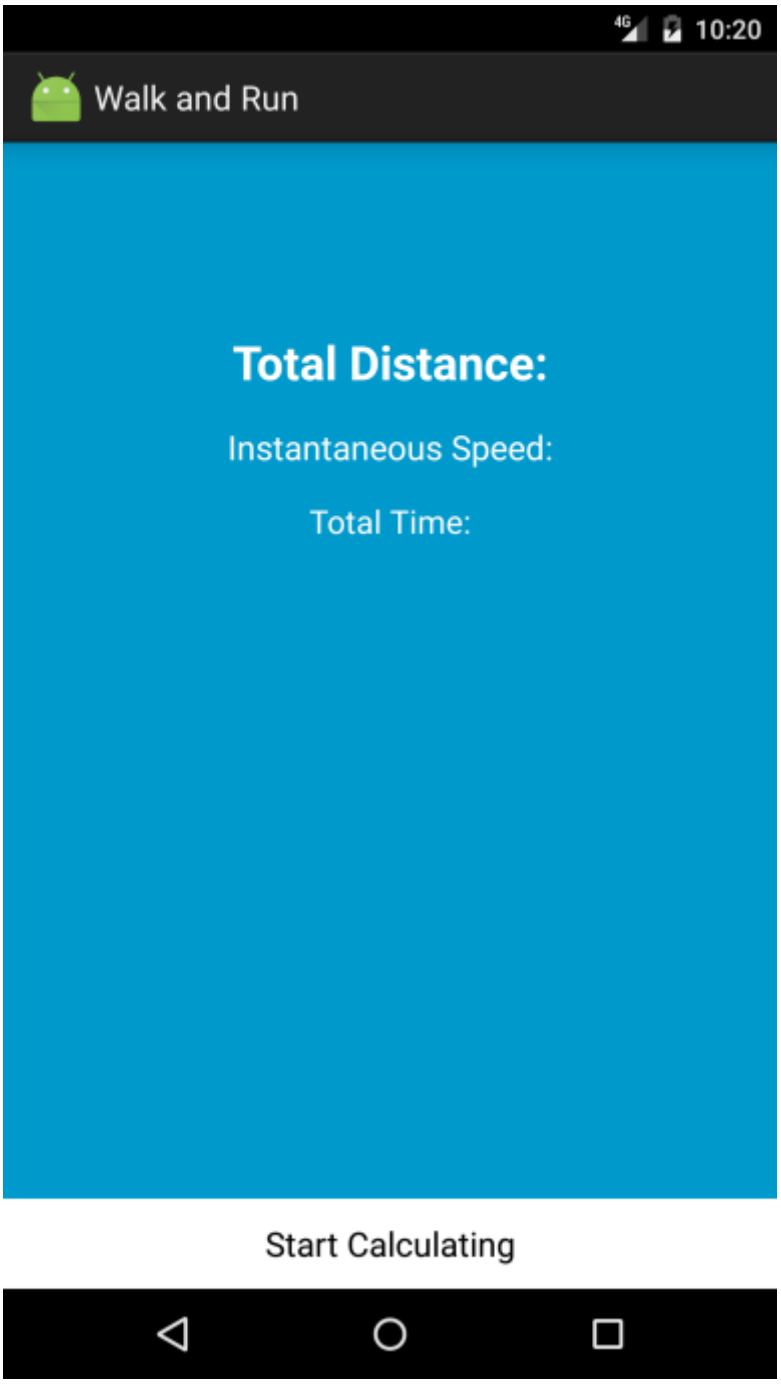
Walk & Run Tracker | Android Studio



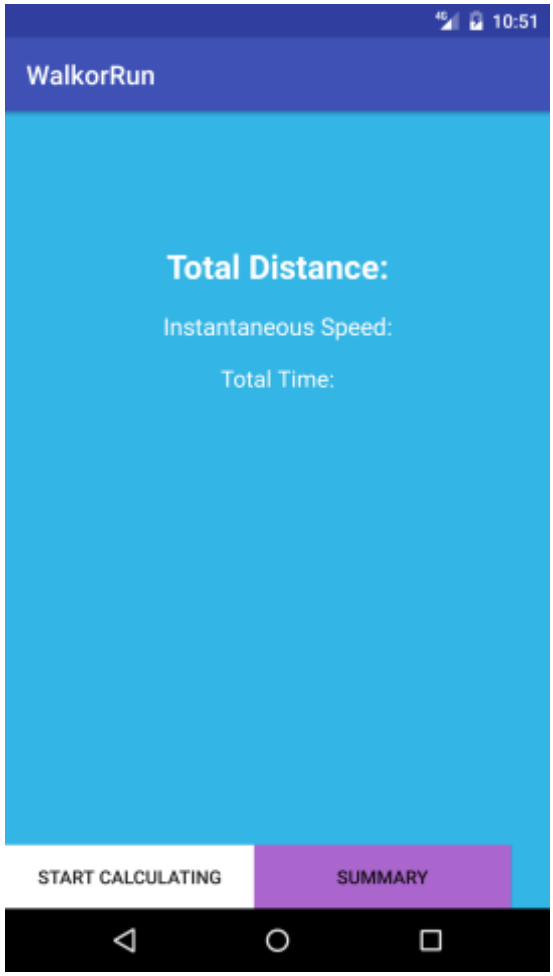
Introduction

In this assignment I am tasked with building an app that will use the GPS sensor to build a fully working walk/run tracker that will track how far and how fast a user has done an activity. I will be required to maintain an entire history for the walk or run that the user is currently doing. The structure of this application will be fairly simple and is detailed below:

- There will be a main activity that will have some stats on the current speed the user is going, total distance travelled, average speed and the elapsed time so far. There should also be a button for stopping and starting the tracking. When tracking is started all fields and information previous listed should be reset.

- There will be a summary activity that will show a bar graph (custom view) of the average time it took for the user to complete each kilometer, along with the stats listed above with the exception of the current speed. For the bar graph I will need a bar for each complete kilometer.

01) Generate the layouts and basic activities that will make up the structure of the application



Here we see the general layout for our application which is the Main Activity. Before the user clicks the "Start Calculating" button, they have the chance of viewing simple coloured layout before using the application. The button is at the very bottom of the page to attract the user's attention.
The light blue background and white text is a good contrast, which makes the app presentable.

The "Summary" button is highlighted in purple to give an alert feeling that this is where the user needs to go in order to view their results.

Now let's view the XML and code for the Main Activity.

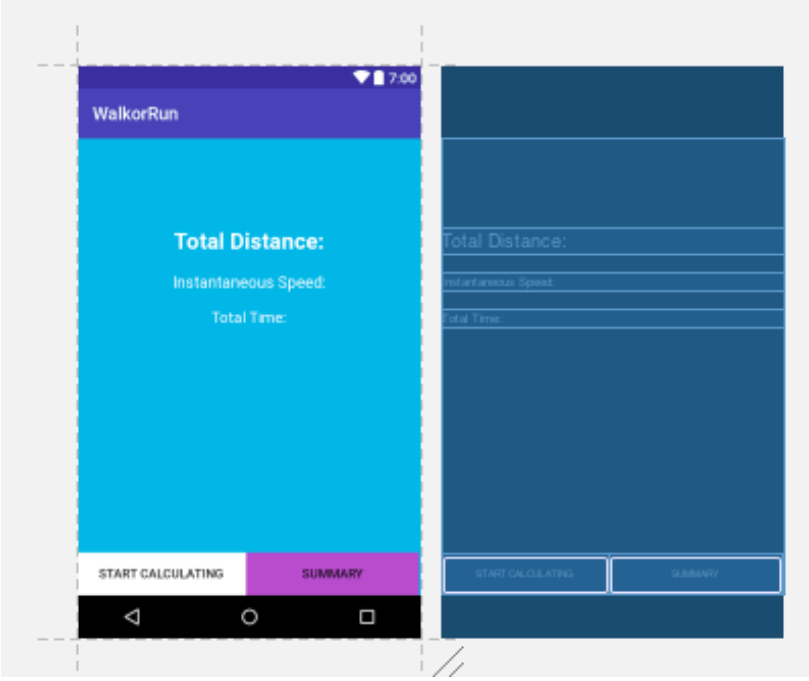
activity_main.xml

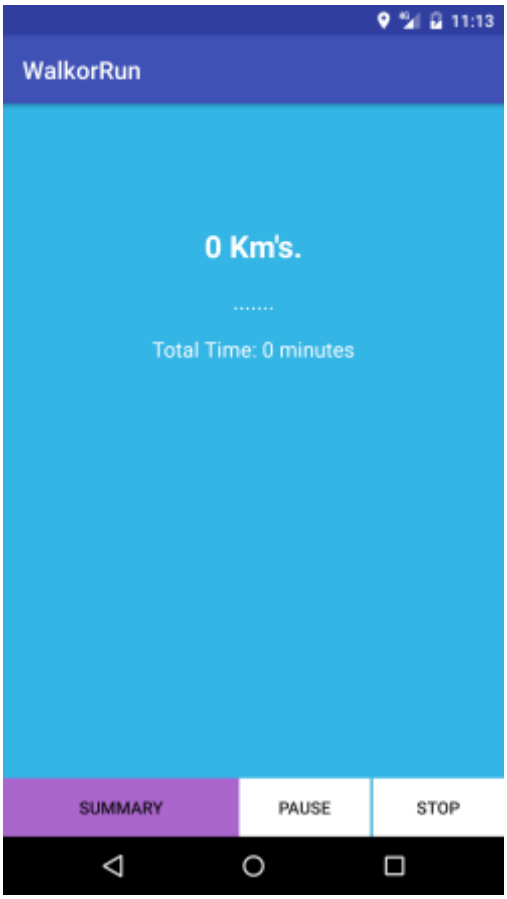
```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"

android:background="@android:color/holo_blue_light"
android:orientation="vertical">
```

Frame Layout is designed to block out an area on the screen to display a single item. Generally, Frame Layout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other.

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:layout_gravity="bottom"
   android:orientation="horizontal">

   <Button
      android:id="@+id/start"
      android:layout_width="188dp"
      android:layout_height="match_parent"
      android:layout_gravity="bottom"
      android:background="@android:color/white"
      android:paddingBottom="0dp"
      android:text="Start Calculating" />
```

Within the Linear Layout is one of our buttons for the main activity. I just want to show the general view of where my buttons are placed. As you can see in the Design View image below, my summary button has a little space at the right hand side. This is to make way for the Pause and Stop buttons, which you will now see when the user starts calculating data.


7:00
WalkorRun
Total Distance:
Instantaneous Speed:
Total Time:
START CALCULATING
SUMMARY
Total Distance:
START CALCULATING
SUMMARY


11:13
WalkorRun
0 Km's.
.....
Total Time: 0 minutes
SUMMARY
PAUSE
STOP

Main_Activity.java

In order for us to use our location/ GPS service on the emulator, we need access. This can be done by adding google play services to our dependency in our app level gradle.

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.google.android.gms:play-services:8.4.0'

    testCompile 'junit:junit:4.12'
}
```

This installs all Google Play Services API which we might not need in some instances. If you would like to know how to include single standard API's, follow this link: https://developers.google.com/android/guides/setup

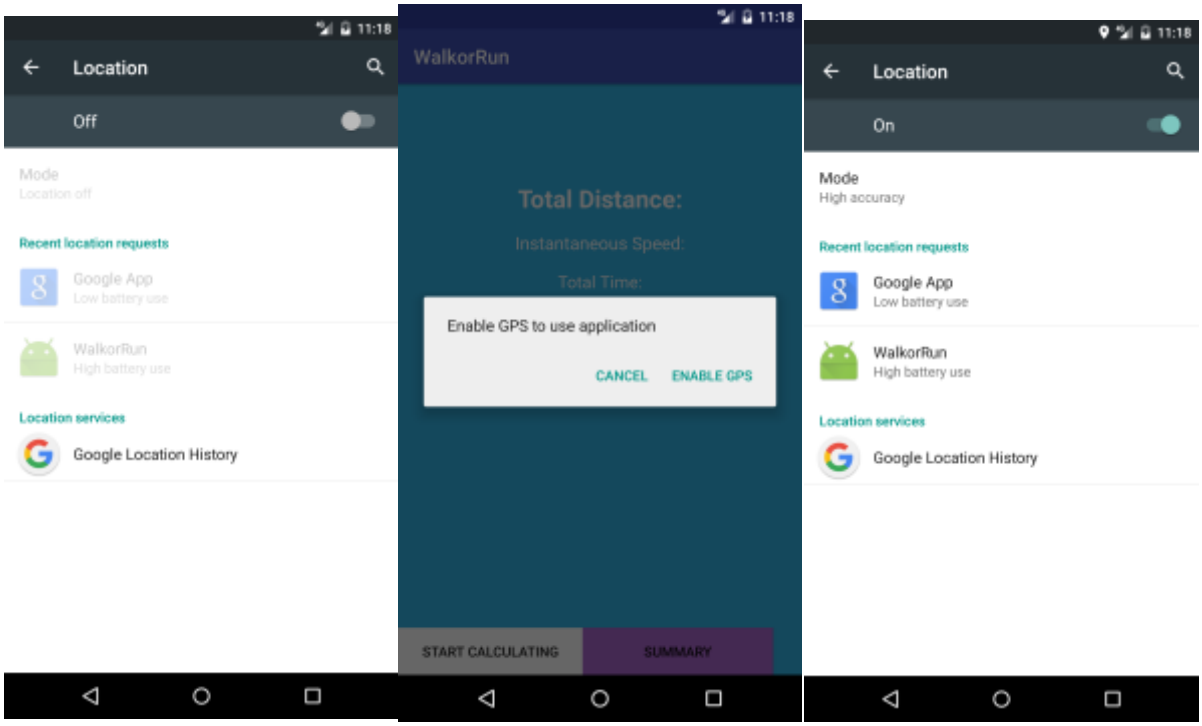
In order to connect to the google location services:

```
private ServiceConnection sc = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        LocationService.LocalBinder binder = (LocationService.LocalBinder) service;
        myService = binder.getService();
        status = true;
    }
```

Check to see if there is a connection for the location to work. If GPS isn't available then give the user an option to turn it on

```
//This method configures the Alert Dialog box.
private void showGPSDisabledAlertToUser() {
    AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);
    alertDialogBuilder.setMessage("Enable GPS to use application")
            .setCancelable(false)
            .setPositiveButton("Enable GPS",
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                            Intent callGPSSettingIntent = new Intent(
                                    android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS);
                            startActivity(callGPSSettingIntent);
                        }
```

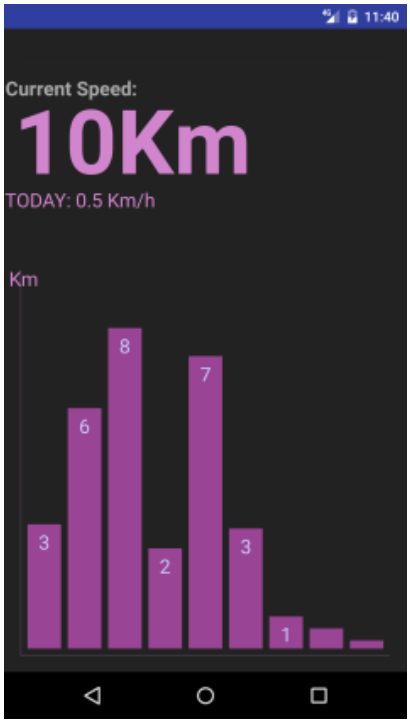
```
                });
    alertDialogBuilder.setNegativeButton("Cancel",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    dialog.cancel();
                }
            });
    AlertDialog alert = alertDialogBuilder.create();
    alert.show();
}
```



As soon as the user clicks on the stop button during calculation, they will be able to enter the summary activity

```
//stop button
    stop.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (status == true)
                unbindService();
            start.setVisibility(View.VISIBLE);
            pause.setText("Pause");
            pause.setVisibility(View.GONE);
            stop.setVisibility(View.GONE);
            p = 0;
```

```
            Intent intent = new Intent(MainActivity.this, Summary.class);
            startActivity(intent);
        }
    });

    // get the button from the layout and add a listener to it
    results = (Button) findViewById(R.id.summary);
    results.setOnClickListener(new OnClickListener() {
        // overridden method to handle a button click
        public void onClick(View v) {
            Intent intent = new Intent(MainActivity.this, Summary.class);
           startActivity(intent);
        }
    });
}
```



Location_Service.java

A class name LocationService.java will update the live values of the speed, distance and duration. It consists of a method called updateUI().

Create location request. The normal frequency is 1000*2 and in case the network request fails we'll use the fastest_interval.

```
private static final long INTERVAL = 1000 * 2;
private static final long FASTEST_INTERVAL = 1000 * 1;
LocationRequest mLocationRequest;
GoogleApiClient mGoogleApiClient;
Location mCurrentLocation, lStart, lEnd;
static double distance = 0;
double speed;


protected void createLocationRequest() {
    mLocationRequest = new LocationRequest();
    mLocationRequest.setInterval(INTERVAL);
    mLocationRequest.setFastestInterval(FASTEST_INTERVAL);
    mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
}
```

In case the user moves from the current location

```
@Override
public void onLocationChanged(Location location) {
    MainActivity.locate.dismiss();
    mCurrentLocation = location;
    if (lStart == null) {
        lStart = mCurrentLocation;
        lEnd = mCurrentLocation;
    } else
        lEnd = mCurrentLocation;

    //Calling the method below updates the  live values of distance and speed to the TextViews.
    updateUI();
    //calculating the speed with getSpeed method it returns speed in m/s so we are converting it into
kmph
    speed = location.getSpeed() * 18 / 5;

}
```

More information:

https://developer.android.com/reference/android/app/Service.html

A bound service is the server in a client-server interface. It allows components (such as activities) to bind to the service, send requests, receive responses, and perform interprocess communication (IPC). A bound service typically lives only while it serves another application component and does not run in the background indefinitely.

```
public class LocalBinder extends Binder {

   public LocationService getService() {
      return LocationService.this;
   }


}
```

The LocalBinder provides the getService() method for clients to retrieve the current instance of LocalService. This allows clients to call public methods in the service. For example, clients can call getRandomNumber() from the service.

This callback is important for handling errors that may occur while attempting to connect with Google.

```
@Override
public void onConnectionFailed(ConnectionResult connectionResult) {

}
```

DrawView.java

The most important step in drawing a custom view is to override the onDraw() method. The parameter to onDraw() is a Canvas object that the view can use to draw itself. The Canvas class defines methods for drawing text, lines, bitmaps, and many other

graphics primitives. You can use these methods in onDraw() to create your custom user interface (UI).

```
Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG);

public DrawView(Context context) {
   super(context);
}
```

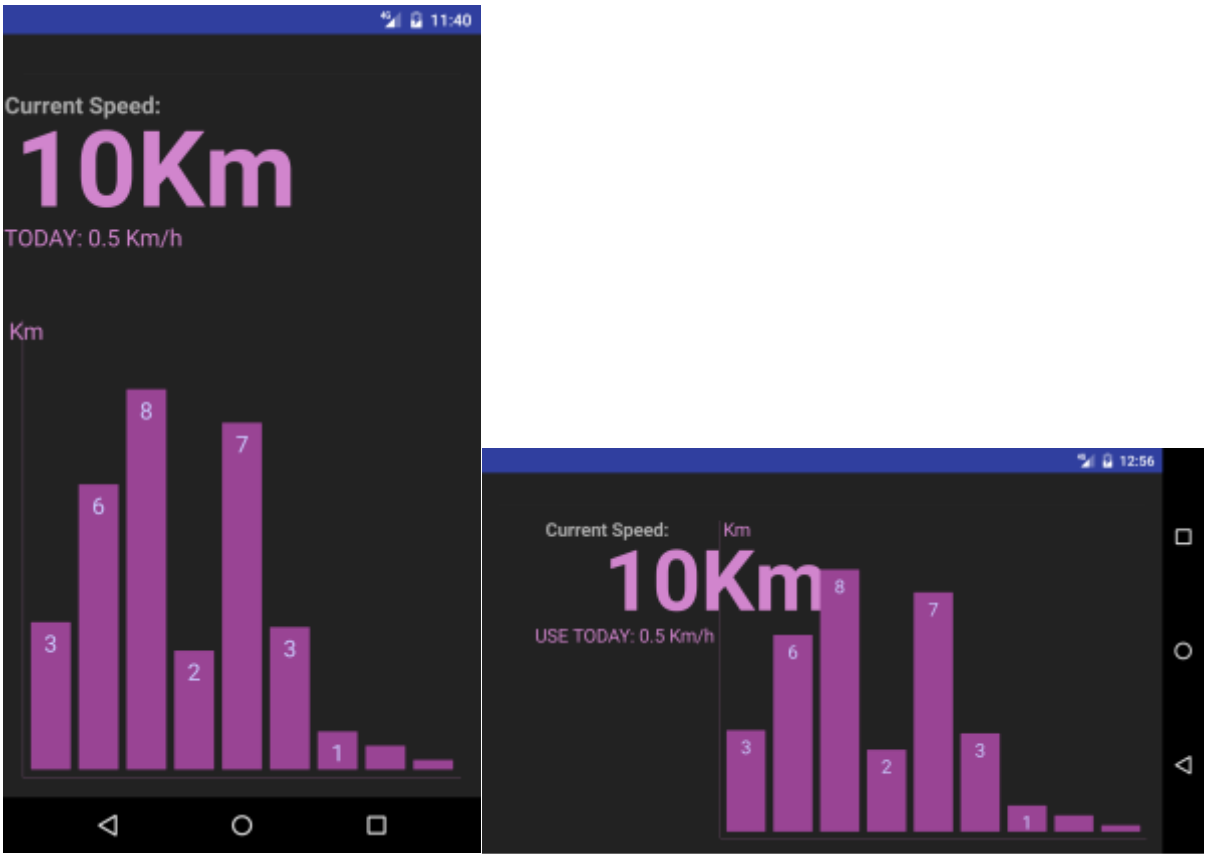
For instance, Canvas provides a method to draw a line, while Paint provides methods to define that line's color. Canvas has a method to draw a rectangle, while Paint defines whether to fill that rectangle with a color or leave it empty. Simply put, Canvas defines shapes that you can draw on the screen, while Paint defines the color, style, font, and so forth of each shape you draw.

More information (https://developer.android.com/training/custom-views/custom-drawing.html)

Changes view from portrait to landscape and vice versa when phone rotates

```
if (screenWidth<screenHeight) {
   // Portrait mode:
   scale = screenWidth / 720.0f;

   left = 30*scale;
   graphWidth = screenWidth - 60*scale;

   graphHeight = 0.6f * screenHeight;
   top = screenHeight - graphHeight-(30*scale);

} else {
   // Landscape mode:
   scale = screenHeight / 720.0f;

   left = screenWidth * 0.35f;
   graphWidth = screenWidth * 0.65f - 30*scale;

   top = (90*scale);
```

```
    graphHeight = screenHeight - (120*scale);
}
```



I did not have time to connect the GPS sensor to the bar graph, thus we will now end the tutorial.