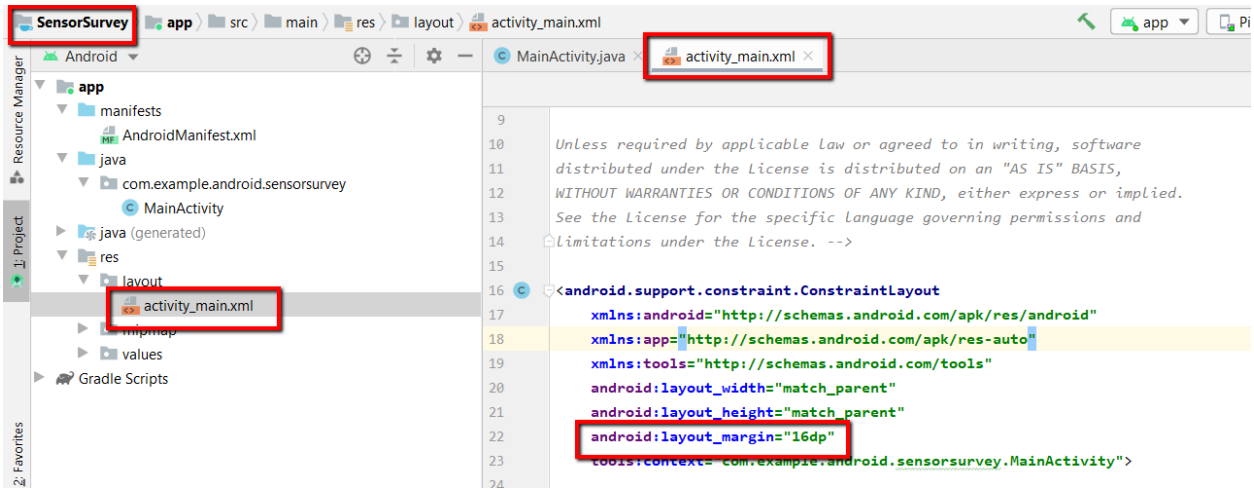
Android Apps.

In this we will learn Android-powered devices which include built-in sensors that measure motion, orientation, and environmental conditions such as ambient light or temperature. These sensors can provide data to your app with high precision and accuracy.

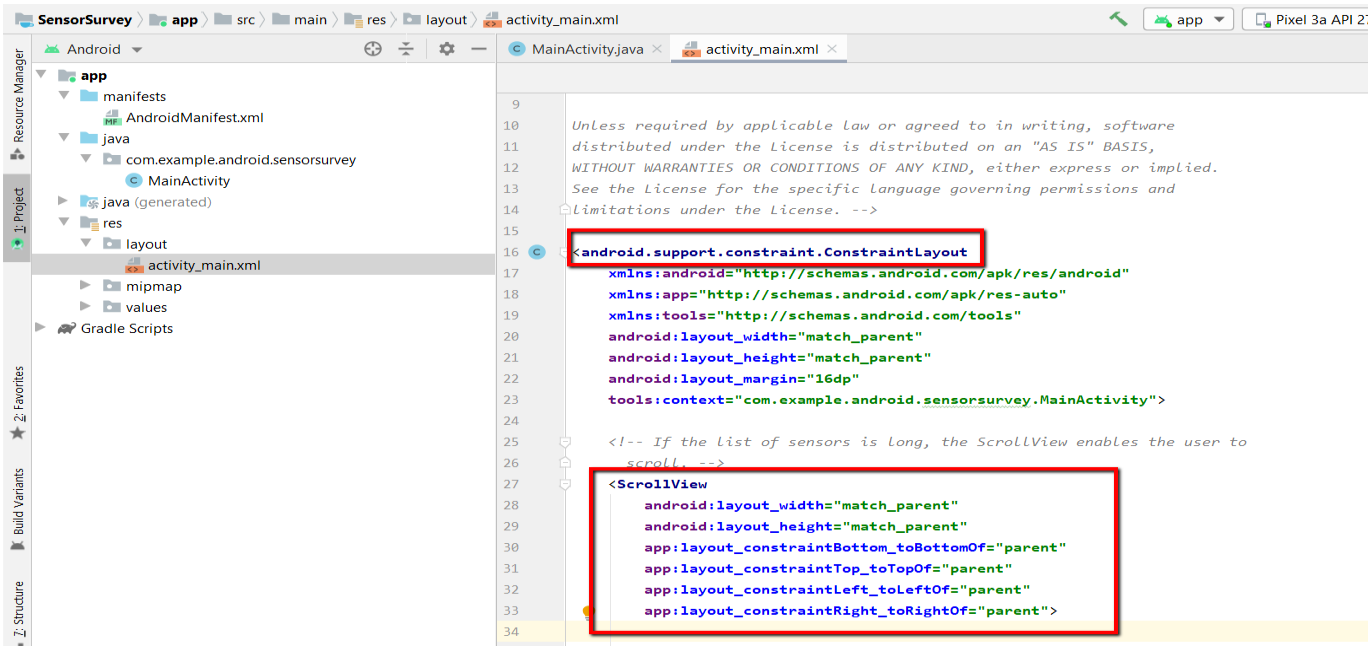
In this assignment we will build two apps, the first app list the available sensors on the device or emulator. The list of sensors is scrollable, if it is too big to fit the screen.

The second app is the modified from the first, which gets data from the ambient light and proximity sensors, and displays that data. Light and proximity sensors are some of the most common Android device sensors.

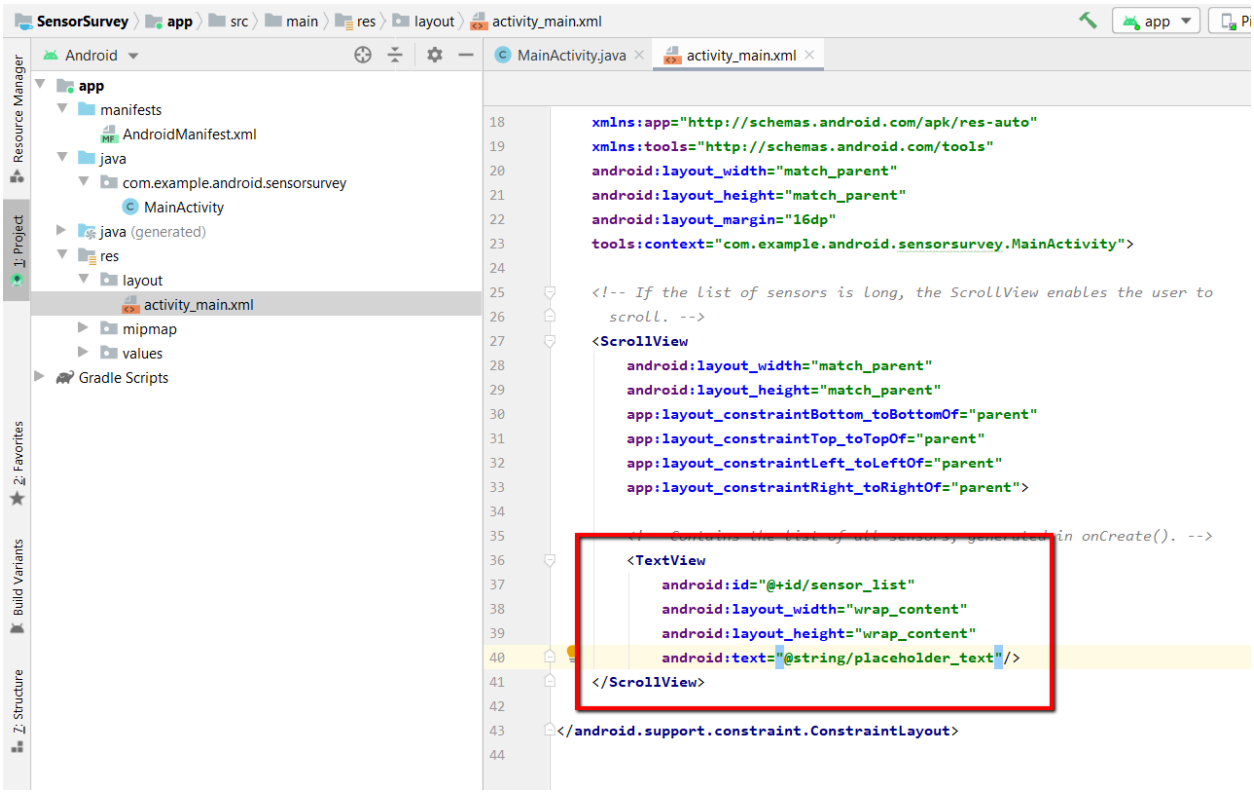
Now starting with the first app, we will create a new project named 'SensorSurvey' and we will open 'activity_main.xml' which is under 'layout' -> 'res' and we will add the margin of 16 dp to the constraint layout. Also delete all the existing 'TextView'.



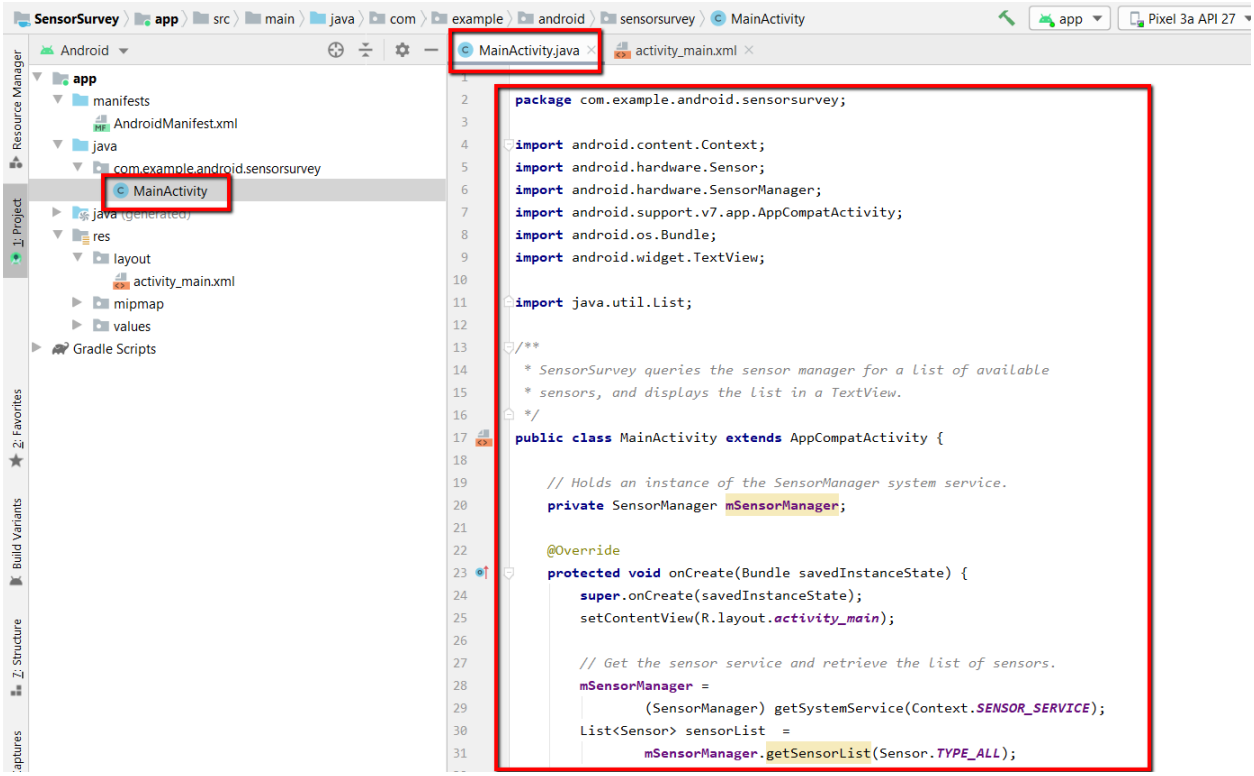
Now for adding 'ScrollView' we will add the following elements inside the constraint layout. The ScrollView is here to allow the list of sensors to scroll if it is longer than the screen.



Now we will add the 'TextView' element inside the 'ScrollView' and add the following attributes into it. This 'TextView' holds the list of sensors. The placeholder text is replaced at runtime by the actual sensor list.

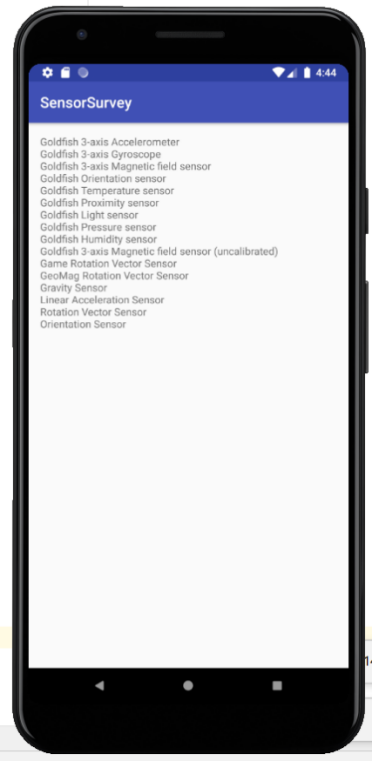


Now we will open ‘MainActivity.java’ and add following code into it.

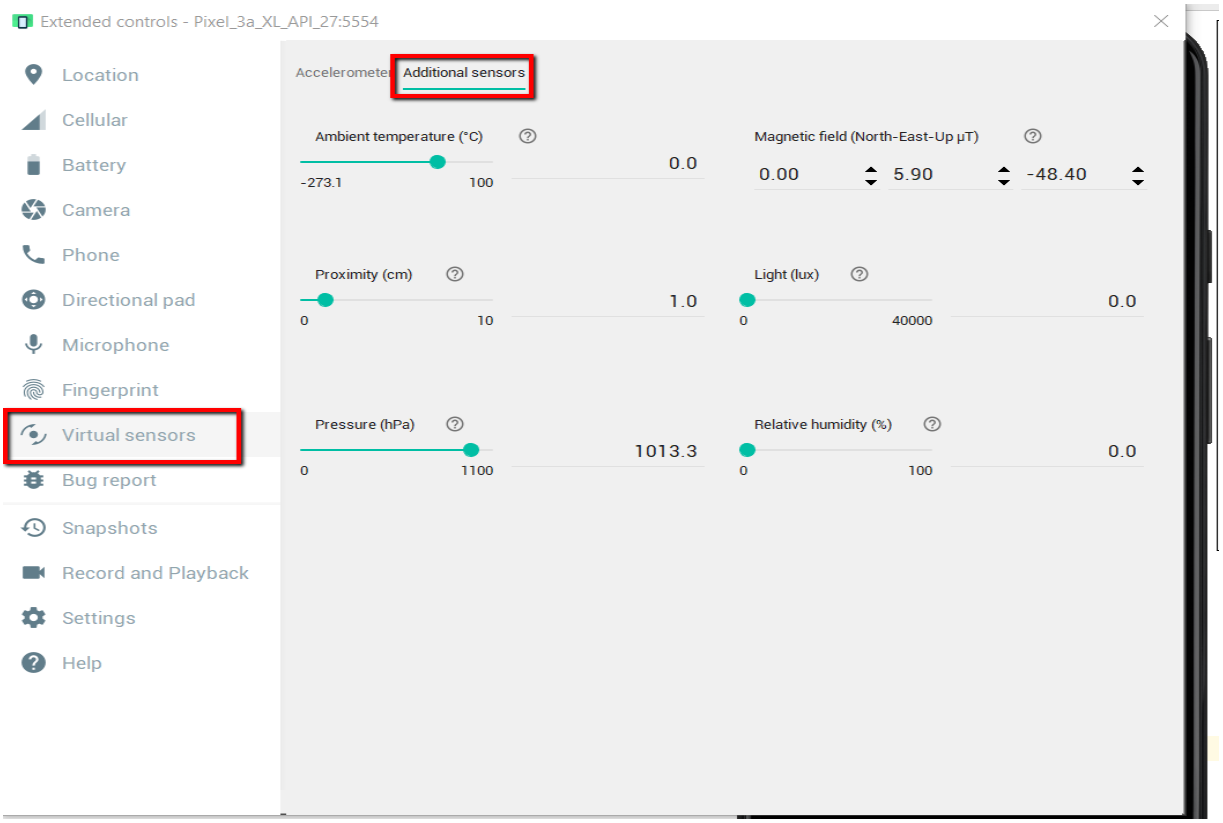



```
        // Iterate through the list of sensors, get the sensor name,
        // append it to the string.
        StringBuilder sensorText = new StringBuilder();
        //String sensorText = "";
        for (Sensor currentSensor : sensorList ) {
            sensorText.append(currentSensor.getName()).append(
                    System.getProperty("line.separator"));
        }

        // Update the sensor list text view to display the list of sensors.
        TextView sensorTextView = (TextView) findViewById(R.id.sensor_list);
        sensorTextView.setText(sensorText);
    }
}
```

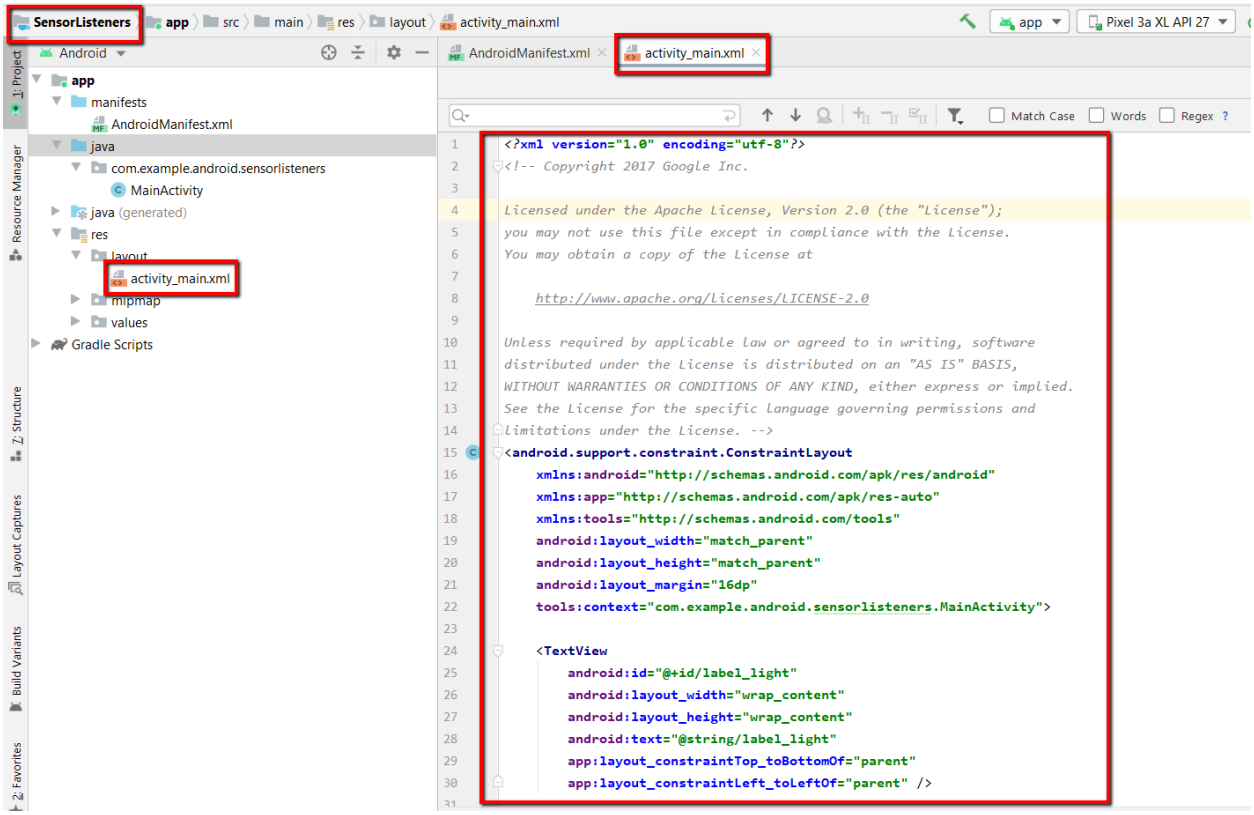
Now we will run the app on the emulator and see the output of the above code.



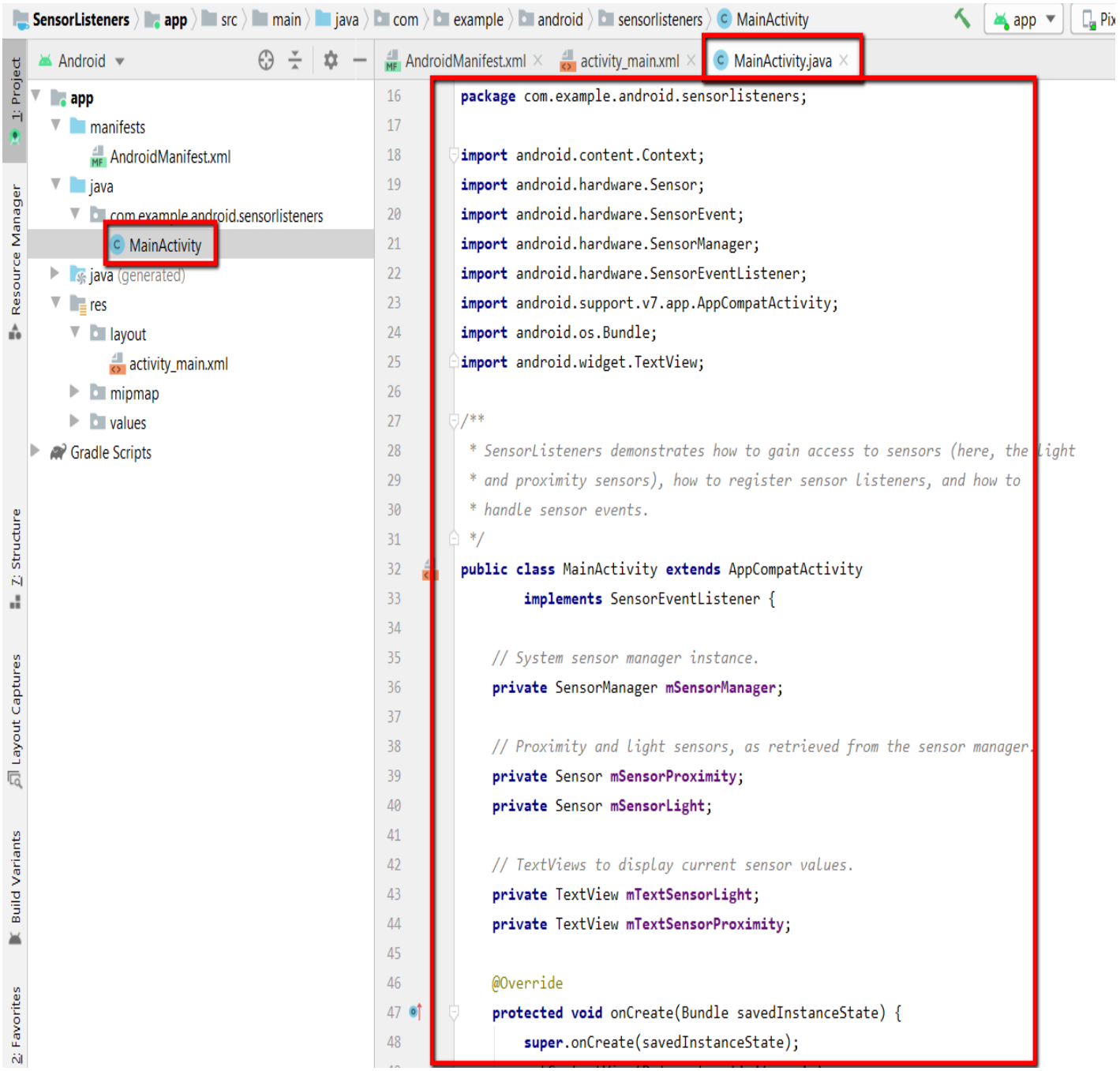
Following are all the additional virtual sensors for the emulator.



Now we will build the second app which will be used to get sensor data. We will open ‘activity_main.xml’ which is in ‘layout’ -> ‘res’ and add the following code into it.



Then we will open ‘MainActivity.java’ and add the following code into it.



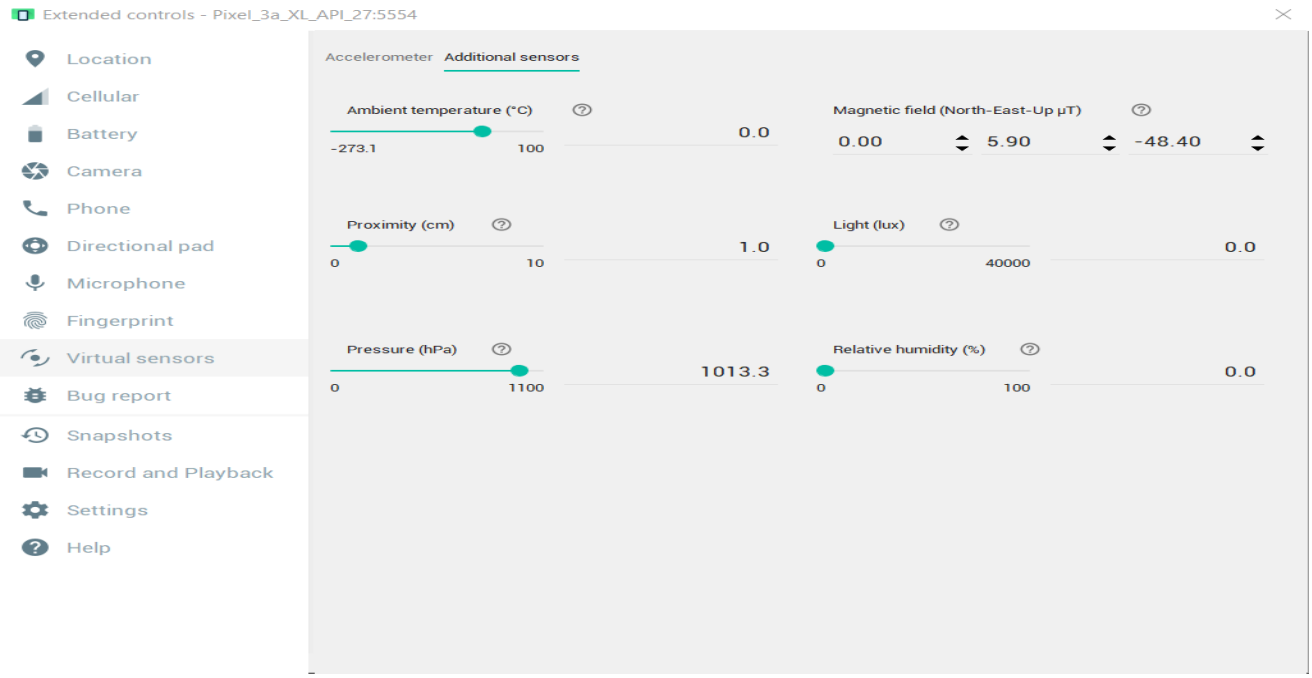
```
        setContentView(R.layout.activity_main);

        // Initialize all view variables.
        mTextSensorLight = (TextView) findViewById(R.id.label_light);
        mTextSensorProximity = (TextView) findViewById(R.id.label_proximity);

        // Get an instance of the sensor manager.
        mSensorManager = (SensorManager) getSystemService(
                Context.SENSOR_SERVICE);

        // Get light and proximity sensors from the sensor manager.
        // The getDefaultSensor() method returns null if the sensor
        // is not available on the device.
        mSensorProximity = mSensorManager.getDefaultSensor(
                Sensor.TYPE_PROXIMITY);
        mSensorLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);

        // Get the error message from string resources.
        String sensor_error = "No sensor";

        // If either mSensorLight or mSensorProximity are null, those sensors
        // are not available in the device.  Set the text to the error message
        if (mSensorLight == null) { mTextSensorLight.setText(sensor_error); }
        if (mSensorProximity == null) {
            mTextSensorProximity.setText(sensor_error);
        }
    }

    @Override
    protected void onStart() {
        super.onStart();
```

```
        if (mSensorProximity != null) {
            mSensorManager.registerListener( listener: this, mSensorProximity,
                    SensorManager.SENSOR_DELAY_NORMAL);
        }
        if (mSensorLight != null) {
            mSensorManager.registerListener( listener: this, mSensorLight,
                    SensorManager.SENSOR_DELAY_NORMAL);
        }
    }

    @Override
    protected void onStop() {
        super.onStop();

        // Unregister all sensor listeners in this callback so they don't
        // continue to use resources when the app is paused.
        mSensorManager.unregisterListener(this);
    }

    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {

        // The sensor type (as defined in the Sensor class).
        int sensorType = sensorEvent.sensor.getType();

        // The new data value of the sensor.  Both the light and proximity
        // sensors report one value at a time, which is always the first
        // element in the values array.
        float currentValue = sensorEvent.values[0];

        switch (sensorType) {
```

```
            case Sensor.TYPE_LIGHT:
                // Set the light sensor text view to the light sensor string
                // from the resources, with the placeholder filled in.
                mTextSensorLight.setText("Light Sensor: {currentValue}");
                break;
            case Sensor.TYPE_PROXIMITY:
                // Set the proximity sensor text view to the light sensor
                // string from the resources, with the placeholder filled in.
                mTextSensorProximity.setText("Proximity Sensor: {currentValue}");
                break;
            default:
                // do nothing
        }
    }

    /**
     * Abstract method in SensorEventListener.  It must be implemented, but is
     * unused in this app.
     */
    @Override
    public void onAccuracyChanged(Sensor sensor, int i) {
    }
}
```

Following is the output on the emulator.

Following are the additional virtual sensors which are available on the emulator.



Sensor-based orientation

The Android platform provides several sensors that enable your app to monitor the motion or position of a device, in addition to other sensors such as the light sensor. Motion sensors such as the accelerometer or gyroscope are useful for monitoring device movement such as tilt, shake, rotation, or swing. Position sensors are useful for determining a device's physical position in relation to the Earth.

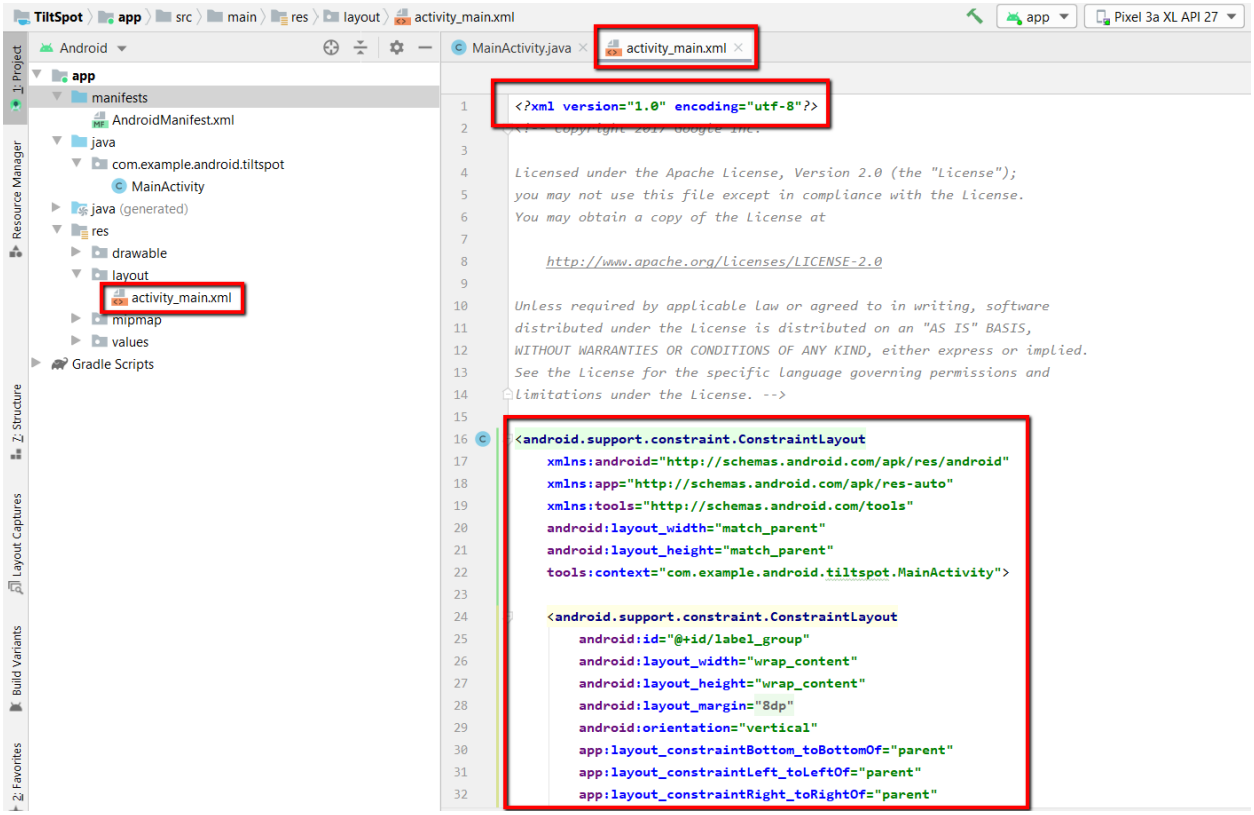
Let’s name the project as “TiltSpot” app. The ‘TiltSpot’ app displays the device orientation angles as numbers and as colored spots along the four edges of the device screen.

- *Azimuth*: The direction (north/south/east/west) the device is pointing. 0 is magnetic north.
- *Pitch*: The top-to-bottom tilt of the device. 0 is flat.
- *Roll*: The left-to-right tilt of the device. 0 is flat.

When you tilt the device, the spots along the edges that are tilted up become darker.

Download the ‘TiltSpot_start’ app from the GitHub and open it in a Android Studio.

Then open ‘activity_main.xml’ which is in the ‘layout’ -> ‘res’ and add the following code into it.



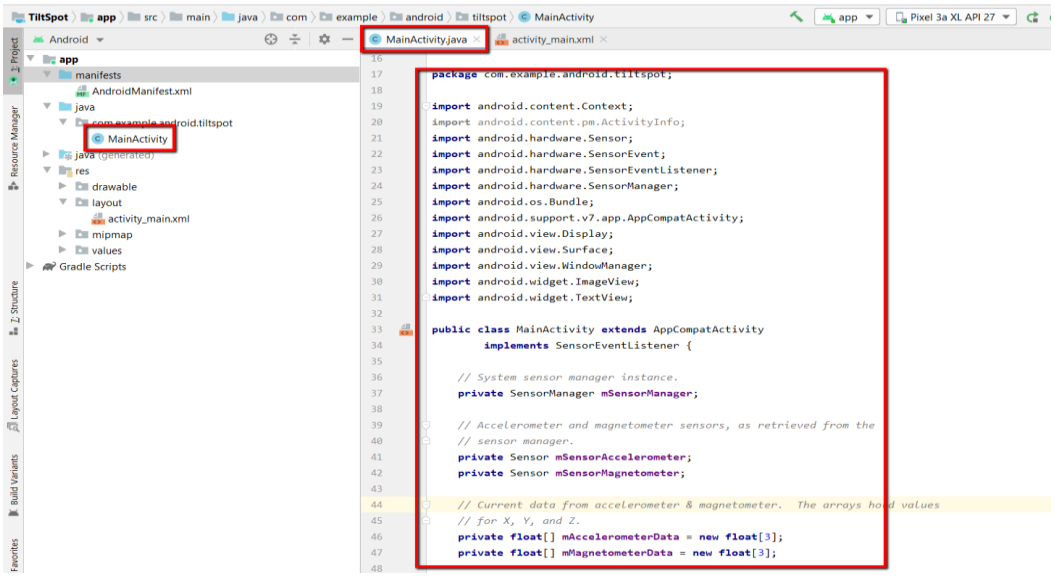
```
    app:layout_constraintTop_toTopOf="parent">

    <!-- Labels -->
    <TextView
        android:id="@+id/label_azimuth"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Azmiuth:"
        style="@style/Label"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="@+id/label_group"/>

    <TextView
        android:id="@+id/label_pitch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Pitch:"
        style="@style/Label"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@id/label_azimuth"/>

    <TextView
        android:id="@+id/label_roll"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Roll:"
        style="@style/Label"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@id/label_pitch"/>
```

```
    <TextView
        android:id="@+id/value_azimuth"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="8dp"
        android:layout_marginStart="8dp"
        android:text="%1$.2f"
        android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"
        app:layout_constraintLeft_toRightOf="@id/label_azimuth"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="@+id/label_group"/>

    <TextView
        android:id="@+id/value_pitch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="%1$.2f"
        android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/value_azimuth"/>

    <TextView
        android:id="@+id/value_roll"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="%1$.2f"
        android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/value_pitch"/>
</android.support.constraint.ConstraintLayout>
```

```
<ImageView
    android:id="@+id/spot_top"
    android:layout_width="84dp"
    android:layout_height="84dp"
    android:layout_marginTop="8dp"
    android:alpha="0.05"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/spot"
    tools:ignore="ContentDescription" />

<ImageView
    android:id="@+id/spot_bottom"
    android:layout_width="84dp"
    android:layout_height="84dp"
    android:layout_marginBottom="8dp"
    android:alpha="0.05"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:srcCompat="@drawable/spot"
    tools:ignore="ContentDescription" />

<ImageView
    android:id="@+id/spot_right"
    android:layout_width="84dp"
    android:layout_height="84dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:alpha="0.05"
    app:layout_constraintBottom_toBottomOf="parent"
```

```
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/spot"
        tools:ignore="ContentDescription"/>

    <ImageView
        android:id="@+id/spot_left"
        android:layout_width="84dp"
        android:layout_height="84dp"
        android:layout_marginLeft="8dp"
        android:layout_marginStart="8dp"
        android:alpha="0.05"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/spot"
        tools:ignore="ContentDescription" />

</android.support.constraint.ConstraintLayout>
```

Now open ‘MainActivity.java’ which is in the ‘Java’ folder and add the following code into it.



```
    // TextViews to display current sensor values.
    private TextView mTextSensorAzimuth;
    private TextView mTextSensorPitch;
    private TextView mTextSensorRoll;

    // ImageView drawables to display spots.
    private ImageView mSpotTop;
    private ImageView mSpotBottom;
    private ImageView mSpotLeft;
    private ImageView mSpotRight;

    // System display. Need this for determining rotation.
    private Display mDisplay;

    // Very small values for the accelerometer (on all three axes) should
    // be interpreted as 0. This value is the amount of acceptable
    // non-zero drift.
    private static final float VALUE_DRIFT = 0.05f;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mTextSensorAzimuth = (TextView) findViewById(R.id.value_azimuth);
        mTextSensorPitch = (TextView) findViewById(R.id.value_pitch);
        mTextSensorRoll = (TextView) findViewById(R.id.value_roll);
        mSpotTop = (ImageView) findViewById(R.id.spot_top);
        mSpotBottom = (ImageView) findViewById(R.id.spot_bottom);
        mSpotLeft = (ImageView) findViewById(R.id.spot_left);
        mSpotRight = (ImageView) findViewById(R.id.spot_right);
```

```
        mSensorManager = (SensorManager) getSystemService(
                Context.SENSOR_SERVICE);
        mSensorAccelerometer = mSensorManager.getDefaultSensor(
                Sensor.TYPE_ACCELEROMETER);
        mSensorMagnetometer = mSensorManager.getDefaultSensor(
                Sensor.TYPE_MAGNETIC_FIELD);

        // Get the display from the window manager (for rotation).
        WindowManager wm = (WindowManager) getSystemService(WINDOW_SERVICE);
        mDisplay = wm.getDefaultDisplay();
    }

    /**
     * Listeners for the sensors are registered in this callback so that
     * they can be unregistered in onStop().
     */
    @Override
    protected void onStart() {
        super.onStart();

        // Listeners for the sensors are registered in this callback and
        // can be unregistered in onStop().
        //
        // Check to ensure sensors are available before registering listeners.
        // Both listeners are registered with a "normal" amount of delay
        // (SENSOR_DELAY_NORMAL).
        if (mSensorAccelerometer != null) {
            mSensorManager.registerListener( listener: this, mSensorAccelerometer,
                    SensorManager.SENSOR_DELAY_NORMAL);
        }
        if (mSensorMagnetometer != null) {
            mSensorManager.registerListener( listener: this, mSensorMagnetometer,
                    SensorManager.SENSOR_DELAY_NORMAL);
```

```
    }
}

@Override
protected void onStop() {
    super.onStop();

    // Unregister all sensor listeners in this callback so they don't
    // continue to use resources when the app is stopped.
    mSensorManager.unregisterListener(this);
}

@Override
public void onSensorChanged(SensorEvent sensorEvent) {
    // The sensor type (as defined in the Sensor class).
    int sensorType = sensorEvent.sensor.getType();

    // The sensorEvent object is reused across calls to onSensorChanged().
    // clone() gets a copy so the data doesn't change out from under us
    switch (sensorType) {
        case Sensor.TYPE_ACCELEROMETER:
            mAccelerometerData = sensorEvent.values.clone();
            break;
        case Sensor.TYPE_MAGNETIC_FIELD:
            mMagnetometerData = sensorEvent.values.clone();
            break;
        default:
            return;
    }
```

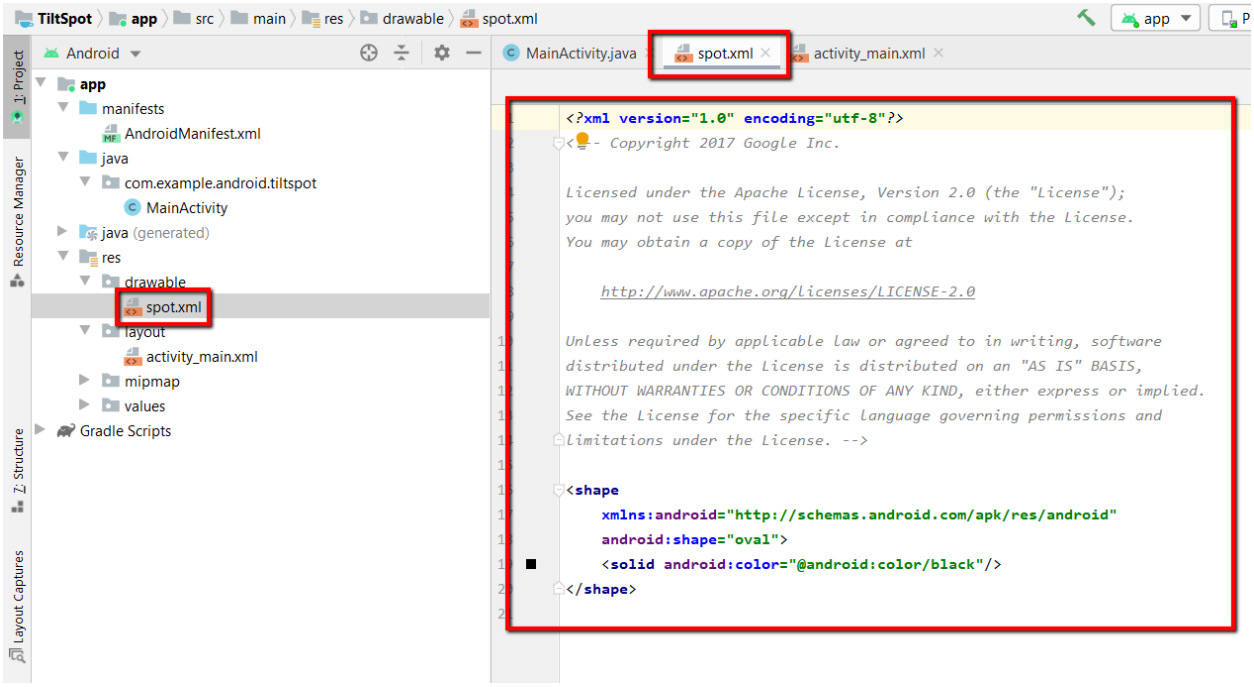
```
switch (sensorType) {
    case Sensor.TYPE_ACCELEROMETER:
        mAccelerometerData = sensorEvent.values.clone();
        break;
    case Sensor.TYPE_MAGNETIC_FIELD:
        mMagnetometerData = sensorEvent.values.clone();
        break;
    default:
        return;
}
// Compute the rotation matrix: merges and translates the data
// from the accelerometer and magnetometer, in the device coordinate
// system, into a matrix in the world's coordinate system.
//
// The second argument is an inclination matrix, which isn't
// used in this example.
float[] rotationMatrix = new float[9];
boolean rotationOK = SensorManager.getRotationMatrix(rotationMatrix,
        I: null, mAccelerometerData, mMagnetometerData);

// Remap the matrix based on current device/activity rotation.
float[] rotationMatrixAdjusted = new float[9];
switch (mDisplay.getRotation()) {
    case Surface.ROTATION_0:
        rotationMatrixAdjusted = rotationMatrix.clone();
        break;
    case Surface.ROTATION_90:
        SensorManager.remapCoordinateSystem(rotationMatrix,
                SensorManager.AXIS_Y, SensorManager.AXIS_MINUS_X,
                rotationMatrixAdjusted);
        break;
    case Surface.ROTATION_180:
```

```
            SensorManager.remapCoordinateSystem(rotationMatrix,
                    SensorManager.AXIS_MINUS_X, SensorManager.AXIS_MINUS_Y,
                    rotationMatrixAdjusted);
            break;
        case Surface.ROTATION_270:
            SensorManager.remapCoordinateSystem(rotationMatrix,
                    SensorManager.AXIS_MINUS_Y, SensorManager.AXIS_X,
                    rotationMatrixAdjusted);
            break;
    }

    // Get the orientation of the device (azimuth, pitch, roll) based
    // on the rotation matrix. Output units are radians.
    float orientationValues[] = new float[3];
    if (rotationOK) {
        SensorManager.getOrientation(rotationMatrixAdjusted,
                orientationValues);
    }

    // Pull out the individual values from the array.
    float azimuth = orientationValues[0];
    float pitch = orientationValues[1];
    float roll = orientationValues[2];

    // Pitch and roll values that are close to but not 0 cause the
    // animation to flash a lot. Adjust pitch and roll to 0 for very
    // small values (as defined by VALUE_DRIFT).
    if (Math.abs(pitch) < VALUE_DRIFT) {
        pitch = 0;
    }
    if (Math.abs(roll) < VALUE_DRIFT) {
        roll = 0;
```

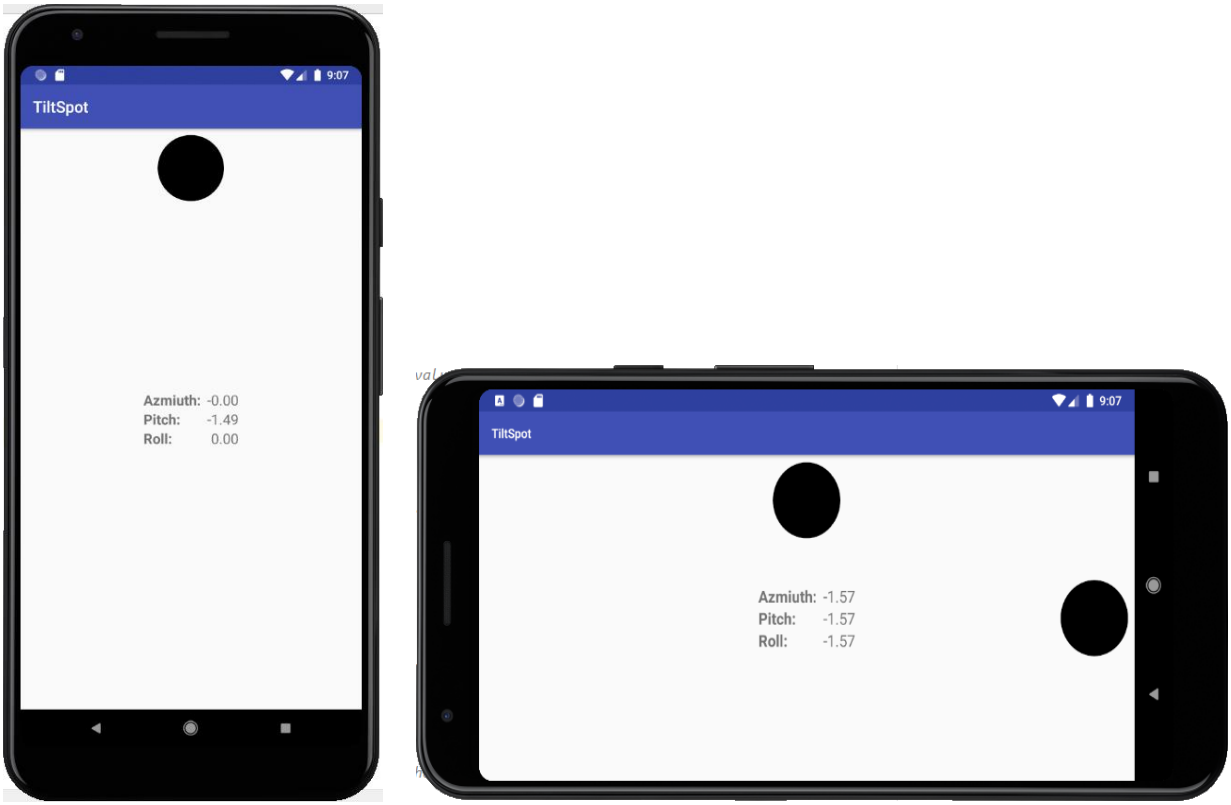
```
    }

    // Fill in the string placeholders and set the textview text.
    mTextSensorAzimuth.setText("{azimuth}");
    mTextSensorPitch.setText("{pitch}");
    mTextSensorRoll.setText("{roll}");

    // Reset all spot values to 0. Without this animation artifacts can
    // happen with fast tilts.
    mSpotTop.setAlpha(0f);
    mSpotBottom.setAlpha(0f);
    mSpotLeft.setAlpha(0f);
    mSpotRight.setAlpha(0f);

    // Set spot color (alpha/opacity) equal to pitch/roll.
    // this is not a precise grade (pitch/roll can be greater than 1)
    // but it's close enough for the animation effect.
    if (pitch > 0) {
        mSpotBottom.setAlpha(pitch);
    } else {
        mSpotTop.setAlpha(Math.abs(pitch));
    }
    if (roll > 0) {
        mSpotLeft.setAlpha(roll);
    } else {
        mSpotRight.setAlpha(Math.abs(roll));
    }
}
```

```
     */
    @Override
    public void onAccuracyChanged(Sensor sensor, int i) {
    }
}
```

For adding the spot in the app, we will add ‘spot.xml’ file in the ‘drawable’ -> ‘res’ file and add the following code into it.



Now when we run the application, we get the following output on the emulator. The colors of the spots change on the correct edges of the device, regardless of how the device is rotated.



Device Location

Users are constantly moving around in the world, usually with their phones in their pockets. Advances in GPS and network technologies have made it possible to create Android apps with precise location awareness, using the location services APIs included in Google Play services.

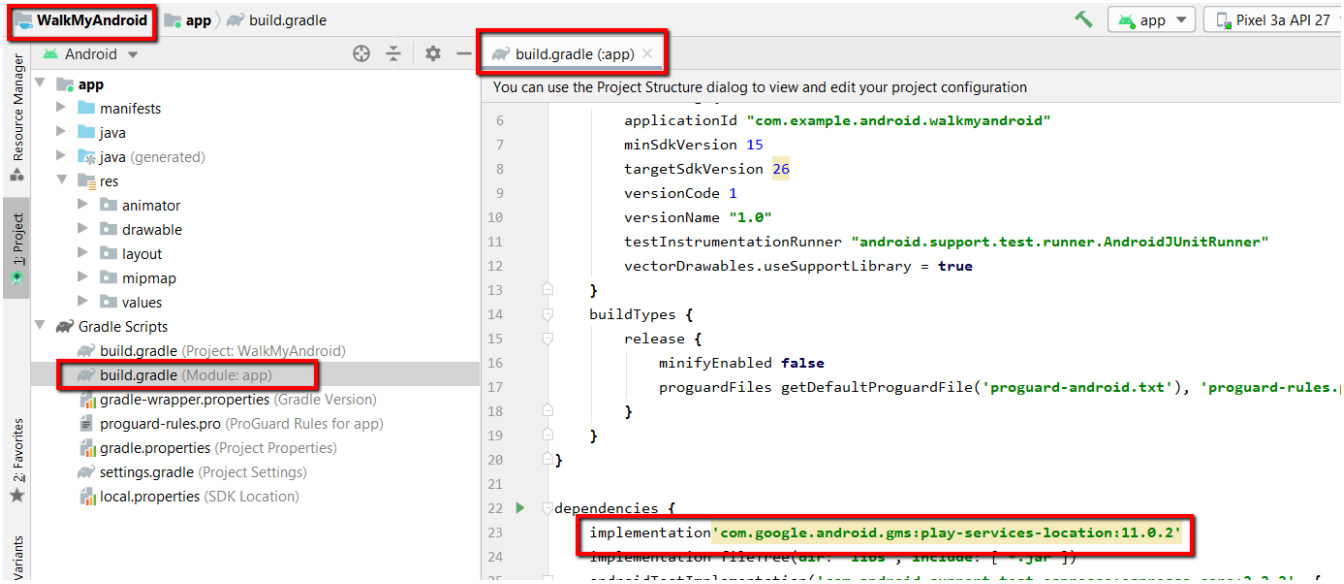
Task 1: Set up location services.

Download the starter app from the google code labs, and open it in Android Studio. Rename the app from 'WalkMyAndroid-Starter' to 'WalkMyAndroid' and run it. You may have to update your Android SDK. Now when you run the project in the emulator their will be a button called 'Get Location' in the UI, when you click it it will not reflect any changes.

Set up Google Play services.

1. Open Android Studio.
2. Select **Tools > Android > SDK Manager**.
3. Select the **SDK Tools** tab.
4. Expand **Support Repository**, select **Google Repository**, and click **OK**.

Then after this in-order to add Google Play Service package into your project add the following code in "build.gradle (Module : app)" which is in the "Gradle Scripts"

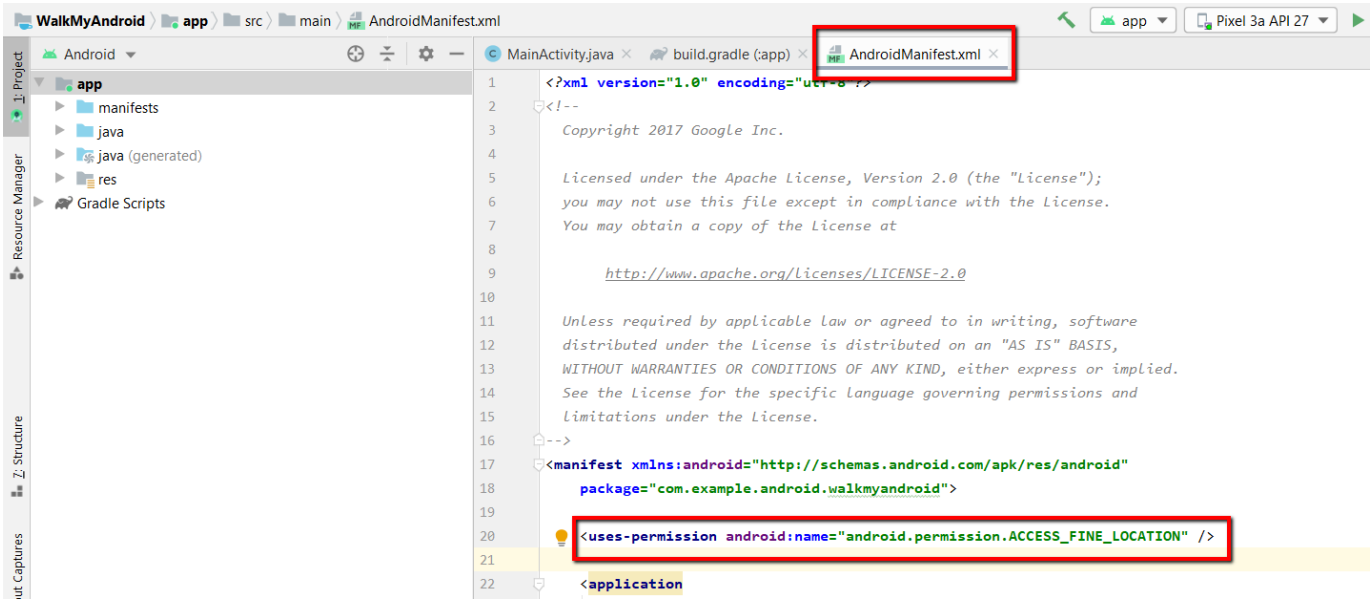


Task 2: Get the last known location.

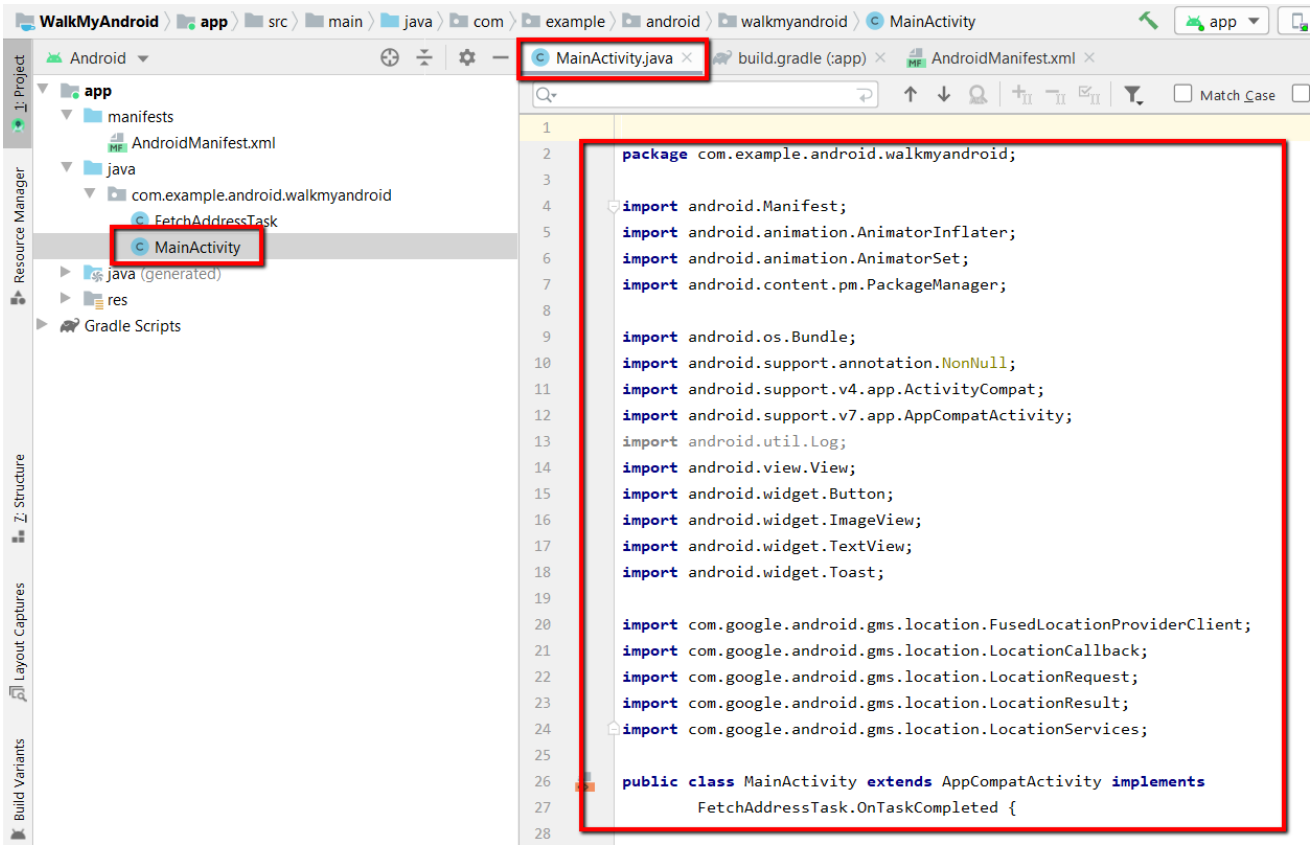
Using the location API requires permission from the user. Android offers two location permissions:

- Access Coarse Location
- Access Fine Location

The permission you choose determines the accuracy of the location returned by the API. We use the Access Fine Location permission, because we want the most accurate location information possible.



Now Open ‘MainActivity.java file’ and add the following code into it.



```java
// Constants
private static final int REQUEST_LOCATION_PERMISSION = 1;
private static final String TRACKING_LOCATION_KEY = "tracking_location";

// Views
private Button mLocationButton;
private TextView mLocationTextView;
private ImageView mAndroidImageView;

// Location classes
private boolean mTrackingLocation;
private FusedLocationProviderClient mFusedLocationClient;
private LocationCallback mLocationCallback;

// Animation
private AnimatorSet mRotateAnim;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mLocationButton = (Button) findViewById(R.id.button_location);
    mLocationTextView = (TextView) findViewById(R.id.textview_location);
    mAndroidImageView = (ImageView) findViewById(R.id.imageview_android);


    // Initialize the FusedLocationClient.
    mFusedLocationClient = LocationServices.getFusedLocationProviderClient(
            activity: this);
```

```
// Set up the animation.
mRotateAnim = (AnimatorSet) AnimatorInflater.loadAnimator
        ( context: this, R.animator.rotate);
mRotateAnim.setTarget(mAndroidImageView);

// Restore the state if the activity is recreated.
if (savedInstanceState != null) {
    mTrackingLocation = savedInstanceState.getBoolean(
            TRACKING_LOCATION_KEY);
}

// Set the listener for the location button.
mLocationButton.setOnClickListener(new View.OnClickListener() {
    /**
     * Toggle the tracking state.
     * @param v The track location button.
     */
    @Override
    public void onClick(View v) {
        if (!mTrackingLocation) {
            startTrackingLocation();
        } else {
            stopTrackingLocation();
        }
    }
});

// Initialize the location callbacks.
mLocationCallback = new LocationCallback() {
    /**
```

```
        @Override
        public void onLocationResult(LocationResult locationResult) {
            // If tracking is turned on, reverse geocode into an address
            if (mTrackingLocation) {
                new FetchAddressTask( applicationContext: MainActivity.this,  listener: MainActivity.this)
                        .execute(locationResult.getLastLocation());
            }
        }
    };
}

/**
 * Starts tracking the device. Checks for
 * permissions, and requests them if they aren't present. If they are,
 * requests periodic location updates, sets a loading text and starts the
 * animation.
 */
private void startTrackingLocation() {
    if (ActivityCompat.checkSelfPermission( context: this,
            Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: this, new String[]
                        {Manifest.permission.ACCESS_FINE_LOCATION},
                REQUEST_LOCATION_PERMISSION);
    } else {
        mTrackingLocation = true;
        mFusedLocationClient.requestLocationUpdates
                (getLocationRequest(),
                        mLocationCallback,
                         looper: null /* Looper */);
```

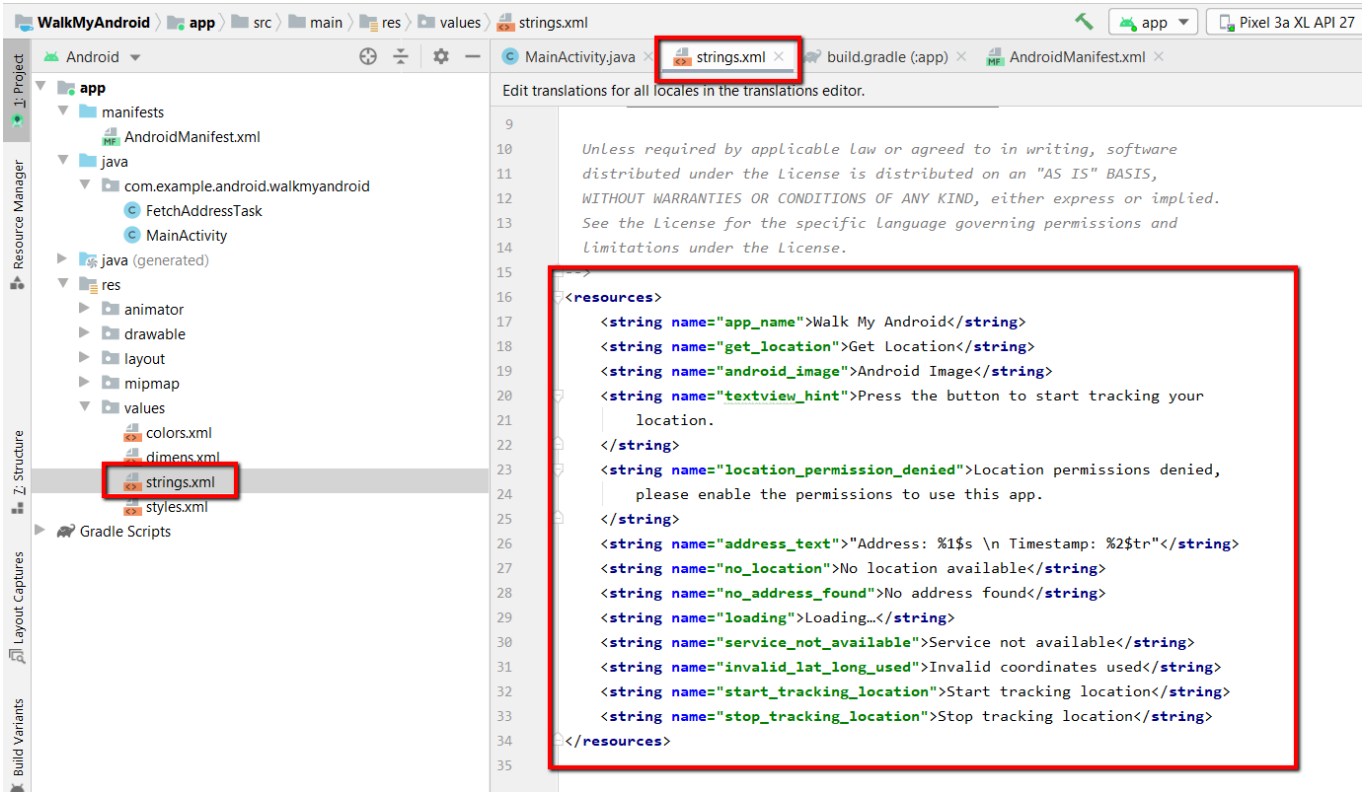
```
            mLocationTextView.setText(getString(R.string.address_text,
                    getString(R.string.loading),
                    System.currentTimeMillis()));
            mLocationButton.setText(R.string.stop_tracking_location);
            mRotateAnim.start();
        }
    }


    /**
     * Stops tracking the device. Removes the location
     * updates, stops the animation, and resets the UI.
     */
    private void stopTrackingLocation() {
        if (mTrackingLocation) {
            mTrackingLocation = false;
            mLocationButton.setText(R.string.start_tracking_location);
            mLocationTextView.setText(R.string.textview_hint);
            mRotateAnim.end();
        }
    }


    /**
     * Sets up the location request.
     *
     * @return The LocationRequest object containing the desired parameters.
     */
    private LocationRequest getLocationRequest() {
        LocationRequest locationRequest = new LocationRequest();
        locationRequest.setInterval(10000);
```

```
        locationRequest.setFastestInterval(5000);
        locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
        return locationRequest;
    }


    /**
     * Saves the last location on configuration change
     */
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        outState.putBoolean(TRACKING_LOCATION_KEY, mTrackingLocation);
        super.onSaveInstanceState(outState);
    }

    /**
     * Callback that is invoked when the user responds to the permissions
     * dialog.
     *
     * @param requestCode  Request code representing the permission request
     *                     issued by the app.
     * @param permissions  An array that contains the permissions that were
     *                     requested.
     * @param grantResults An array with the results of the request for each
     *                     permission requested.
     */
    @Override
    public void onRequestPermissionsResult(int requestCode,
            @NonNull String[] permissions,
            @NonNull int[] grantResults) {
        switch (requestCode) {
            case REQUEST_LOCATION_PERMISSION:
```

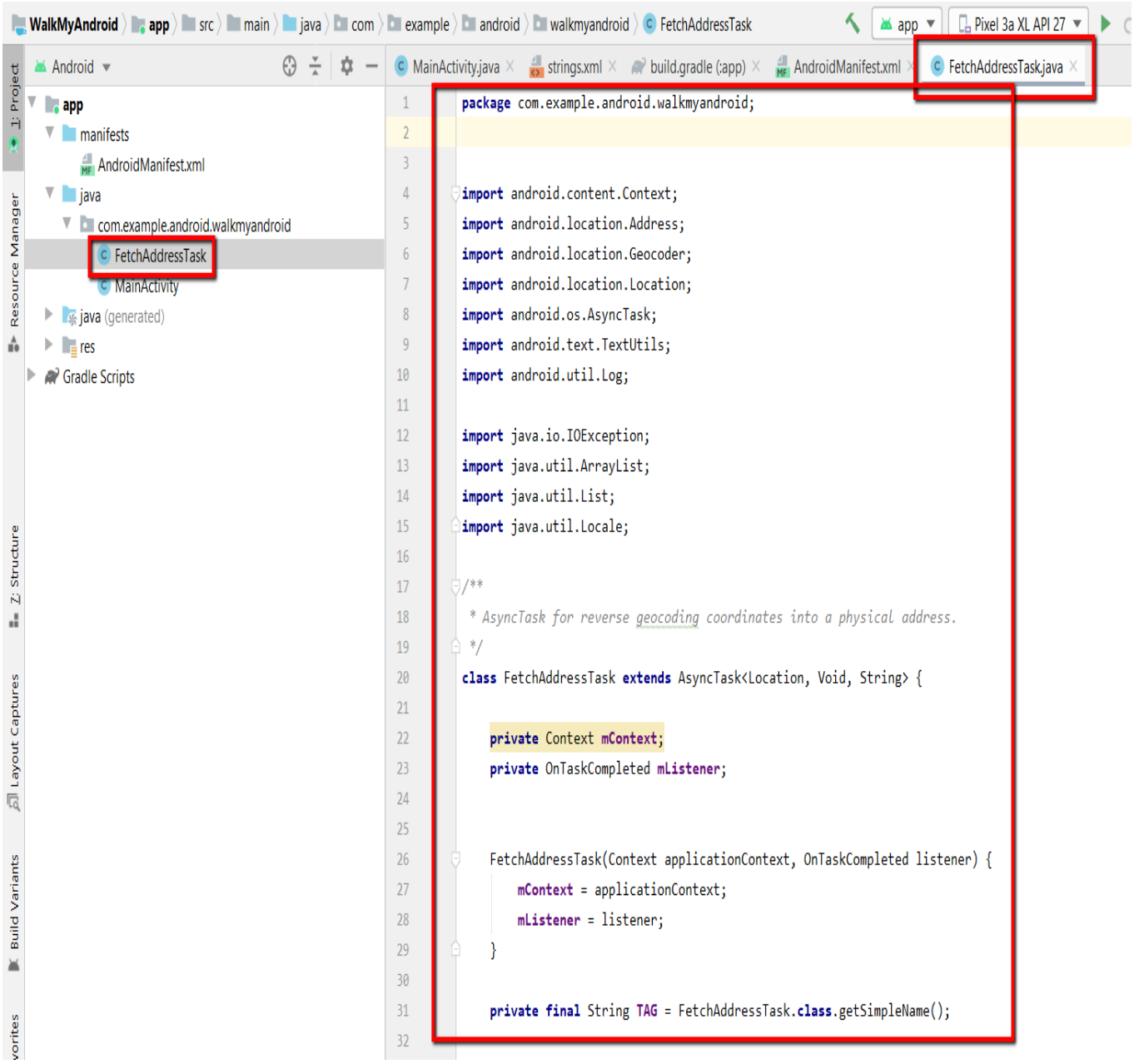
```
                if (grantResults.length > 0
                        && grantResults[0]
                        == PackageManager.PERMISSION_GRANTED) {
                    startTrackingLocation();
                } else {
                    Toast.makeText( context: this,
                            R.string.location_permission_denied,
                            Toast.LENGTH_SHORT).show();
                }
                break;
        }
    }

    @Override
    public void onTaskCompleted(String result) {
        if (mTrackingLocation) {
            // Update the UI
            mLocationTextView.setText(getString(R.string.address_text,
                            result, System.currentTimeMillis()));
        }
    }

    @Override
    protected void onPause() {
        if (mTrackingLocation) {
            stopTrackingLocation();
            mTrackingLocation = true;
        }
        super.onPause();
    }
```

```
    @Override
    protected void onResume() {
        if (mTrackingLocation) {
            startTrackingLocation();
        }
        super.onResume();
    }

}
```

Now open ‘strings.xml’ file and add the following code into it.



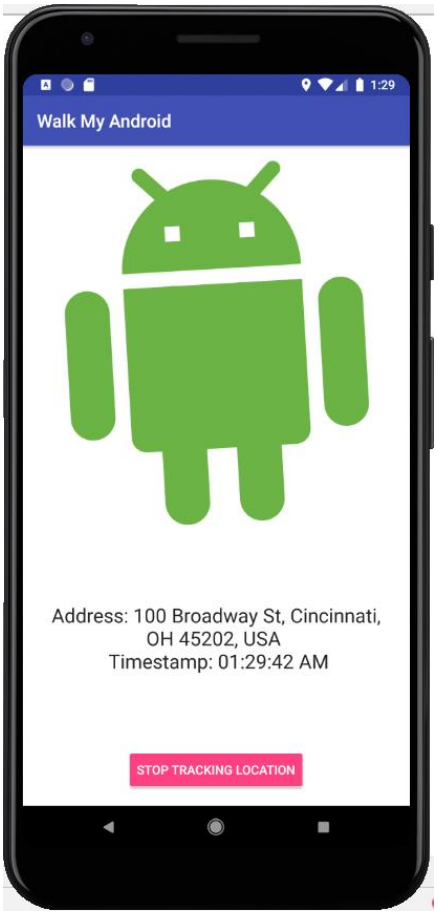
Now Open file ‘FetchAddressTask.java’ and add following code into it.


WalkMyAndroid
app
src
main
java
com
example
android
walkmyandroid
FetchAddressTask
app
Pixel 3a XL API 27
Android
MainActivity.java
strings.xml
build.gradle (:app)
AndroidManifest.xml
FetchAddressTask.java
app
manifests
AndroidManifest.xml
java
com.example.android.walkmyandroid
FetchAddressTask
MainActivity
java (generated)
res
Gradle Scripts
1: Project
Resource Manager
Z: Structure
Layout Captures
Build Variants
package com.example.android.walkmyandroid;
import android.content.Context;
import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.os.AsyncTask;
import android.text.TextUtils;
import android.util.Log;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
/**
* AsyncTask for reverse geocoding coordinates into a physical address.
*/
class FetchAddressTask extends AsyncTask<Location, Void, String> {
private Context mContext;
private OnTaskCompleted mListener;
FetchAddressTask(Context applicationContext, OnTaskCompleted listener) {
mContext = applicationContext;
mListener = listener;
}
private final String TAG = FetchAddressTask.class.getSimpleName();

```
@Override
protected String doInBackground(Location... params) {
    // Set up the geocoder
    Geocoder geocoder = new Geocoder(mContext,
            Locale.getDefault());

    // Get the passed in location
    Location location = params[0];
    List<Address> addresses = null;
    String resultMessage = "";

    try {
        addresses = geocoder.getFromLocation(
                location.getLatitude(),
                location.getLongitude(),
                // In this sample, get just a single address
                maxResults: 1);
    } catch (IOException ioException) {
        // Catch network or other I/O problems
        resultMessage = mContext.getString(R.string.service_not_available);
        Log.e(TAG, resultMessage, ioException);
    } catch (IllegalArgumentException illegalArgumentException) {
        // Catch invalid latitude or longitude values
        resultMessage = mContext.getString(R.string.invalid_lat_long_used);
        Log.e(TAG, msg: resultMessage + ". " +
                "Latitude = " + location.getLatitude() +
                ", Longitude = " +
                location.getLongitude(), illegalArgumentException);
    }
```

```
        if (addresses == null || addresses.size() == 0) {
            if (resultMessage.isEmpty()) {
                resultMessage = mContext.getString(R.string.no_address_found);
                Log.e(TAG, resultMessage);
            }
        } else {
            // If an address is found, read it into resultMessage
            Address address = addresses.get(0);
            ArrayList<String> addressParts = new ArrayList<>();

            // Fetch the address lines using getAddressLine,
            // join them, and send them to the thread
            for (int i = 0; i <= address.getMaxAddressLineIndex(); i++) {
                addressParts.add(address.getAddressLine(i));
            }

            resultMessage = TextUtils.join(
                    delimiter: "\n",
                    addressParts);

        }

        return resultMessage;
    }


    @Override
    protected void onPostExecute(String address) {
        mListener.onTaskCompleted(address);
        super.onPostExecute(address);
    }

    interface OnTaskCompleted {
        void onTaskCompleted(String result);
    }
}
```

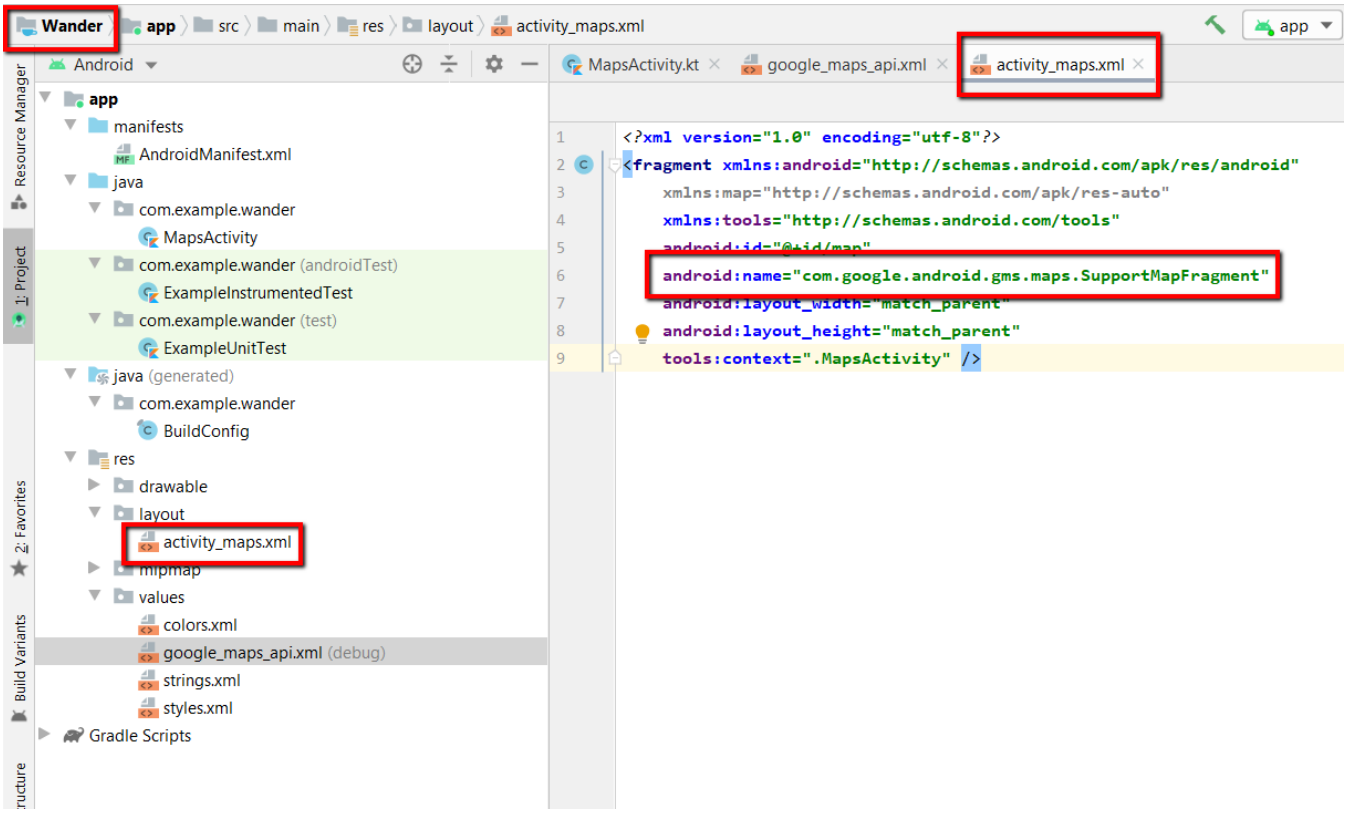
Now we will run the project in the emulator and see the output.



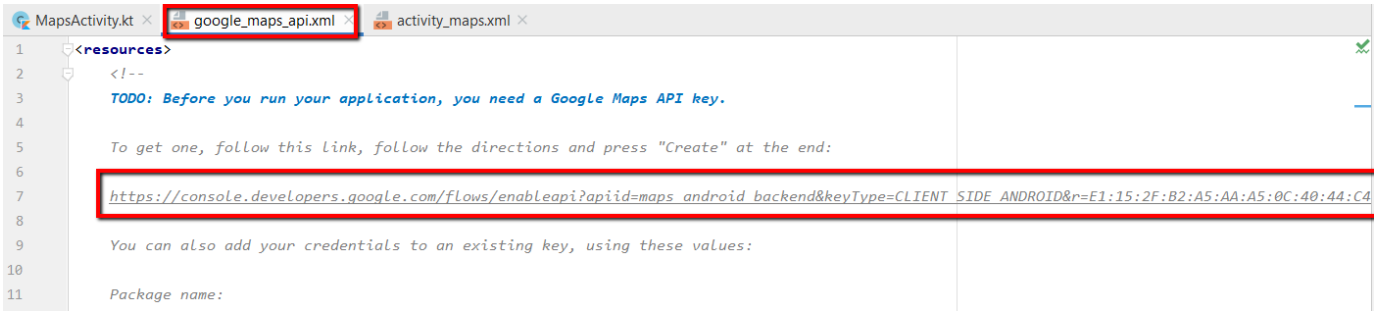
Google Maps

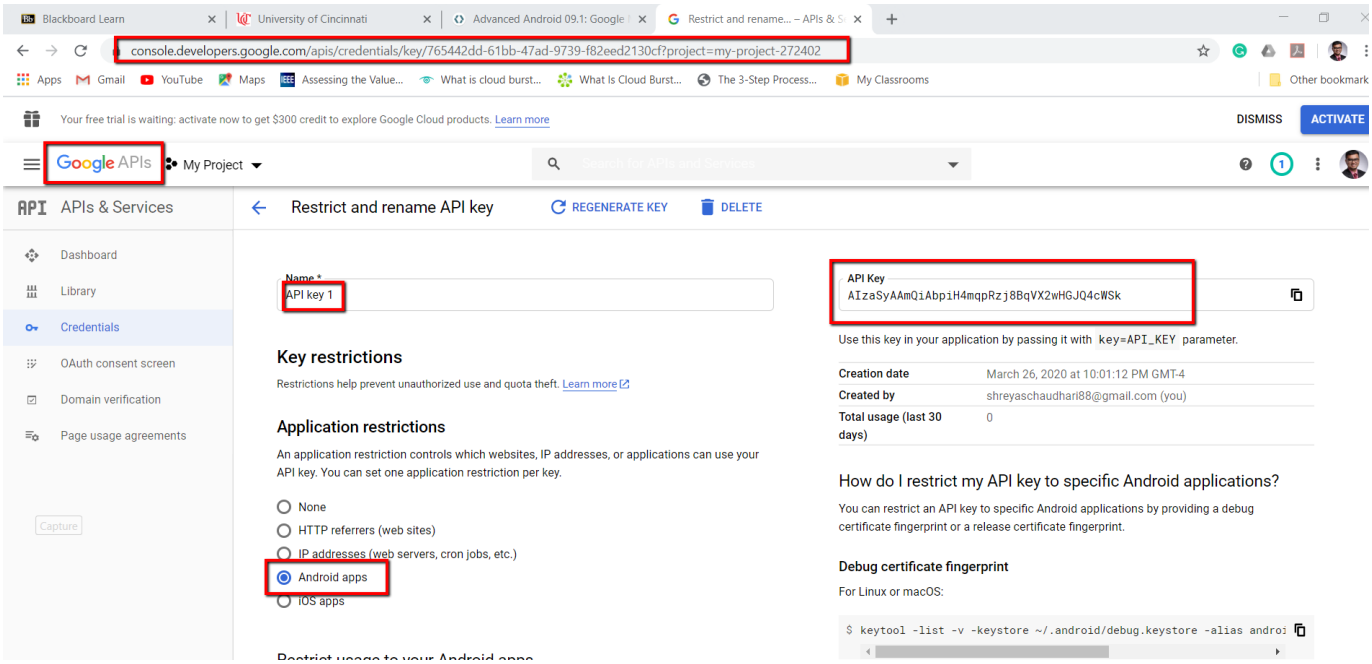
Building apps with Google Maps allows you to add features to your app such as satellite imagery, robust UI controls, location tracking, and location markers. You can add value to the standard Google Maps by showing information from your own data set, such as the locations of well-known fishing or climbing areas. You can also create games tied to the real world, like Pokémon Go.

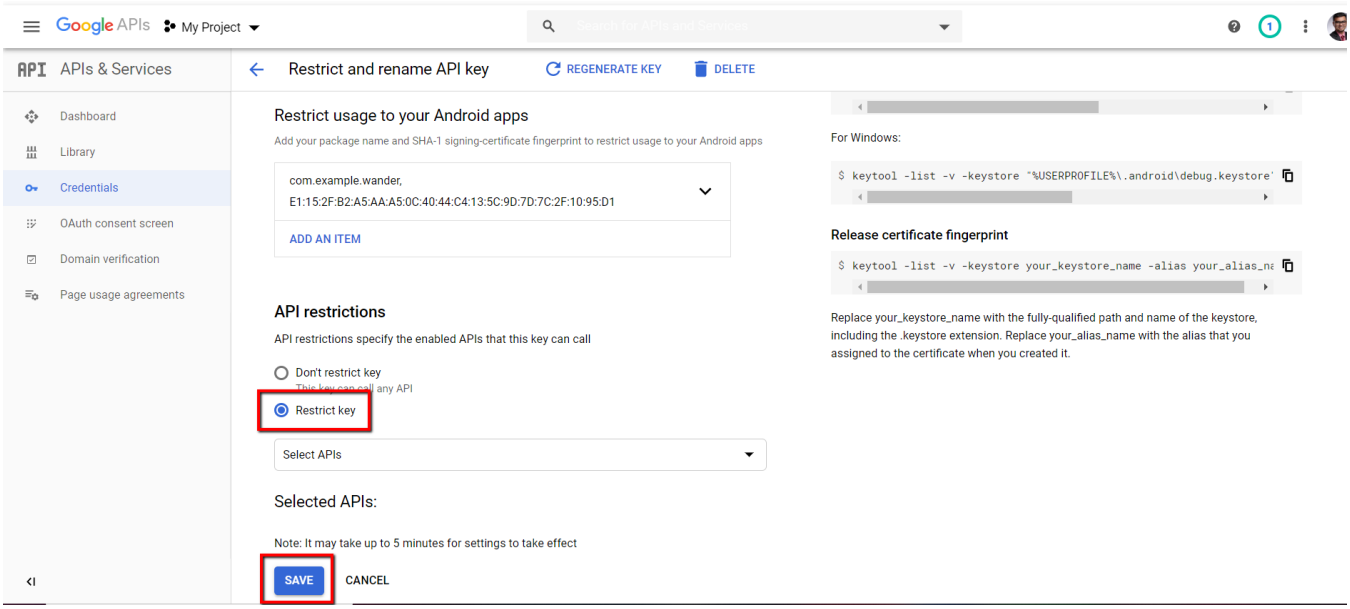
Set up the project and get an API key.



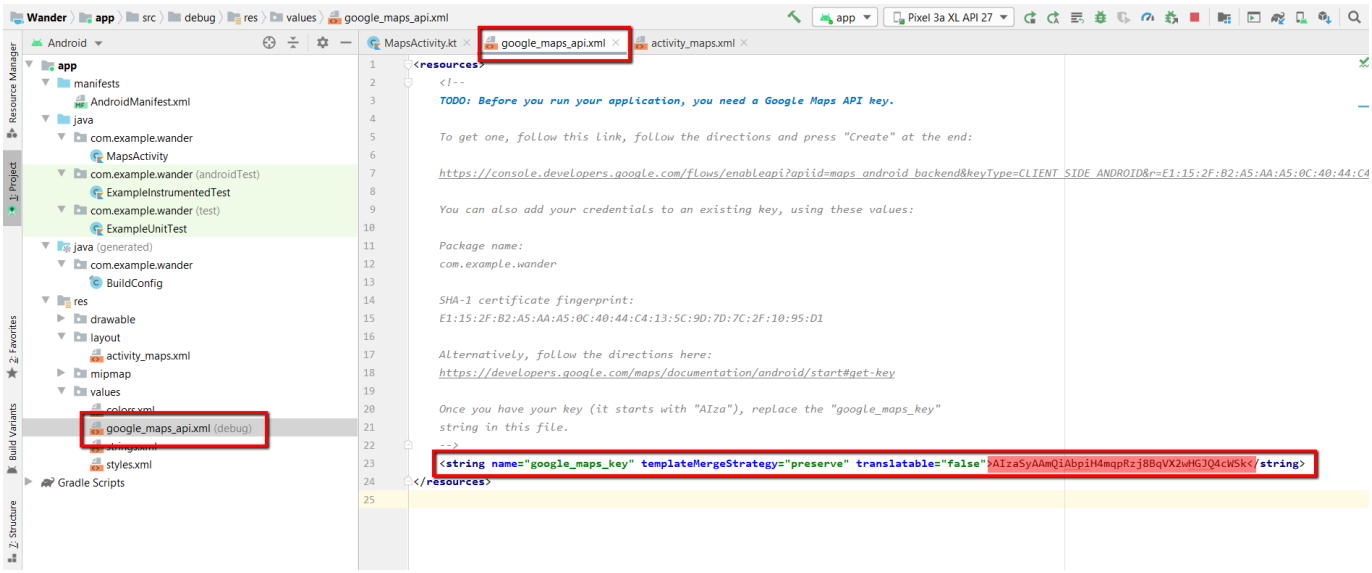
Then to obtain the API key go to 'google_maps_api.xml' file and in that file there will be link which is commented just copy pest that URL to the browser, you will be redirected to the Google Maps API key page. Then follow the instructions and obtain an API key from Google for Maps.



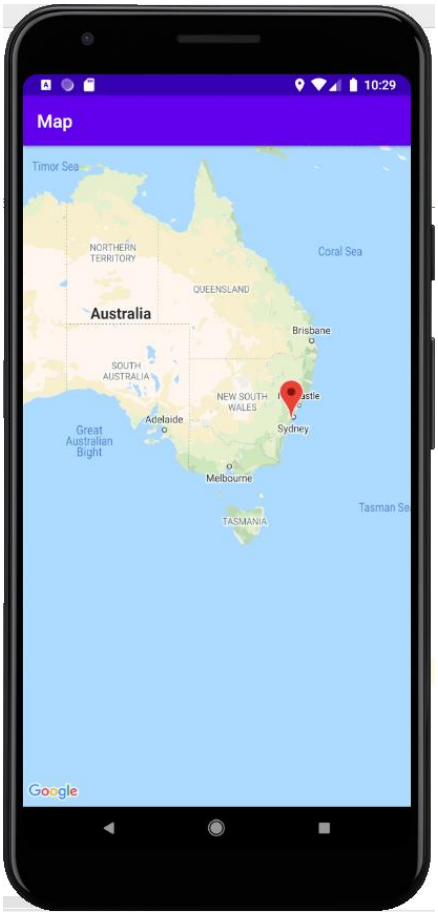




Then after obtaining the key copy that API key and paste it in the file ‘google_maps_api.xml’ file in the string name command where it is written “Your_Key_Here”.

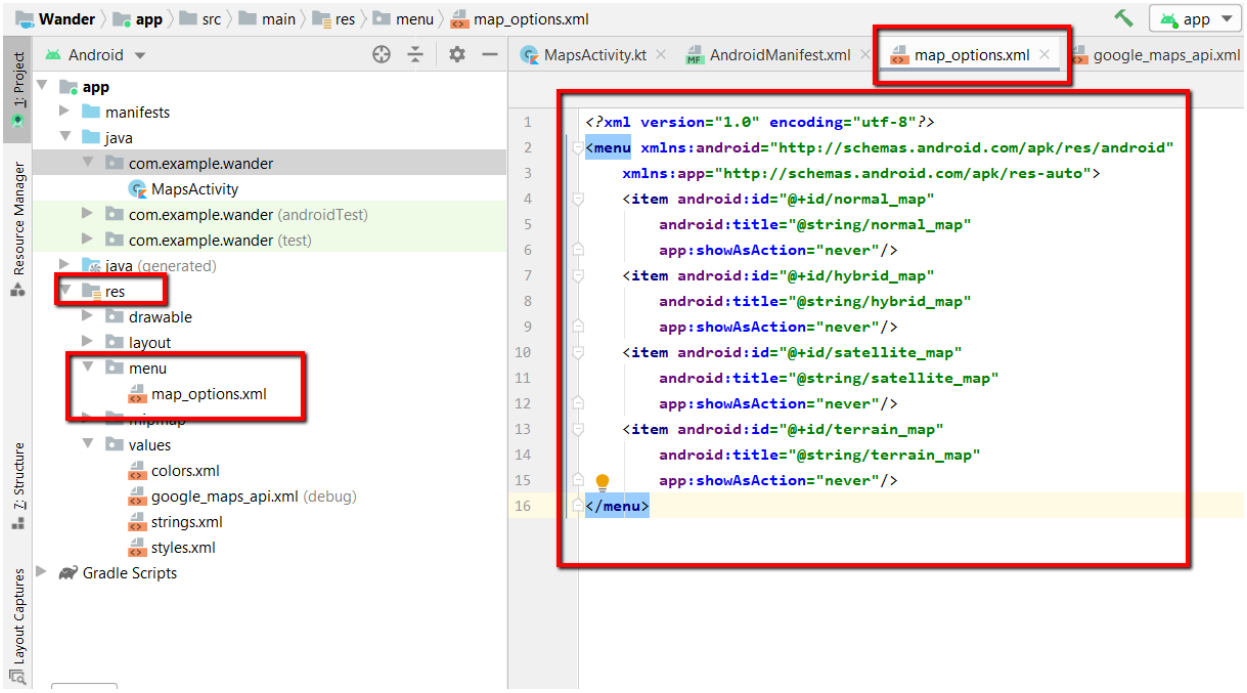


Now when you run the app in the emulator You have an embedded map in your activity, with a marker set in Sydney, Australia. (The Sydney marker is part of the template, and you change it later.)

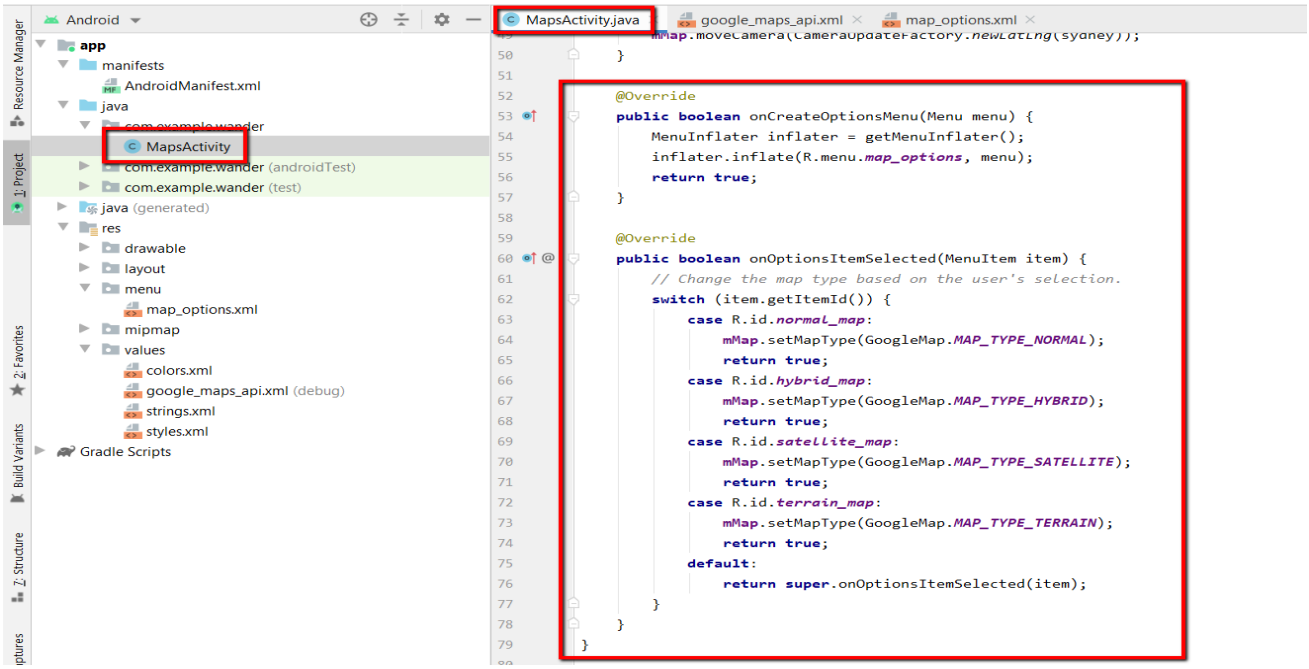


Add Map types and makers.

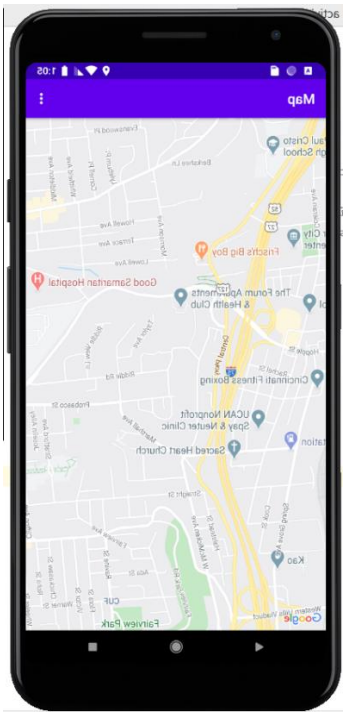
For adding new map types, we will create new XML file by right-clicking on ‘res’ directory then select ‘New’ -> ‘Android Resource File’. We will name the file as ‘map_options.xml’ and add following code into it.



Then we will add the following code into the ‘MainActivity.java’ file.



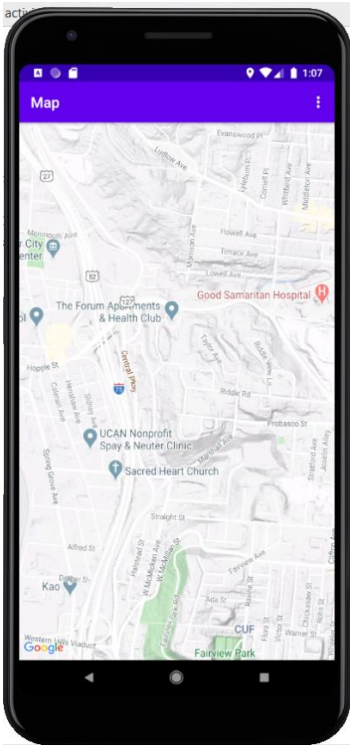
Now when we run the app, we will notice that there is a menu in the app bar to change the map type. We can see how the appearance changes.



Normal View

Hybrid View

Satellite View



Terrain View

Move the default Map location.

For making the default map location in Google Map application we will make the following changes in 'MapsActivity.kt'.

By default, the onMapReady() callback includes code that places a marker in Sydney, Australia, where Google Maps was created. The default callback also animates the map to pan to Sydney. In this step, we will make the map pan to our home location without placing a marker, then zoom to a level you specify.

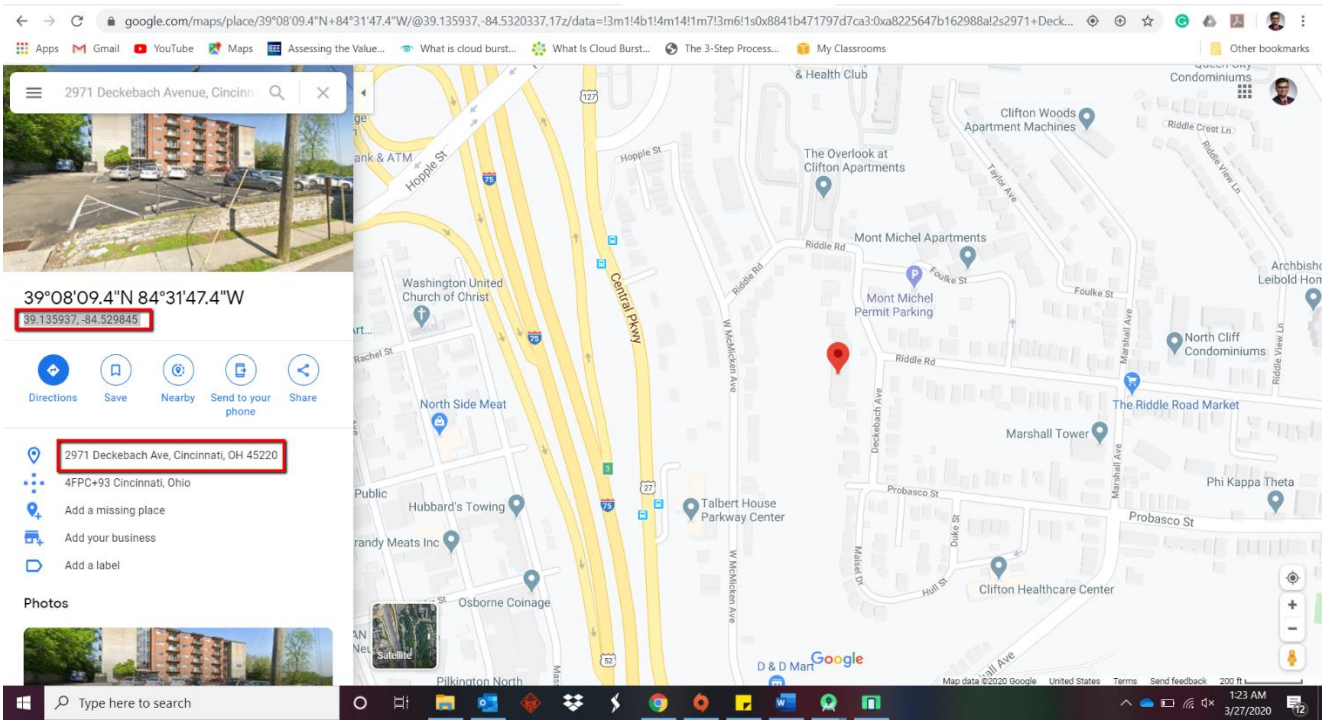
1. In the onMapReady() method, remove the code that places the marker in Sydney and moves the camera.
2. Go to **www.google.com/maps** in your browser and find your home.
3. Right-click on the location and select **What's here?**

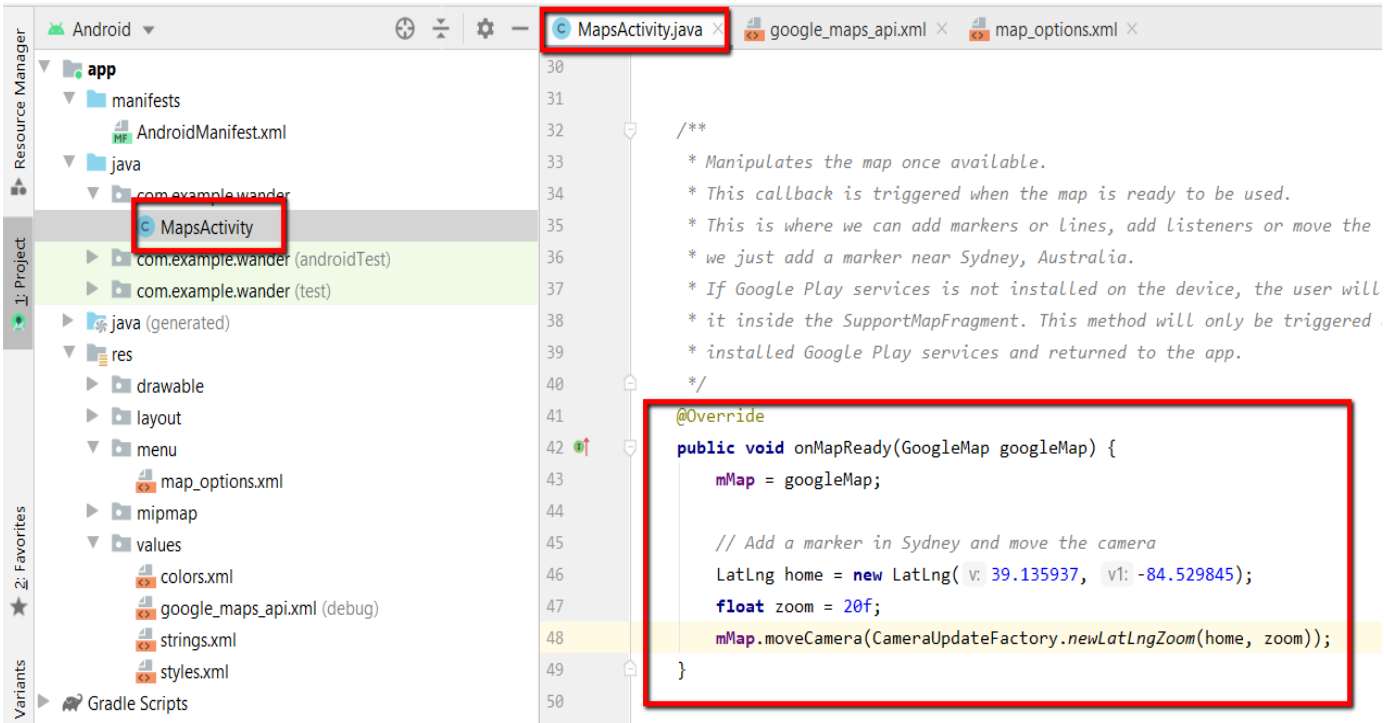
Near the bottom of the screen, a small window pops up with location information, including latitude and longitude.

4. Create a new LatLng object called home. In the LatLng object, use the coordinates you found from Google Maps in the browser.
5. Create a float variable called zoom and set the variable to your desired initial zoom level. The following list gives you an idea of what level of detail each level of zoom shows:

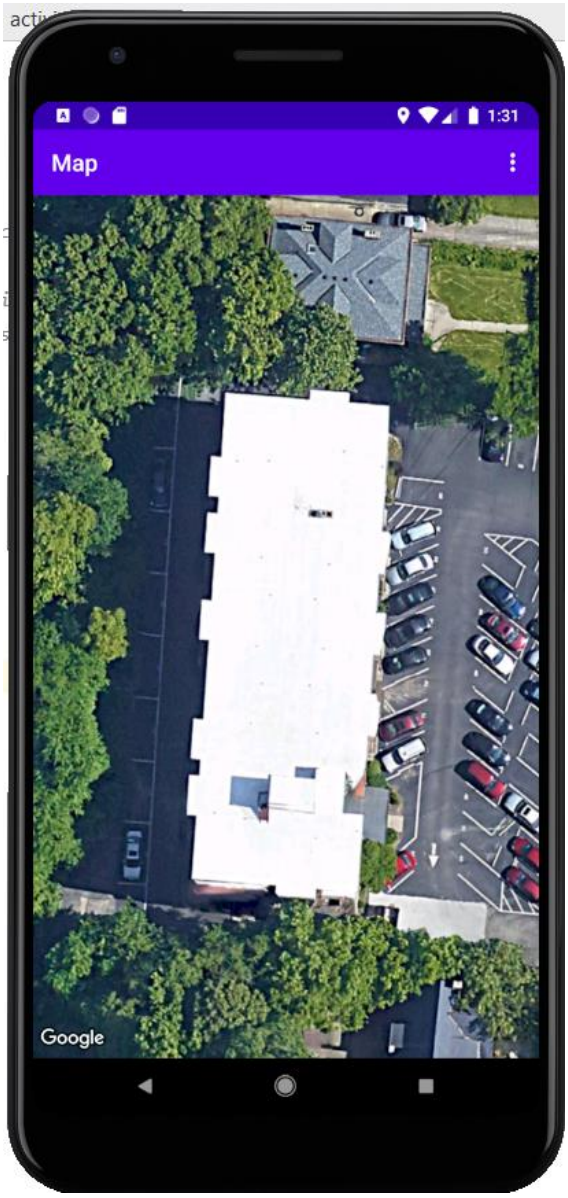
- 1: World
- 5: Landmass/continent
- 10: City
- 15: Streets
- 20: Buildings

6. Create a CameraUpdate object using CameraUpdateFactory_newLatlngZoom(), passing in your LatLng object and zoom variable. Pan and zoom the camera by calling moveCamera() on the GoogleMap object, passing in the new CameraUpdate object.


2971 Deckebach Avenue, Cincinn
39°08'09.4"N 84°31'47.4"W
39.135937, -84.529845
Directions
Save
Nearby
Send to your phone
Share
2971 Deckebach Ave, Cincinnati, OH 45220
4FPC+93 Cincinnati, Ohio
Add a missing place
Add your business
Add a label
Photos
Type here to search
Hopple St
Central Pkwy
Riddle Rd
Foulke St
Probasco St
Clifton Woods Apartment Machines
The Overlook at Clifton Apartments
Mont Michel Apartments
Mont Michel Permit Parking
Washington United Church of Christ
North Side Meat
Hubbard's Towing
Talbert House Parkway Center
Marshall Tower
The Riddle Road Market
North Cliff Condominiums
Phi Kappa Theta
Clifton Healthcare Center
Osborne Coinage
D & D Mart
Google
1:23 AM
3/27/2020

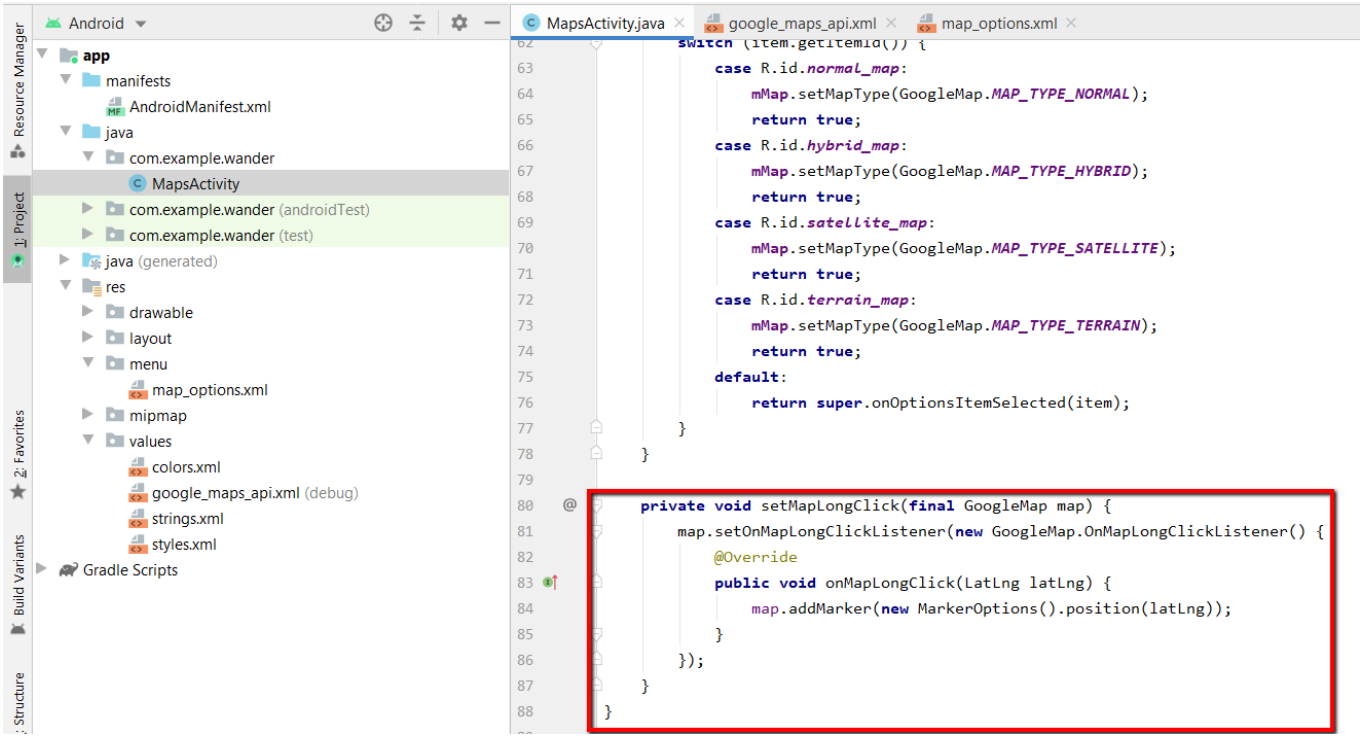

Android
MapsActivity.java
google_maps_api.xml
map_options.xml
app
manifests
AndroidManifest.xml
java
com.example.wander
MapsActivity
com.example.wander (androidTest)
com.example.wander (test)
java (generated)
res
drawable
layout
menu
map_options.xml
mipmap
values
colors.xml
google_maps_api.xml (debug)
strings.xml
styles.xml
Gradle Scripts
Resource Manager
1: Project
2: Favorites
Variants
/**
* Manipulates the map once available.
* This callback is triggered when the map is ready to be used.
* This is where we can add markers or lines, add listeners or move the
* we just add a marker near Sydney, Australia.
* If Google Play services is not installed on the device, the user will
* it inside the SupportMapFragment. This method will only be triggered
* installed Google Play services and returned to the app.
*/
@Override
public void onMapReady(GoogleMap googleMap) {
mMap = googleMap;
// Add a marker in Sydney and move the camera
LatLng home = new LatLng(v: 39.135937, v1: -84.529845);
float zoom = 20f;
mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(home, zoom));
}

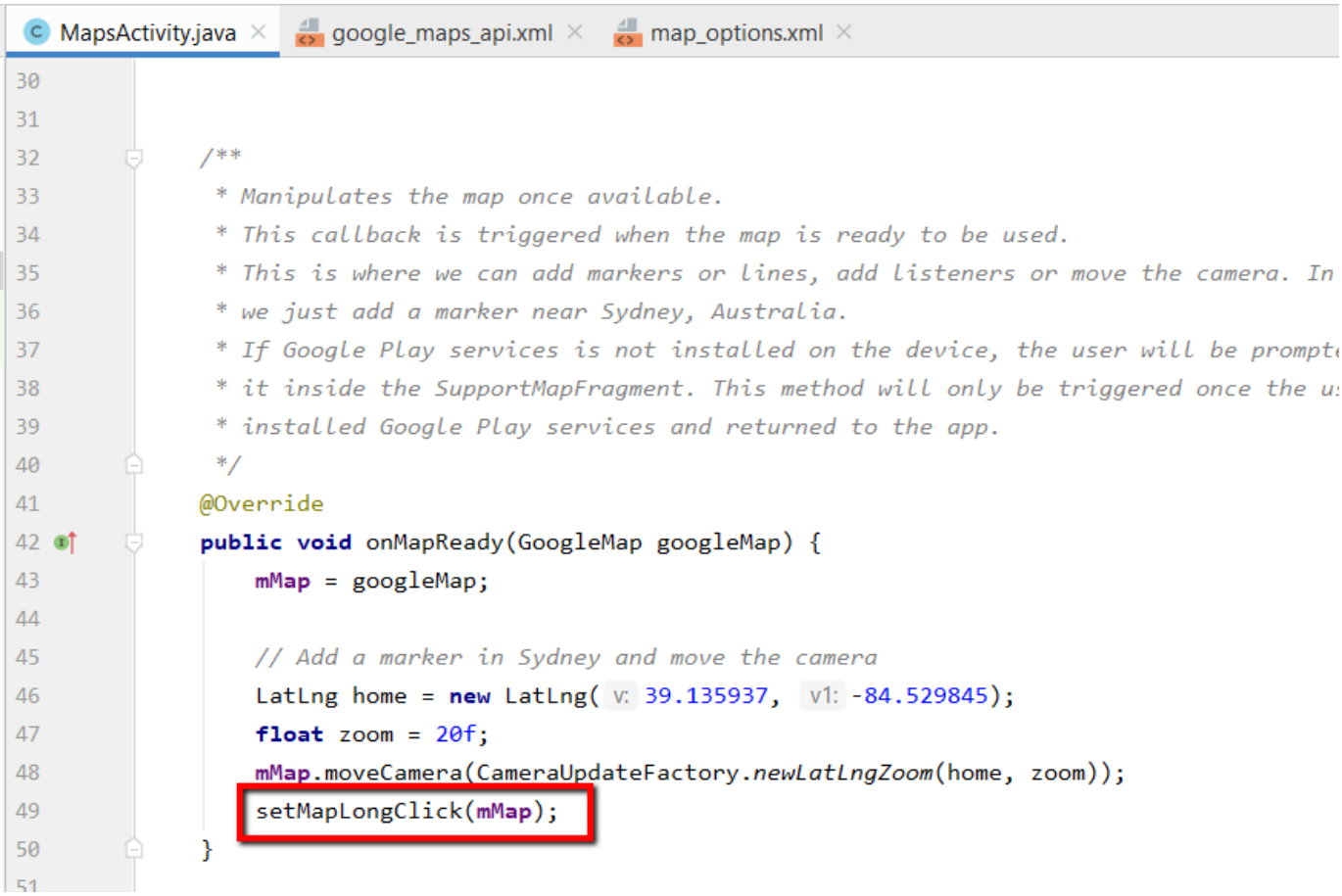
Now when you run the app, the map should pan to your home and zoom to your desired location. I am view the map in ‘Satellite View’.

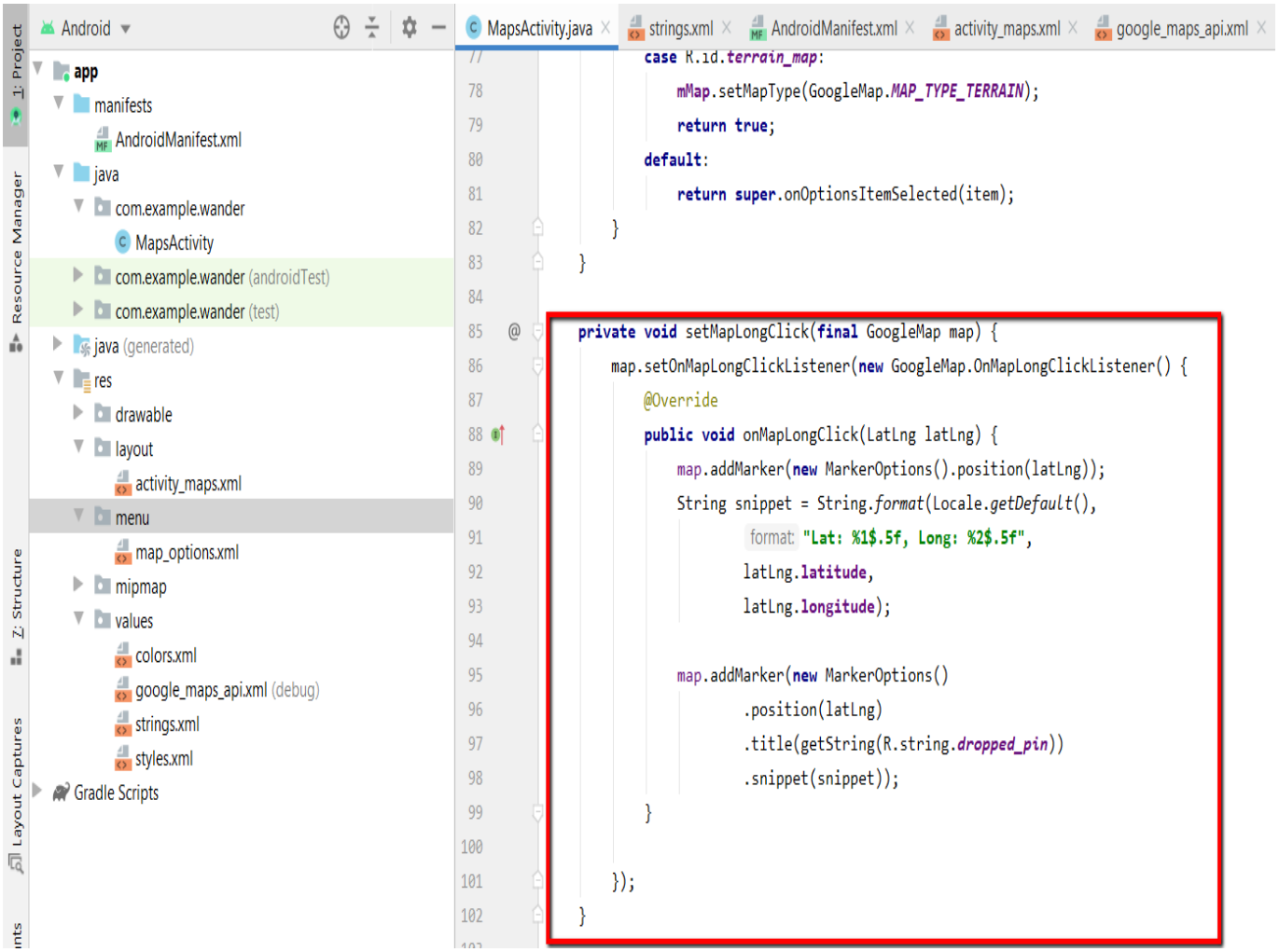


Add map maker.

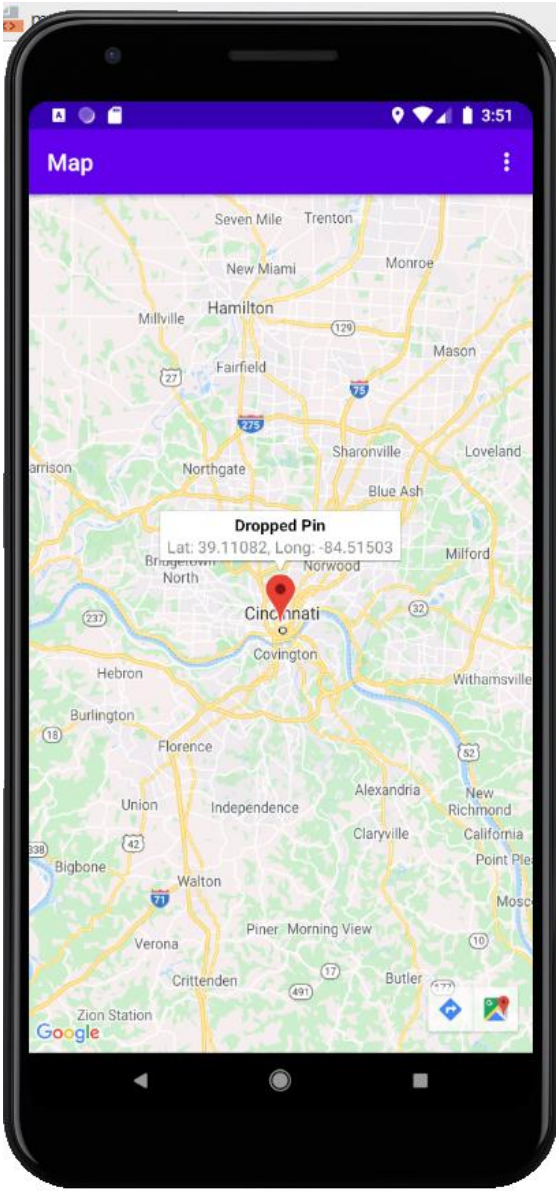
In this step, you add a marker when the user touches and holds a location on the map. You then add an InfoWindow that displays the coordinates of the marker when the marker is tapped. For this we will add the following code into the ‘MainActivity.java’ file.


Android
app
manifests
AndroidManifest.xml
java
com.example.wander
MapsActivity
com.example.wander (androidTest)
com.example.wander (test)
java (generated)
res
drawable
layout
menu
map_options.xml
mipmap
values
colors.xml
google_maps_api.xml (debug)
strings.xml
styles.xml
Gradle Scripts
MapsActivity.java
google_maps_api.xml
map_options.xml
switch (item.getItemId()) {
case R.id.normal_map:
mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
return true;
case R.id.hybrid_map:
mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
return true;
case R.id.satellite_map:
mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
return true;
case R.id.terrain_map:
mMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
return true;
default:
return super.onOptionsItemSelected(item);
}
}
private void setMapLongClick(final GoogleMap map) {
map.setOnMapLongClickListener(new GoogleMap.OnMapLongClickListener() {
@Override
public void onMapLongClick(LatLng latLng) {
map.addMarker(new MarkerOptions().position(latLng));
}
});
}
}


MapsActivity.java
google_maps_api.xml
map_options.xml
/**
* Manipulates the map once available.
* This callback is triggered when the map is ready to be used.
* This is where we can add markers or lines, add listeners or move the camera. In
* we just add a marker near Sydney, Australia.
* If Google Play services is not installed on the device, the user will be prompte
* it inside the SupportMapFragment. This method will only be triggered once the u
* installed Google Play services and returned to the app.
*/
@Override
public void onMapReady(GoogleMap googleMap) {
mMap = googleMap;
// Add a marker in Sydney and move the camera
LatLng home = new LatLng(v: 39.135937, v1: -84.529845);
float zoom = 20f;
mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(home, zoom));
setMapLongClick(mMap);
}



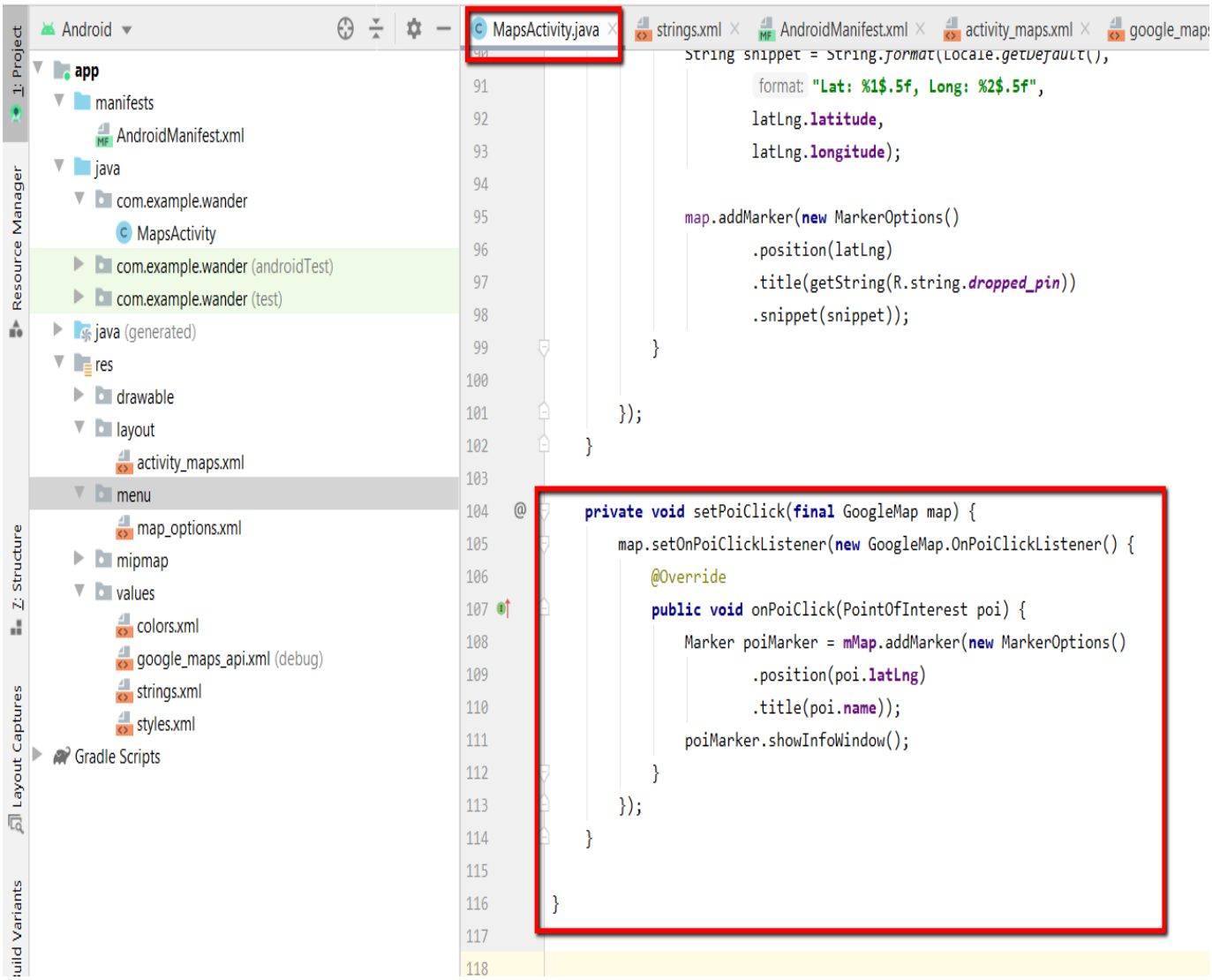
Now when you run the app. Touch and hold on the map to drop a location marker. Tap the marker to show the info window.



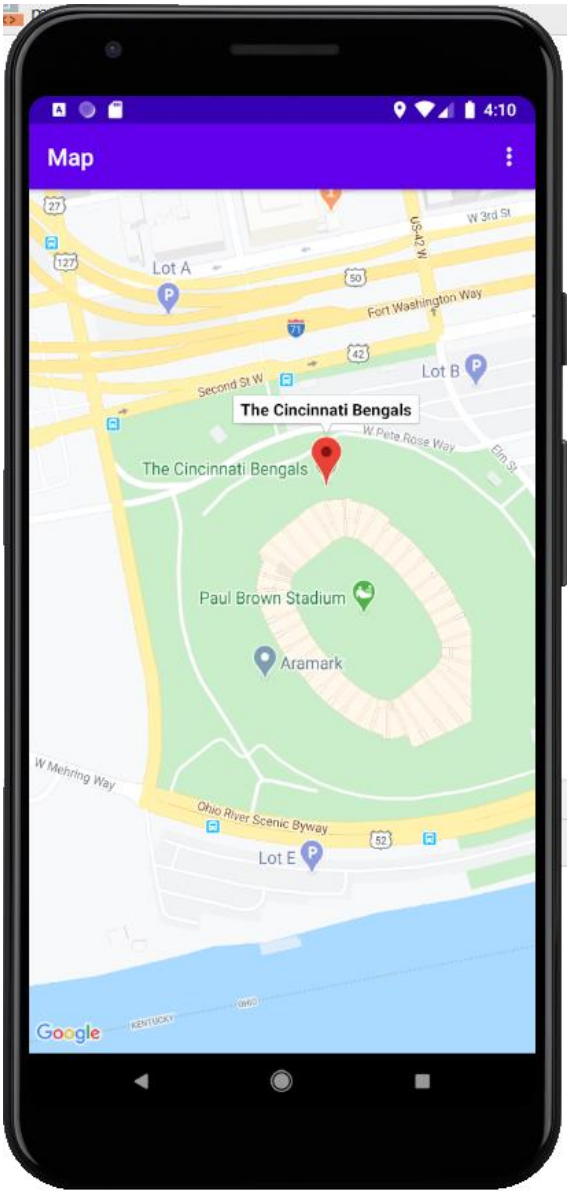
Adding POI (points of interest) listener:

By default, points of interest (POIs) appear on the map along with their corresponding icons. POIs include parks, schools, government buildings, and more. When the map type is set to normal, business POIs also appear on the map. Business POIs represent businesses such as shops, restaurants, and hotels.

To add POI listener we will add the following code in ‘MapsActivity.java’.


Android
app
manifests
AndroidManifest.xml
java
com.example.wander
MapsActivity
com.example.wander (androidTest)
com.example.wander (test)
java (generated)
res
drawable
layout
activity_maps.xml
menu
map_options.xml
mipmap
values
colors.xml
google_maps_api.xml (debug)
strings.xml
styles.xml
Gradle Scripts
MapsActivity.java
strings.xml
AndroidManifest.xml
activity_maps.xml
google_maps
String snippet = String.format(Locale.getDefault(),
format: "Lat: %1$.5f, Long: %2$.5f",
latLng.latitude,
latLng.longitude);
map.addMarker(new MarkerOptions()
.position(latLng)
.title(getString(R.string.dropped_pin))
.snippet(snippet));
}
});
}
private void setPoiClick(final GoogleMap map) {
map.setOnPoiClickListener(new GoogleMap.OnPoiClickListener() {
@Override
public void onPoiClick(PointOfInterest poi) {
Marker poiMarker = mMap.addMarker(new MarkerOptions()
.position(poi.latLng)
.title(poi.name));
poiMarker.showInfoWindow();
}
});
}
}

Now when we run the application and find a POI such as a stadium. Tap on the POI to place a marker on it and display the POI's name in an info window.



Style your map

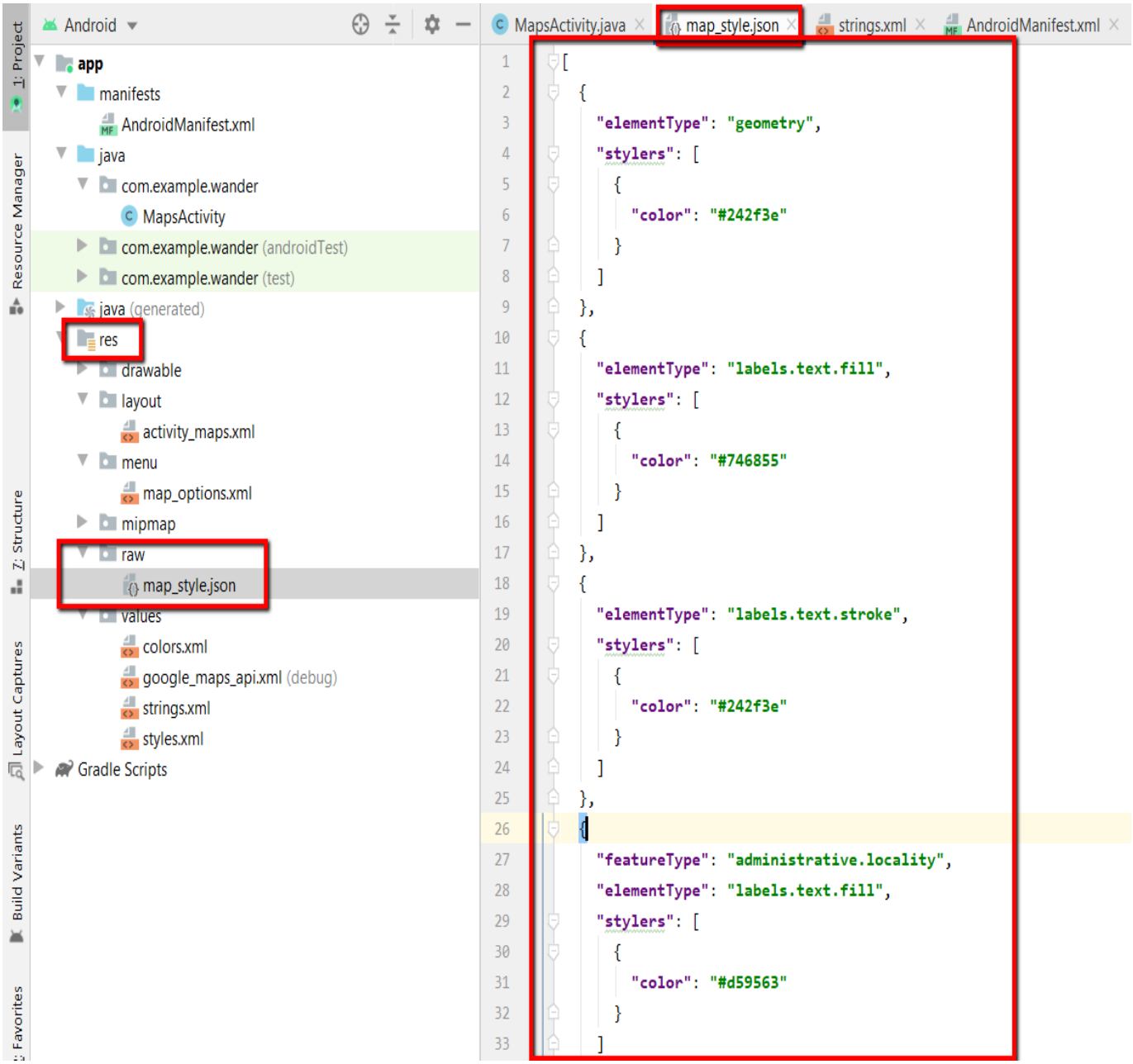
You can customize Google Maps in many ways, giving your map a unique look and feel.

Add a style to your map.

To create a customized style for your map, you generate a JSON file that specifies how features in the map are displayed. You don't have to create this JSON file manually: Google provides the Style Wizard, which generates the JSON for you after you visually style your map. In this practical, you style the map for "night mode," meaning that the map uses dim colors and low contrast for use at night.

1. Navigate to https://mapstyle.withgoogle.com/ in your browser.
2. Select **Create a Style**.
3. Select the **Night** theme.
4. Click **More Options** at the bottom of the menu.
5. At the bottom of the **Feature type** list, select **Water > Fill**. Change the color of the water to a dark blue (for example, #160064).
6. Click **Finish**. Copy the JSON code from the resulting pop-up window.
7. In Android Studio, create a resource directory called raw in the res directory. Create a file in res/raw called map_style.jason.
8. Paste the JSON code into the new resource file.
9. To set the JSON style to the map, call setMapStyle() on the GoogleMap object. Pass in a MapStyleOptions object, which loads the JSON file. The method return setMapSTyle() is a boolean indicating the success of the styling. If the file can't be loaded, the method throws a Resource.NotFoundException.

Now we will copy the following “json” code which we got in the Google Map Style in a file which we made in ‘res’ -> ‘raw’ -> ‘map_style.json’.


Android
app
manifests
AndroidManifest.xml
java
com.example.wander
MapsActivity
com.example.wander (androidTest)
com.example.wander (test)
java (generated)
res
drawable
layout
activity_maps.xml
menu
map_options.xml
mipmap
raw
map_style.json
values
colors.xml
google_maps_api.xml (debug)
strings.xml
styles.xml
Gradle Scripts
MapsActivity.java
map_style.json
strings.xml
AndroidManifest.xml
[
{
"elementType": "geometry",
"stylers": [
{
"color": "#242f3e"
}
]
},
{
"elementType": "labels.text.fill",
"stylers": [
{
"color": "#746855"
}
]
},
{
"elementType": "labels.text.stroke",
"stylers": [
{
"color": "#242f3e"
}
]
},
{
"featureType": "administrative.locality",
"elementType": "labels.text.fill",
"stylers": [
{
"color": "#d59563"
}
]

MapsActivity.java | map_style.json | strings.xml | AndroidManifest.xml

```
43    },
44    {
45      "featureType": "poi.park",
46      "elementType": "geometry",
47      "stylers": [
48        {
49          "color": "#263c3f"
50        }
51      ]
52    },
53    {
54      "featureType": "poi.park",
55      "elementType": "labels.text.fill",
56      "stylers": [
57        {
58          "color": "#6b9a76"
59        }
60      ]
61    },
62    {
63      "featureType": "road",
64      "elementType": "geometry",
65      "stylers": [
66        {
67          "color": "#38414e"
68        }
69      ]
70    },
```

MapsActivity.java | map_style.json | strings.xml | AndroidManifes

```
71    {
72      "featureType": "road",
73      "elementType": "geometry.stroke",
74      "stylers": [
75        {
76          "color": "#212a37"
77        }
78      ]
79    },
80    {
81      "featureType": "road",
82      "elementType": "labels.text.fill",
83      "stylers": [
84        {
85          "color": "#9ca5b3"
86        }
87      ]
88    },
89    {
90      "featureType": "road.highway",
91      "elementType": "geometry",
92      "stylers": [
93        {
94          "color": "#746855"
95        }
96      ]
97    },
98    {
99      "featureType": "road.highway",
100     "elementType": "geometry.stroke",
101     "stylers": [
102       {
103         "color": "#1f2835"
104       }
```

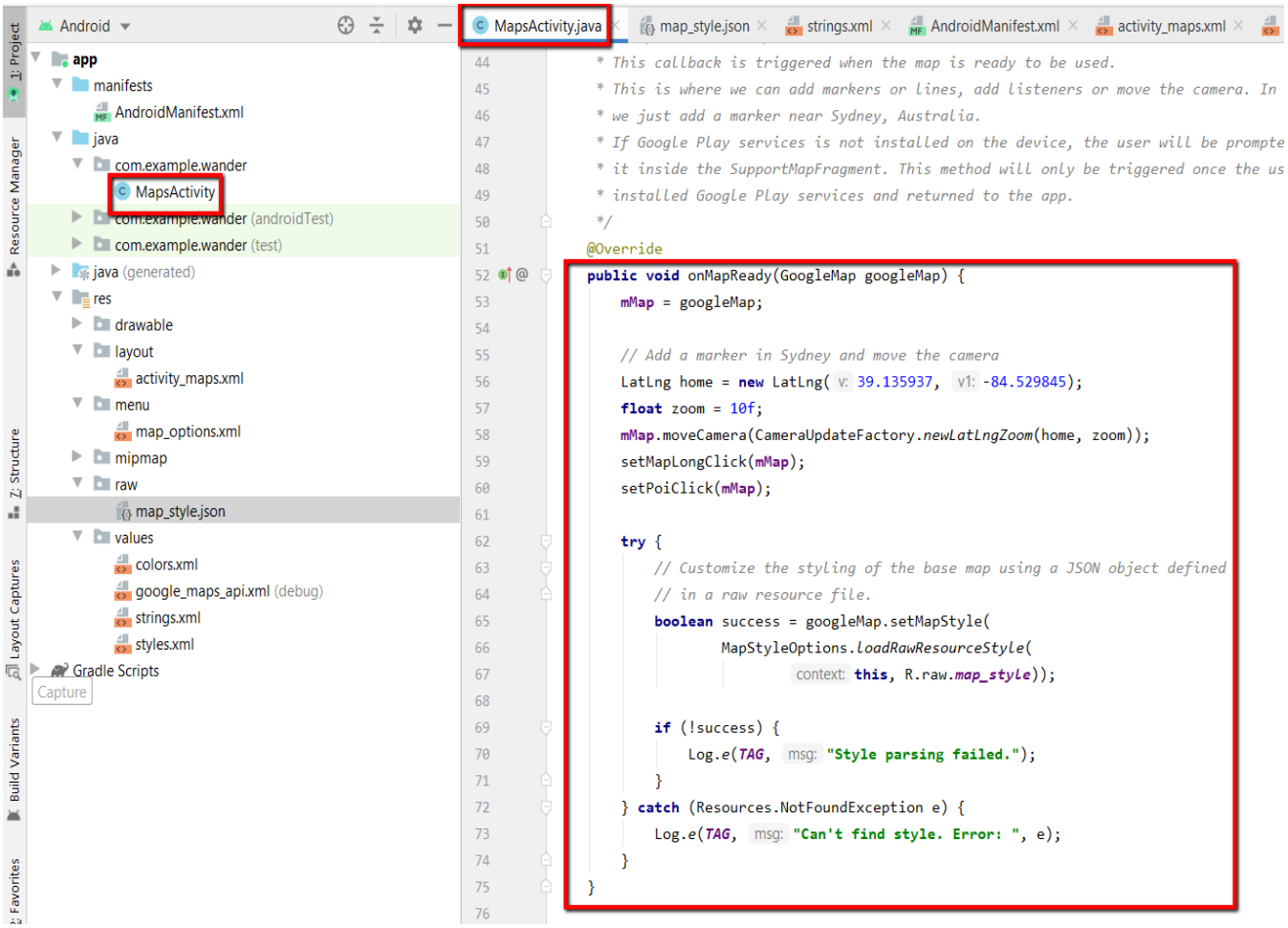
MapsActivity.java | map_style.json | strings.xml | AndroidManif

```
105       ]
106     },
107     {
108       "featureType": "road.highway",
109       "elementType": "labels.text.fill",
110       "stylers": [
111         {
112           "color": "#f3d19c"
113         }
114       ]
115     },
116     {
117       "featureType": "transit",
118       "elementType": "geometry",
119       "stylers": [
120         {
121           "color": "#2f3948"
122         }
123       ]
124     },
125     {
126       "featureType": "transit.station",
127       "elementType": "labels.text.fill",
128       "stylers": [
129         {
130           "color": "#d59563"
131         }
132       ]
133     },
134     {
135       "featureType": "water",
136       "elementType": "geometry",
137       "stylers": [
```

```
        {
            "color": "#17263c"
        }
    ]
  }
]
```

Copy the following code into the onMapReady() method to style the map. You may need to create a TAG string for your log statements:

We will copy the following code in the 'MapsActivity.java'.

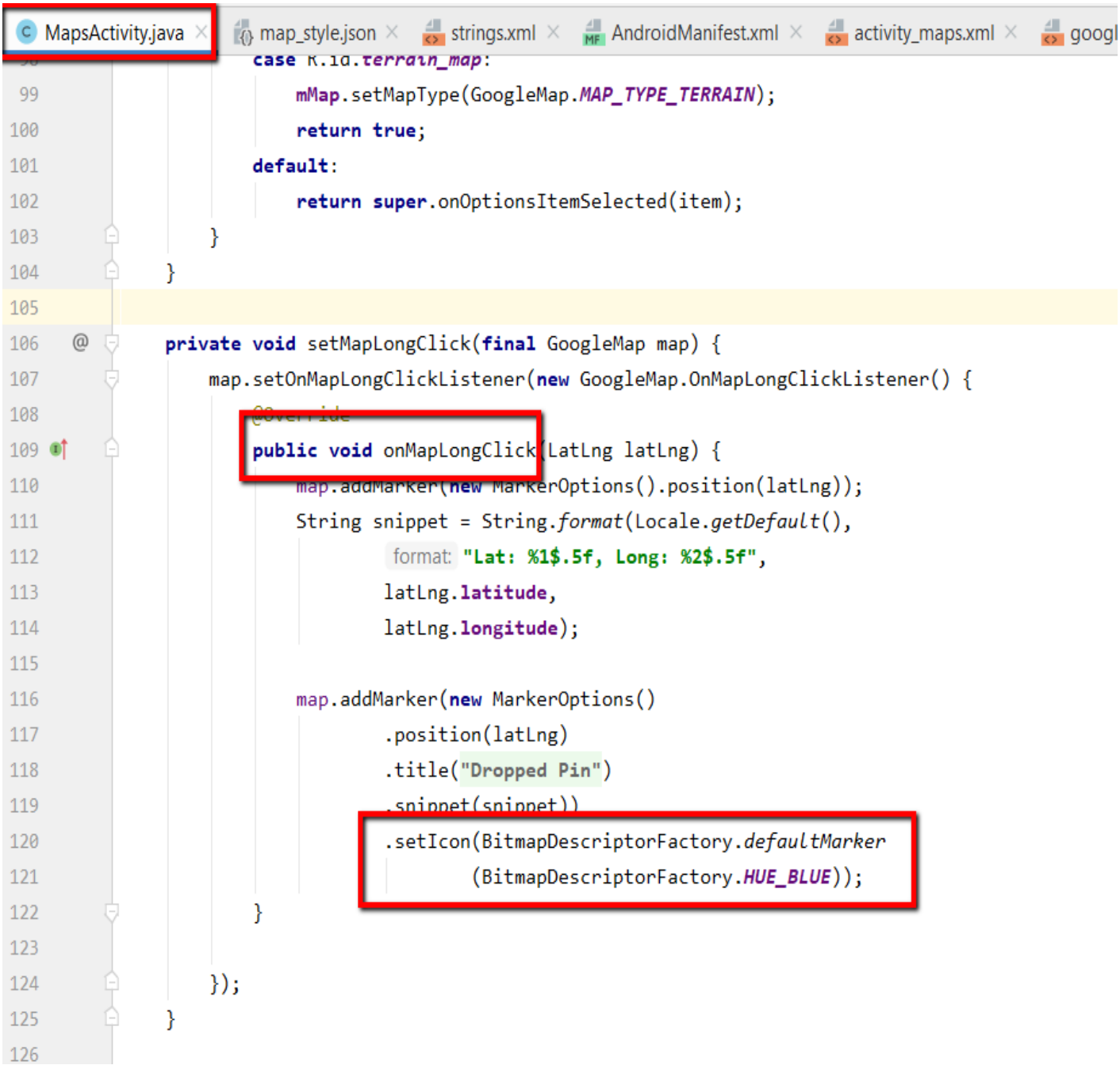


Now when we run the app the new styling should be visible when the map is in normal mode.

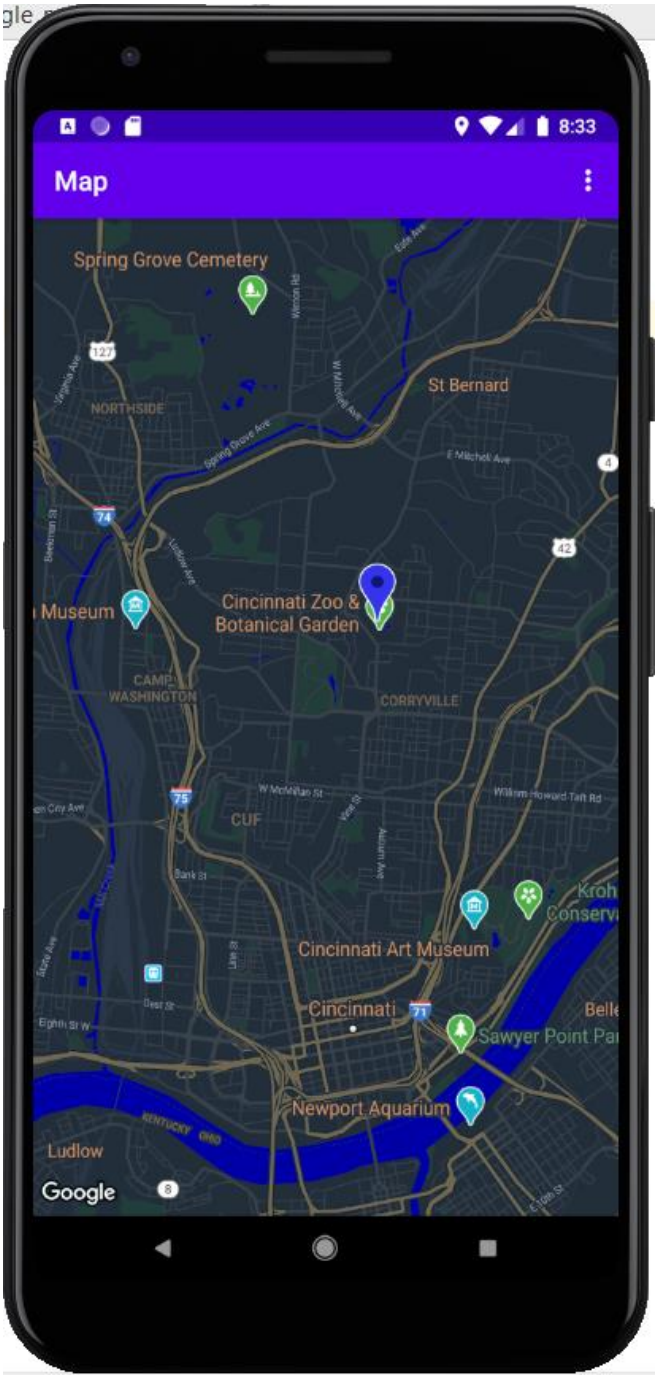
Style your marker.

We can personalize your map further by styling the map markers. In this step, we will change the default red markers to match the night mode color scheme.

For this we will add the following code into onMapLongClick() method which is 'MapsActivity.java' file.



Now when we Run the app. The markers you place are now shaded blue, which is more consistent with the night-mode theme of the app.



Note that POI markers are still red, because you didn't add styling to the onPoiClick() method.

Add an Overlay.

One way you can customize the Google Map is by drawing on top of it. This technique is useful if we want to highlight a particular type of location, such as popular fishing spots. Three types of overlays are supported:

- Shapes: we can add polylines, polygons, and circles to the map.
- TileOverlay objects: A tile overlay defines a set of images that are added on top of the base map tiles. Tile overlays are useful when you want to add extensive imagery to the map. A typical tile overlay covers a large geographical area.
- GroundOverlay objects: A ground overlay is an image that is fixed to a map. Unlike markers, ground overlays are oriented to the Earth's surface rather than to the screen. Rotating, tilting, or zooming the map changes the orientation of the image. Ground overlays are useful when you wish to fix a single image at one area on the map

In this step, we will add a ground overlay in the shape of an Android to our home location. For this we will download the image from the link provided in code lab and copy that image in ‘res’ -> ‘drawable’ folder. The we will add the following code in ‘MapsActivity.java’ file.

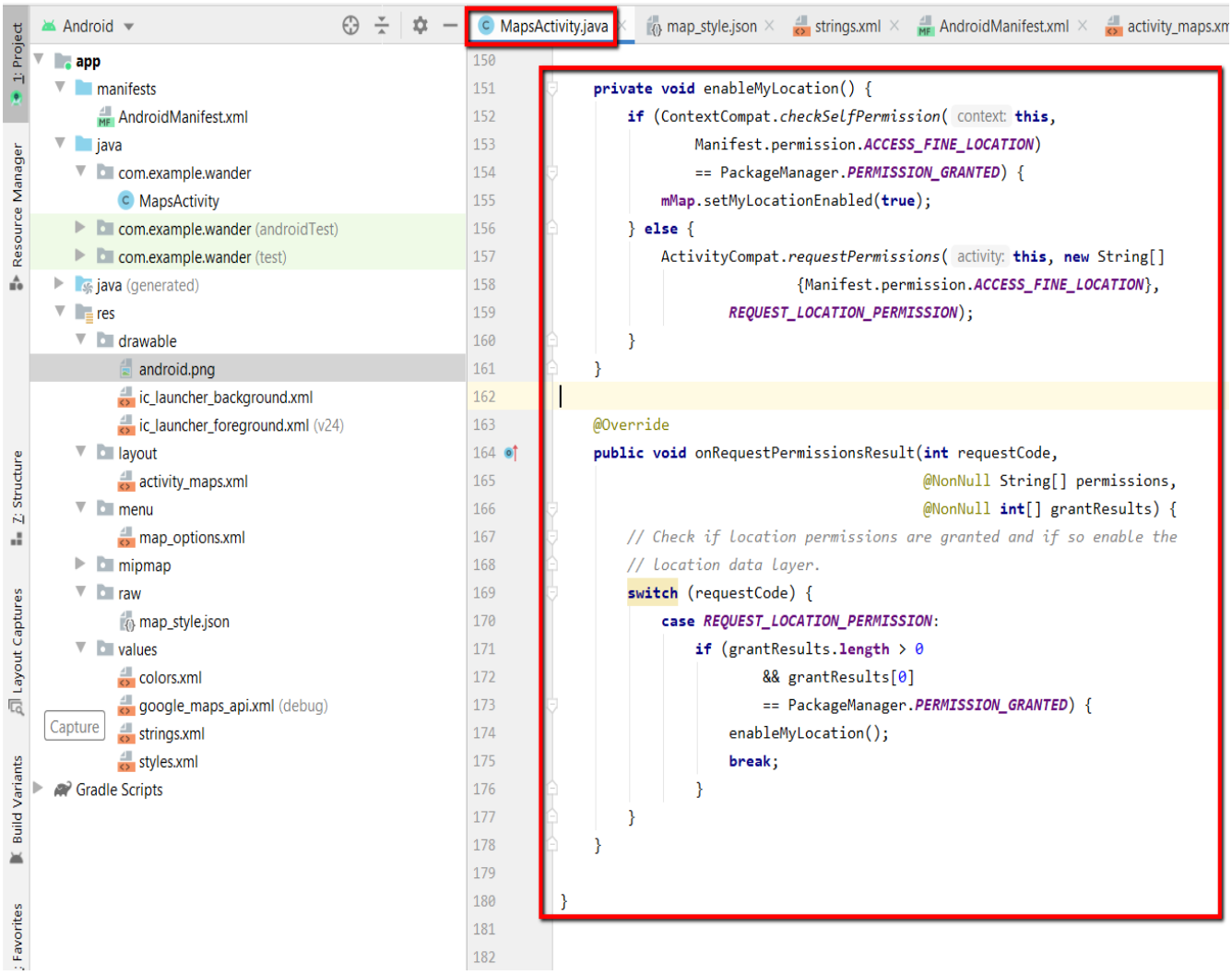
Now when we run the app and zoom in on our home location, and we will see the Android image as an overlay.

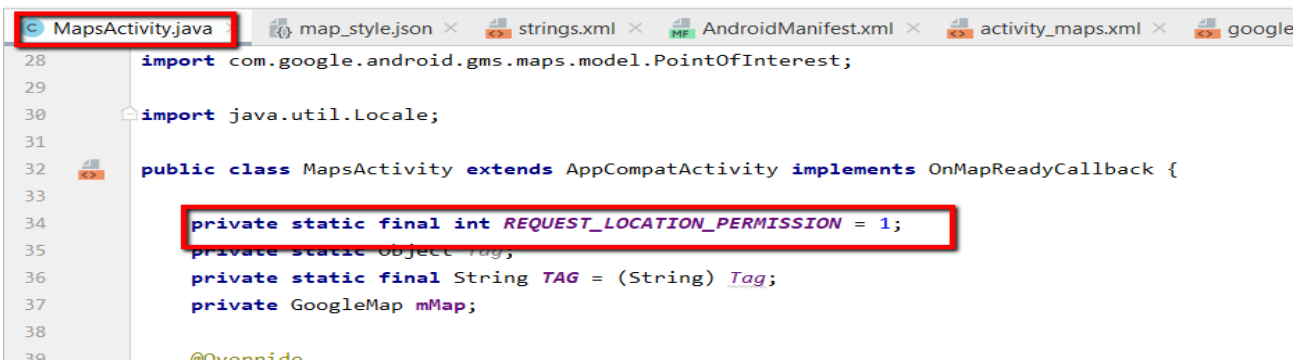
Enable location tracking and street view.

Enable location tracking.

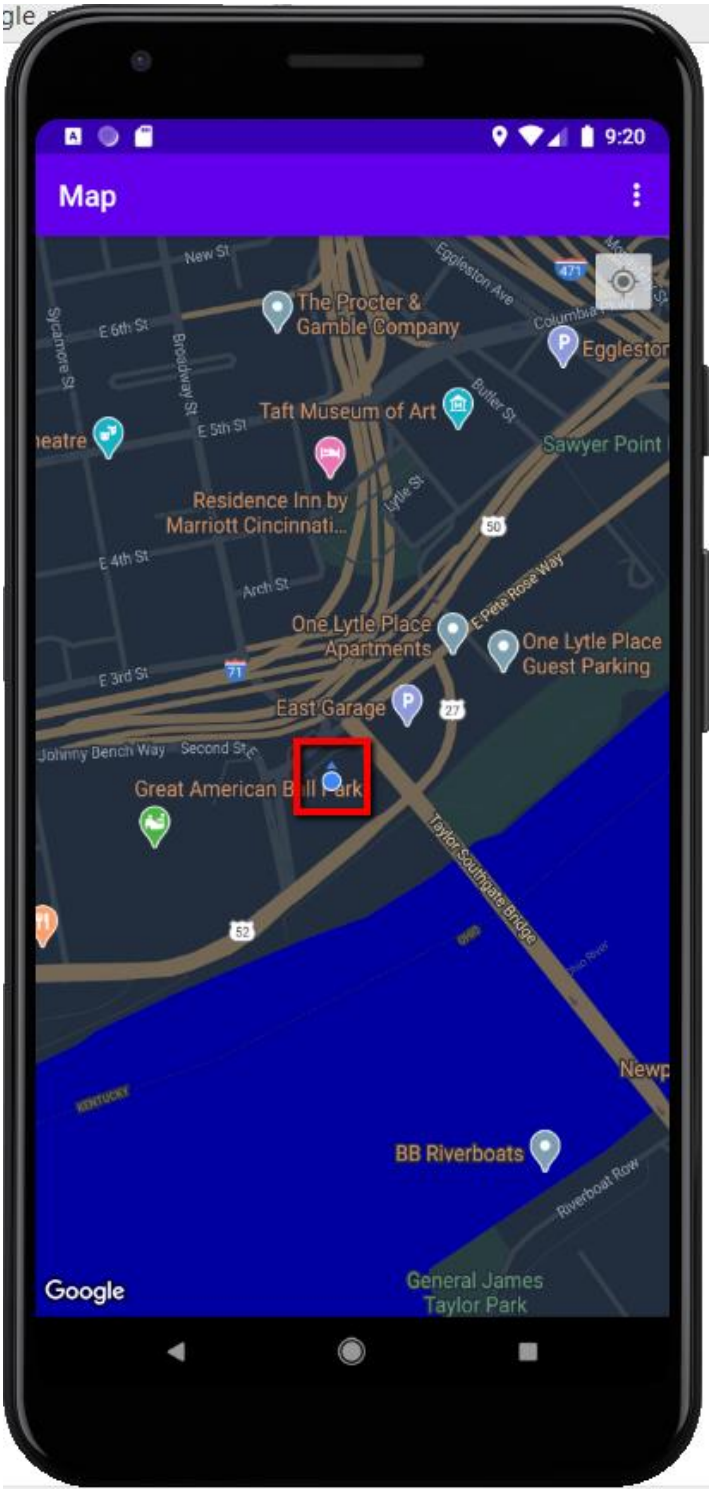
Enabling location tracking in Google Maps requires a single line of code. However, you must make sure that the user has granted location permissions (using the runtime-permission model).

We will add the following code in 'MapsActivity.java' file.





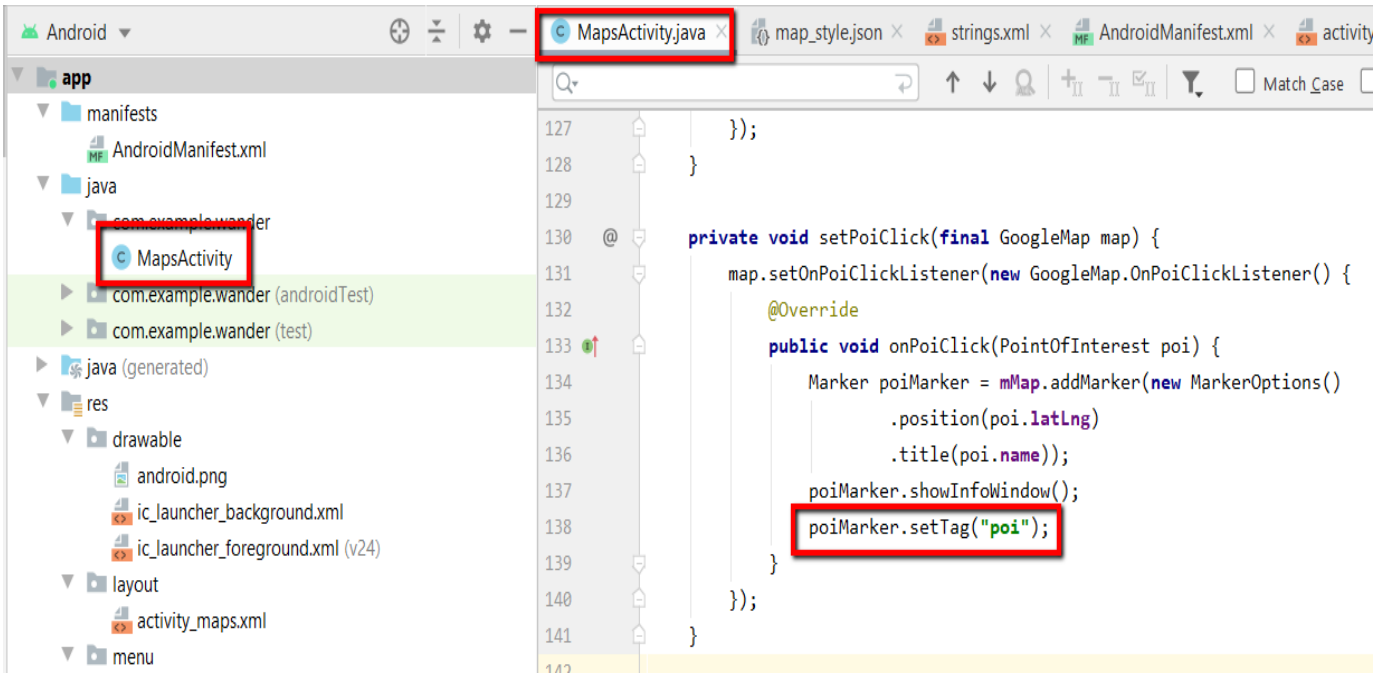
Now when we run the app in the top-right corner now contains the **My Location** button, which displays the device's current location.

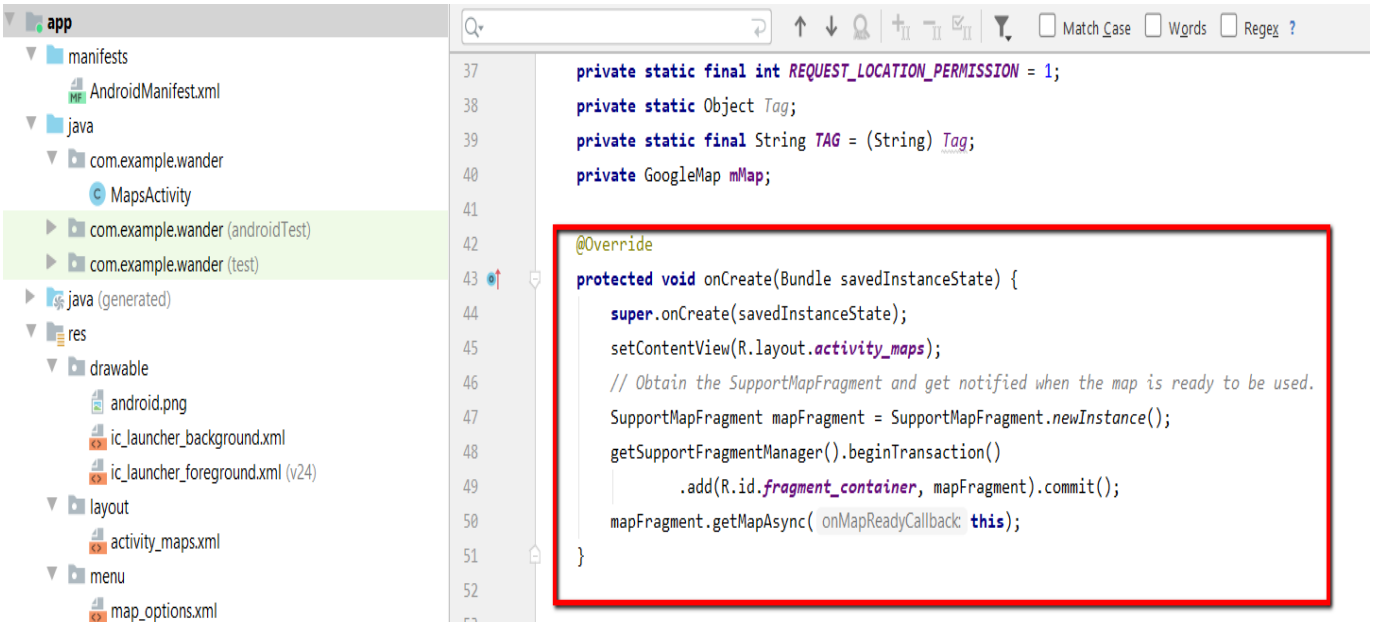


Enable Street View.

Google Maps provides Street View, which is a panoramic view of a location with controls for navigating along a designated path.

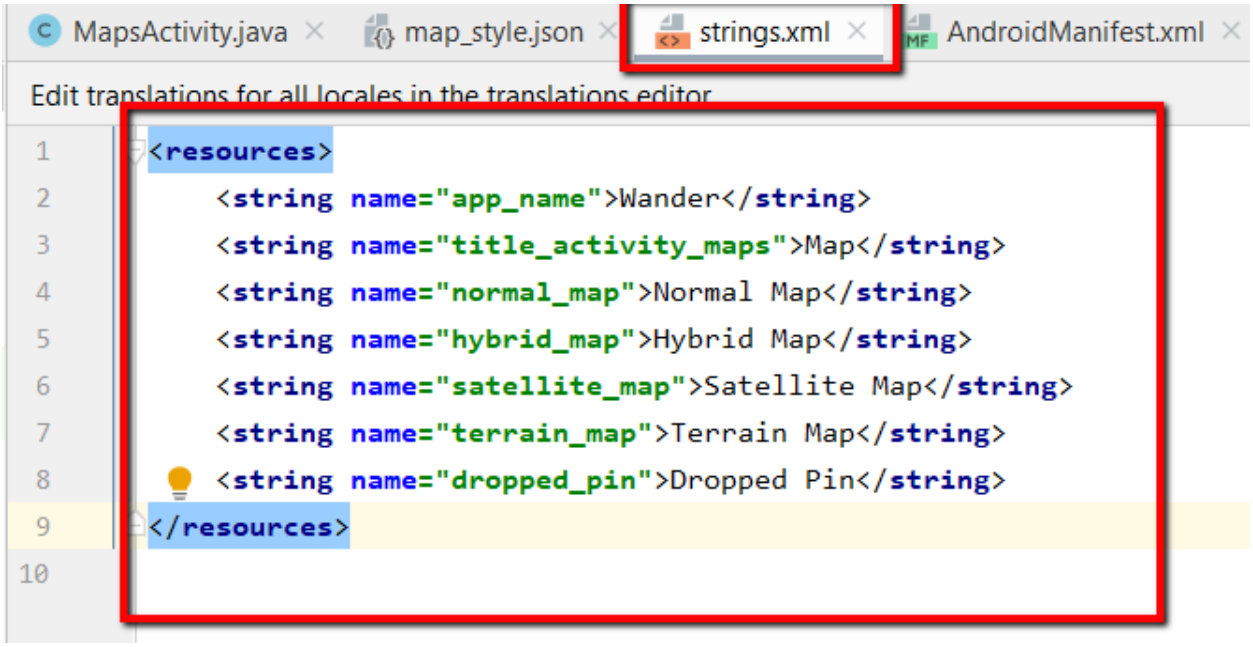
For this will add the following code into 'MapsActivity.java' file.



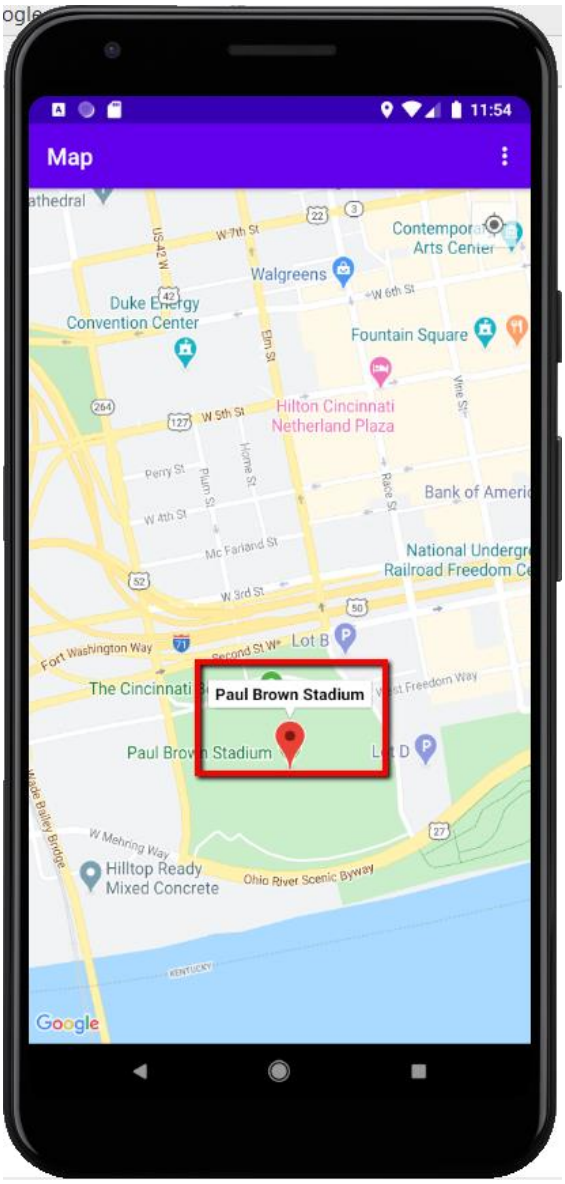


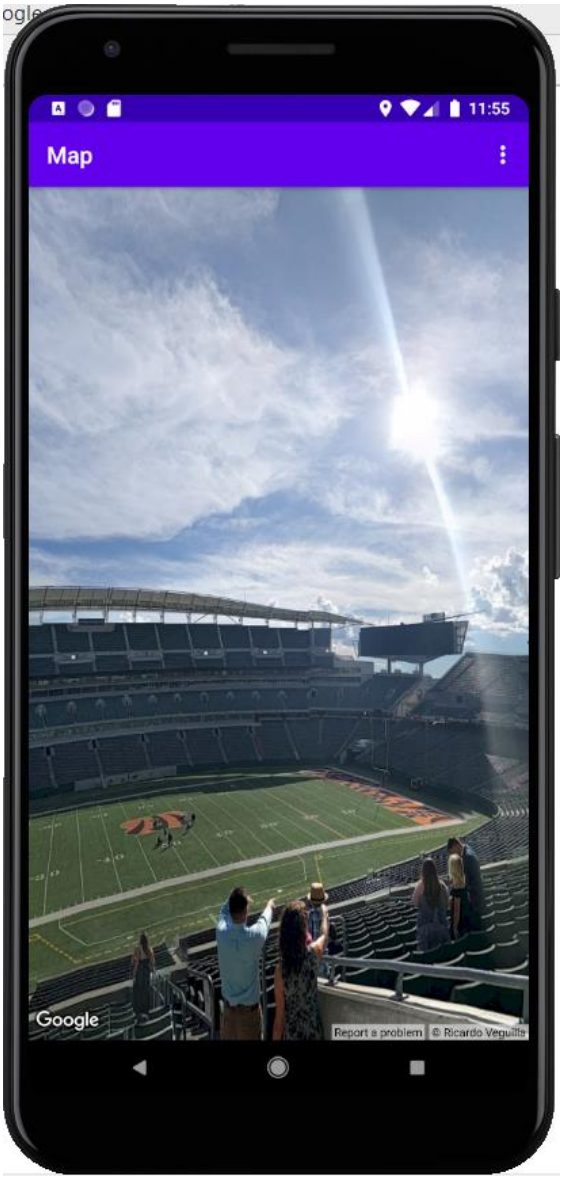
```
private void setInfoWindowClickToPanorama(GoogleMap map) {
    map.setOnInfoWindowClickListener(
            new GoogleMap.OnInfoWindowClickListener() {
                @Override
                public void onInfoWindowClick(Marker marker) {
                    if (marker.getTag() == "poi") {
                        StreetViewPanoramaOptions options =
                                new StreetViewPanoramaOptions().position(
                                        marker.getPosition());

                        SupportStreetViewPanoramaFragment streetViewFragment
                                = SupportStreetViewPanoramaFragment
                                .newInstance(options);

                        // Replace the fragment and add it to the backstack
                        getSupportFragmentManager().beginTransaction()
                                .replace(R.id.fragment_container,
                                        streetViewFragment)
                                .addToBackStack(null).commit();
                    }
                }
            });

}
```

Then we will add the following code in ‘strings.xml’ file.



Now when we Run the app and zoom into a city that has Street View Coverage such as Paul Brown Stadium, and find a POI, such as a Stadium. Tap on the POI to place a marker and show the info window. Tap the info window to enter Street View mode for the location of the marker. Press the back button to return to the map fragment.





So which app I can develop which will contain all this 4 operations in one?

With the knowledge of all the four hands-on, I can develop a Tourist app which can helpful for tourist as well as people who like to explore new places. There will be an in am compass which will be helpful to know the direction from the current spot, then their location locator which will be helpful to find the location where the user is currently staying. Also, there will a in app google map to navigate the desired location, as well as, the user can find the particular location information on the map for example what that particular place is known as, also the user can click on the info tab and find some photographic information regarding that place as well.