Ubiquitous Computing

COMP H4025

Lecturer: Simon McLoughlin

Lecture 6

Lecture Overview

This week:

- The Sensor Framework
- Sending SMS
- JSON parsing

Android Sensors

• Most Android-powered devices have **built-in sensors** that measure motion, orientation, and various environmental conditions.

• These sensors are capable of providing raw data with high precision and accuracy, and are useful if you want to monitor three-dimensional **device movement or positioning**, or you want to monitor changes in the **ambient environment** near a device.

• For example, a game might track readings from a device's gravity sensor to infer complex user gestures and motions, such as tilt, shake, rotation, or swing.

• A weather application might use a device's **temperature sensor and humidity** sensor to calculate and report the dewpoint, or a navigation application might use the **geomagnetic field sensor and accelerometer** to report a compass bearing.

The Sensor Framework

• The sensor framework provides several classes and interfaces that help you perform a wide variety of sensor-related tasks. You can use the sensor framework to do the following:

- Determine which sensors are available on a device.

- Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.

- Acquire raw sensor data and define the minimum rate at which you acquire sensor data.

- Register and unregister sensor event listeners that monitor sensor changes.

Sensor Types

• The Android platform supports three broad categories of sensors:

Motion sensors - These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.

Environmental sensors - These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

Navigation sensors - These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

• Some of these sensors are **hardware-based** and some are **software-based**. Hardware-based sensors are physical components built into a handset or tablet device. They derive their data by directly measuring specific environmental properties, such as acceleration, or angular change. Software-based sensors are not physical devices, although they mimic hardware-based sensors. Software-based sensors **derive their data from one or more** of the hardware-based sensors.

Sensor Types

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s^2 that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass.

Sensor Types

TYPE_ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the getRotationMatrix() method.	Determining device position.
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in	Monitoring temperatures.

Sensor Framework

• You can access these sensors and acquire raw sensor data by using the Android sensor framework. The sensor framework is part of the android.hardware package and includes the following classes and interfaces:

• SensorManager: You can use this class to create **an instance of the sensor service**. This class provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information. This class also provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.
• Sensor: You can use this class to create **an instance of a specific sensor**. This class provides various methods that let you determine a sensor's capabilities.
• SensorEvent: The system uses this class to create a sensor event object, which provides information about a sensor event. A sensor event object includes the following information: **the raw sensor data**, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.
• SensorEventListener: You can use this interface to create two callback methods that receive notifications (sensor events) when **sensor values change** or when **sensor accuracy changes**.

Identifying Sensors

• To identify the sensors that are on a device you first need to get a reference to the sensor service. To do this, you create an instance of the SensorManager class by calling the getSystemService() method and passing in the SENSOR_SERVICE argument.

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

• Next, you can get a listing of every sensor on a device by calling the getSensorList() method and using theTYPE_ALL constant.

```
List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

• If you want to list all of the sensors of a given type, you could use another constant instead of TYPE_ALL such as TYPE_GYROSCOPE, TYPE_LINEAR_ACCELERATION, or TYPE_GRAVITY.

• You can also determine whether a specific type of sensor exists on a device by using the getDefaultSensor()method and passing in the type constant for a specific sensor. If it returns null then the device does not have that sensor.

Identifying Sensor Capabilities

• You can use the public methods of the Sensor class to determine the capabilities and attributes of individual sensors. This is useful if you want your application to behave differently based on which sensors or sensor capabilities are available on a device.

• For example, you can use the getResolution() and getMaximumRange() methods to obtain a sensor's resolution and maximum range of measurement. You can also use the getPower() method to obtain a sensor's power requirements.

• Another useful method is the getMinDelay() method, which returns the minimum time interval (in microseconds) a sensor can use to sense data. Any sensor that returns a non-zero value for the getMinDelay() method is a streaming sensor. Streaming sensors sense data at regular intervals and were introduced in Android 2.3 (API Level 9).

• If a sensor returns zero when you call the getMinDelay() method, it means the sensor is not a streaming sensor because it reports data only when there is a change in the parameters it is sensing.

Monitoring Sensor Events

• To monitor raw sensor data you need to implement two callback methods of the SensorEventListener interface: onAccuracyChanged() and onSensorChanged().

• The Android system calls these methods whenever the following occurs:

A sensor's accuracy changes. In this case the system invokes the onAccuracyChanged() method, providing you with a reference to the Sensor object that changed and the new accuracy of the sensor. Accuracy is represented by one of four status constants:

SENSOR_STATUS_ACCURACY_LOW/MEDIUM/HIGH, or SENSOR_STATUS_UNRELIABLE.

A sensor reports a new value. In this case the system invokes the onSensorChanged() method, providing you with a SensorEvent object. A SensorEvent object contains information about the new sensor data, including: the accuracy of the data, the sensor that generated the data, the timestamp at which the data was generated, and the new data that the sensor recorded.

Monitoring Sensor Events – Light Sensor

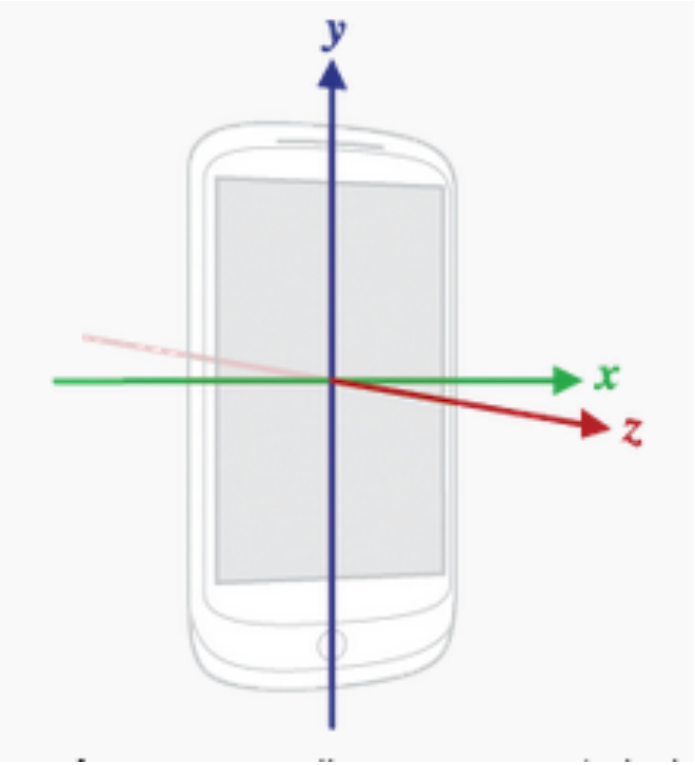
```
public class SensorActivity extends Activity implements SensorEventListener {
  private SensorManager mSensorManager;
  private Sensor mLight;

  @Override
  public final void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
  }

  @Override
  public final void onAccuracyChanged(Sensor sensor, int accuracy) {
    // Do something here if sensor accuracy changes.
  }

  @Override
  public final void onSensorChanged(SensorEvent event) {
    // The light sensor returns a single value.
    // Many sensors return 3 values, one for each axis.
    float lux = event.values[0];
    // Do something with this sensor valu
  }
```

- The data delay (or sampling rate) controls the interval at which sensor events are sent to your application via the onSensorChanged() callback method. The default data delay is suitable for monitoring typical screen orientation changes and uses a delay of 200,000 microseconds.

```
  @Override
  protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
  }

  @Override
  protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
  }
}
```

The Sensor Coordinate System

• When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values.



• The most important point to understand about this coordinate system is that the axes are not swapped when the device's screen orientation changes—that is, the sensor's coordinate system never changes as the device moves.

• Another point to understand is that your application must not assume that a device's natural (default) orientation is portrait. The natural orientation for many tablet devices is landscape. And the sensor coordinate system is always based on the natural orientation of a device.

Motion Sensors

• Most Android-powered devices have an accelerometer, and many now include a gyroscope. There sensors are useful for monitoring device movement, such as tilt, shake, rotation, or swing. The movement is usually a reflection of direct user input (for example, a user steering a car in a game).

• All of the motion sensors return multi-dimensional arrays of sensor values for each SensorEvent. For example, during a single sensor event the accelerometer returns acceleration force data for the three coordinate axes, and the gyroscope returns rate of rotation data for the three coordinate axes. These data values are returned in a float array (values) along with other SensorEvent parameters.

Motion Sensors

Sensor	Sensor event data	Description	Units of measure
`TYPE_ACCELEROMETER`	`SensorEvent.values[0]`	Acceleration force along the x axis (including gravity).	m/s^2
	`SensorEvent.values[1]`	Acceleration force along the y axis (including gravity).	
	`SensorEvent.values[2]`	Acceleration force along the z axis (including gravity).	
`TYPE_GYROSCOPE`	`SensorEvent.values[0]`	Rate of rotation around the x axis.	rad/s
	`SensorEvent.values[1]`	Rate of rotation around the y axis.	
	`SensorEvent.values[2]`	Rate of rotation around the z axis.	

Using the Accelerometer

• An acceleration sensor measures the acceleration applied to the device, including the force of gravity. For this reason, when the device is sitting on a table (and not accelerating), the accelerometer reads a magnitude of g = 9.81 m/s^2.

• Similarly, when the device is in free fall and therefore rapidly accelerating toward the ground at 9.81 m/s^2, its accelerometer reads a magnitude of g = 0 m/s^2. Therefore, to measure the real acceleration of the device, the contribution of the force of gravity must be removed from the accelerometer data.

• To get an instance of the accelerometer you use the following:

```
private SensorManager mSensorManager;
private Sensor mSensor;
  ...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Using the Accelerometer

• Accelerometers use the standard sensor coordinate system. In practice, this means that the following conditions apply when a device is laying flat on a table in its natural orientation:

• If you push the device on the left side (so it moves to the right), the x acceleration value is positive.
• If you push the device on the bottom (so it moves away from you), the y acceleration value is positive.
• If you push the device toward the sky with an acceleration of A m/s^2, the z acceleration value is equal to A + 9.81, which corresponds to the acceleration of the device (+A m/s^2) minus the force of gravity (-9.81 m/s^2).
• The stationary device will have an acceleration value of +9.81, which corresponds to the acceleration of the device (0 m/s^2 minus the force of gravity, which is -9.81 m/s^2).

In general, the accelerometer is a good sensor to use if you are monitoring device motion. Almost every Android-powered handset and tablet has an accelerometer, and it uses about 10 times less power than the other motion sensors. One drawback is that you might have to implement low-pass and high-pass filters to eliminate gravitational forces and reduce noise.

Accessing Acceleration Data and Removing the Gravity Component

- We can isolate the force of gravity using a low pass filter. This can then be subtracted from the acceleration data to get the true linear acceleration

```
public void onSensorChanged(SensorEvent event){
  // In this example, alpha is calculated as t / (t + dT),
  // where t is the low-pass filter's time-constant and
  // dT is the event delivery rate.

  final float alpha = 0.8;

  // Isolate the force of gravity with the low-pass filter.
  gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
  gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
  gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];

  // Remove the gravity contribution with the high-pass filter.
  linear_acceleration[0] = event.values[0] - gravity[0];
  linear_acceleration[1] = event.values[1] - gravity[1];
  linear_acceleration[2] = event.values[2] - gravity[2];
}
```

Device Orientation in Android

•Device orientation lets you monitor the position of a device relative to the earth's frame of reference (specifically, magnetic north).

• There is an orientation sensor (software based) that you can access in the usual way (using SensorEvent) but this has been deprecated and the recommended approach is now to use the method getOrientation() from the SensorManager class.

• Being able to retrieve device orientation is useful for navigation applications.

Device Orientation in Android

public static float[] getOrientation (float[] R, float[] values)

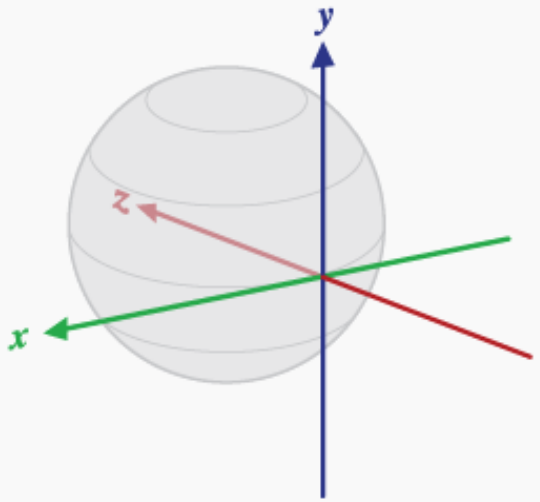
Computes the device's orientation based on the rotation matrix.

When it returns, the array values is filled with the result:

- values[0]: *azimuth*, rotation around the Z axis.
- values[1]: *pitch*, rotation around the X axis.
- values[2]: *roll*, rotation around the Y axis.

The reference coordinate-system used is different from the world coordinate-system defined for the rotation matrix:

- X is defined as the vector product **Y.Z** (It is tangential to the ground at the device's current location and roughly points West).
- Y is tangential to the ground at the device's current location and points towards the magnetic North Pole.
- Z points towards the center of the Earth and is perpendicular to the ground.



All three angles above are in **radians** and **positive** in the **counter-clockwise** direction.

Sending an SMS in your app

- In Android, you can use SmsManager API or device's Built-in SMS application to send a SMS message.

- 1.SmsManager API

```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage("phoneNo", null, "sms message",
  null, null);
```

- 2.Built-in SMS application

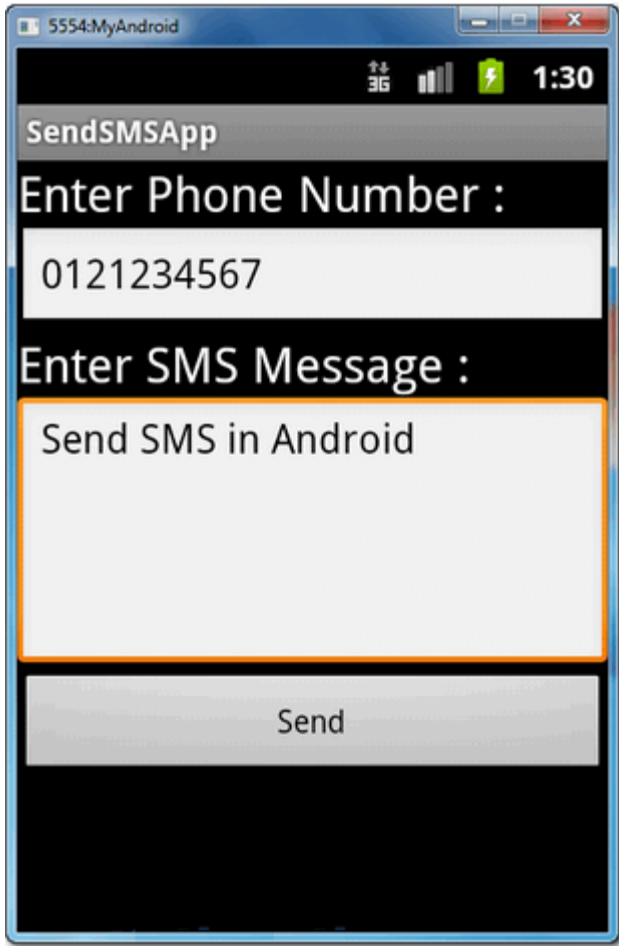
```
Intent sendIntent = new Intent(Intent.ACTION_VIEW);
sendIntent.putExtra("sms_body", "default content");
sendIntent.setType("vnd.android-dir/mms-sms");
startActivity(sendIntent);
```

Both need SEND_SMS permission in the manifest.

<uses-permission android:name="android.permission.SEND_SMS" />

Sending an SMS using the API

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/linearLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textViewPhoneNo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Enter Phone Number : "
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/editTextPhoneNo"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:phoneNumber="true" >
    </EditText>

    <TextView
        android:id="@+id/textViewSMS"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Enter SMS Message : "
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/editTextSMS"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="textMultiLine"
        android:lines="5"
        android:gravity="top" />

    <Button
        android:id="@+id/buttonSend"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Send" />

</LinearLayout>
```



Sending an SMS in your app

```
import android.app.Activity;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class SendSMSActivity extends Activity {

        Button buttonSend;
        EditText textPhoneNo;
        EditText textSMS;

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);

                buttonSend = (Button) findViewById(R.id.buttonSend);
                textPhoneNo = (EditText) findViewById(R.id.editTextPhoneNo);
                textSMS = (EditText) findViewById(R.id.editTextSMS);

                buttonSend.setOnClickListener(new OnClickListener() {

                        @Override
                        public void onClick(View v) {

                          String phoneNo = textPhoneNo.getText().toString();
                          String sms = textSMS.getText().toString();

                          try {
                                SmsManager smsManager = SmsManager.getDefault();
                                smsManager.sendTextMessage(phoneNo, null, sms, null, null);
                                Toast.makeText(getApplicationContext(), "SMS Sent!",
                                                Toast.LENGTH_LONG).show();
                          } catch (Exception e) {
                                Toast.makeText(getApplicationContext(),
                                        "SMS faild, please try again later!",
                                        Toast.LENGTH_LONG).show();
                                e.printStackTrace();
                          }

                        }
                });
        }
}
```

JSON Parsing

• JSON stands for JavaScript Object Notation.It is an independent data exchange format (similar to XML).

• Android provides four different classes to manipulate JSON data. These classes are **JSONArray,JSONObject, JSONStringer and JSONTokenizer.** We will focus on the first two

• The first step is to identify the fields in the JSON data in which you are interested in. For example. In the JSON given below we interested in getting temperature only:


```
{
   "sys":
   {
      "country":"GB",
      "sunrise":1381107633,
      "sunset":1381149604
   },
   "weather":[
   {
      "id":711,
      "main":"Smoke",
      "description":"smoke",
      "icon":"50n"
   }
],
"main":
   {
      "temp":304.15,
      "pressure":1009,
   }
}
```

JSON Elements

1	**Array([)** In a JSON file , square bracket ([) represents a JSON array
2	**Objects({)** In a JSON file, curly bracket ({) represents a JSON object
3	**Key:** A JSON object contains a key that is just a string. Pairs of key/value make up a JSON object
4	**Value:** Each key has a value that could be string , integer or double e.t.c

JSON Parsing

• To parse the JSON data we create a Java JSONObject and pass in the JSON data as a String

```
JSONObject reader = new JSONObject(JSONdata);
```

• An JSON file may consist of different objects with different key/value pairs so the JSONObject has separate functions for parsing the different components of the JSON file, e.g. to get the country and temperature from the previous example

```
JSONObject sys = reader.getJSONObject("sys");
String country = sys.getString("country");

JSONObject main = reader.getJSONObject("main");
 double temperature = main.getDouble("temp");
```

• The method getJSONObject returns the JSON object associated with the key value passed (e.g. “sys”). There are different typed methods then to get the object attributes (e.g. getInt, getDouble etc). There is also a length() method to find the number of name/value pairs and a names() method to return an array of the names or keys

JSON Parsing - Example

```
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class MainActivity extends Activity {
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      TextView output = (TextView) findViewById(R.id.textView1);
      String strJson="
      {
         \"Employee\" :[
         {
            \"id\":\"01\",
            \"name\":\"Gopal Varma\",
            \"salary\":\"500000\"
         },
         {
            \"id\":\"02\",
            \"name\":\"Sairamkrishna\",
            \"salary\":\"500000\"
         },
         {
            \"id\":\"03\",
            \"name\":\"Sathish kallakuri\",
            \"salary\":\"600000\"
         }
         ]
      }";
      String data = "";
```

JSON Parsing - Example

```
        String data = "";
        try {
            JSONObject  jsonRootObject = new JSONObject(strJson);

            //Get the instance of JSONArray that contains JSONObjects
            JSONArray jsonArray = jsonRootObject.optJSONArray("Employee");

            //Iterate the jsonArray and print the info of JSONObjects
            for(int i=0; i < jsonArray.length(); i++){
                JSONObject jsonObject = jsonArray.getJSONObject(i);

                int id = Integer.parseInt(jsonObject.optString("id").toString());
                String name = jsonObject.optString("name").toString();
                float salary = Float.parseFloat(jsonObject.optString("salary").toString());

                data += "Node"+i+" : \n id= "+ id +" \n Name= "+ name +" \n Salary= "+ salary +" \n ";
            }
            output.setText(data);
        } catch (JSONException e) {e.printStackTrace();}
    }
}
```