Mobile Application Development for IoT Unit-1 Material

Syllabus:

Setting up Your Workspace: Hardware and Software requirements, installing Java Developer Kit, Installing Android Studio, setting up the Android Software Development Kit, Hardware configuration, learning to use aREST library, Creating your first Android project
Wi-Fi Remote Security Camera: Hardware and software requirements, Android phone Sensor

Setting Up Your Workspace

Project-1: Simple project with a relay module and a temperature and humidity sensor

Hardware Requirements:

- The Arduino Uno board
- The 5V relay module
- The DHT11 or DHT22 sensor
- 4.7K resistor,
- Small breadboard
- Jumper wires

Note: All the above components are available in Amazon.in

Software Requirements

- Java Developer Kit Version 6 or higher
- Android Studio
- Android Software Development Kit
- Android Device with Bluetooth SMART technology
- Arduino IDE at http://Arduino.cc/en/Main/Software
- Library for DHT11 sensor, which can be found at https://github.com/adafruit/DHT-sensor-library
- REST library found at https://github.com/marcoschwartz/aREST

To install a given library, simply extract the folder in your Arduino/libraries folder (or create this folder if it doesn't exist yet)

Installing Java Developer Kit

- Checking the JDK version
- Open command prompt and type the following command:

  java –version

If java not installed download latest version of java from Oracle website and install

Installing Android Studio:

- Go the Android Developers website https://developer.android.com/studio and download the latest Android Studio

- Open the downloaded file, and then go through the Android Studio Setup Wizard window to complete the installation process
- The latest Software Development Kit (SDK) come preinstalled with the Android Studio install package.
- Select SDK Manager from Android Studio main toolbar, and install any packages that are needed by accepting the licenses and clicking on 'Install packages'

Setting up physical Android device for development:

The following are the three main steps that need to be executed in order to enable your Android device for development

1. Enable Developer options on your specific Android device
2. Enable USB debugging
3. Entrust the computer with and installed IDE via secure USB debugging (devices with Android 4.4.2)

1. ***Enabling Developer options:***

Depending on your device, this option might vary slightly, but from Android 4.2 and higher, the Developer options screen is hidden by default

To make it available, go the Settings | About phone and tap on Build number seven times. You will find Developer options enabled by returning to the previous screen or displaying a pop up message 'You are now a developer'
(on my realme mobile-> Settings-> About phone-> version->Build number)

2. Enabling USB Debugging:
USB debugging enables the IDE to communicate with the device via the USB port.
This can be activated after enabling Developer options and is done by checking the USB debugging options by navigating to
Settings | Developer options | Debugging | USB debugging

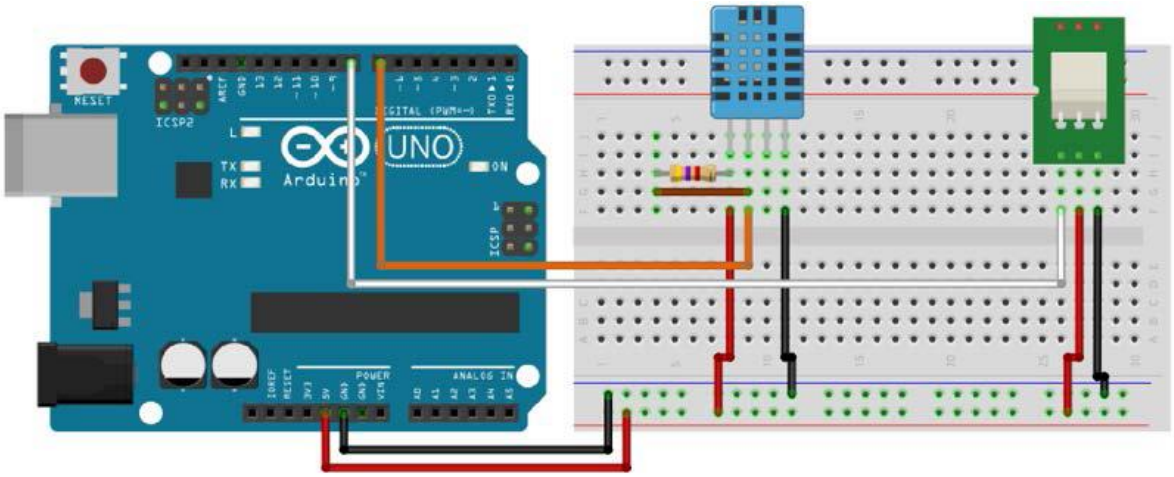
On my realme mobile: Settings-> System->Developer options -> Debugging->USB Debugging

3.Entrusting the computer with the installed IDE using secure USB debugging (devices with Android 4.4.2)

We have to accept the RSA key on our phone or tablet before anything can flow between the device via Android Debug Bridge (ADB).
This is done by connecting the device to the computer via a USB, which triggers a notification entitled "Enable USB Debugging'
Check Always allow from this Computer followed by clicking on OK

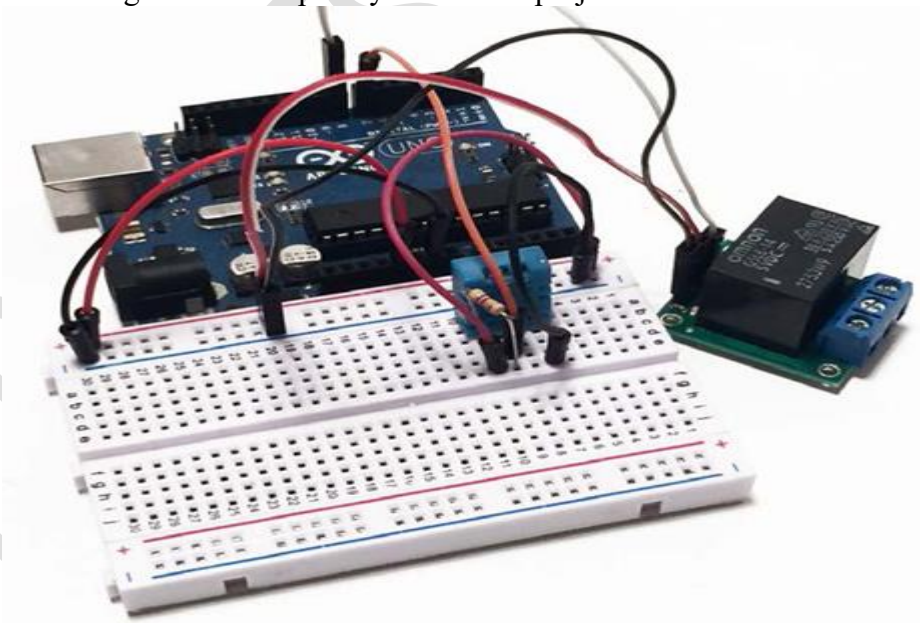
Note: When I enable USB Debugging in my realme mobile, it asked confirmation "Enable USB Debugging" -> Select Yes. The computer's RSA key fingerprint is: F7:B3:0B:9A:--- (Enable Always allow from this computer) and then click 'Allow' [will be asked only first time]

Hardware Configuration:
The following image summarizes the hardware connections with DHT sensor on the left of the breadboard, and the relay module on the right



Step-1: Connect the power supply. The Voltage Common Collector(VCC) pin goes to the red power rail on the breadboard, and the Ground(GND) pin goes to the blue power rail.
Step-2: Connect the DATA pin to the pin number 7 of the Arduino board.
Step-3:Place the 4.7k ohm resistor between the VCC and the DATA pin of the sensor
Step-4: For the relay module, Connect the VCC pin to the red power rail on the breadboard, GND to the blue power rail, and Signal (SIG) pin to Arduino pin 8.
Step-5: For DHT11 Sensor, Connect pin no.1 to VCC, pin no.2 to DATA, pin no.3 to NC and pin no.4 to GND

The following is an image of the completely assembled project

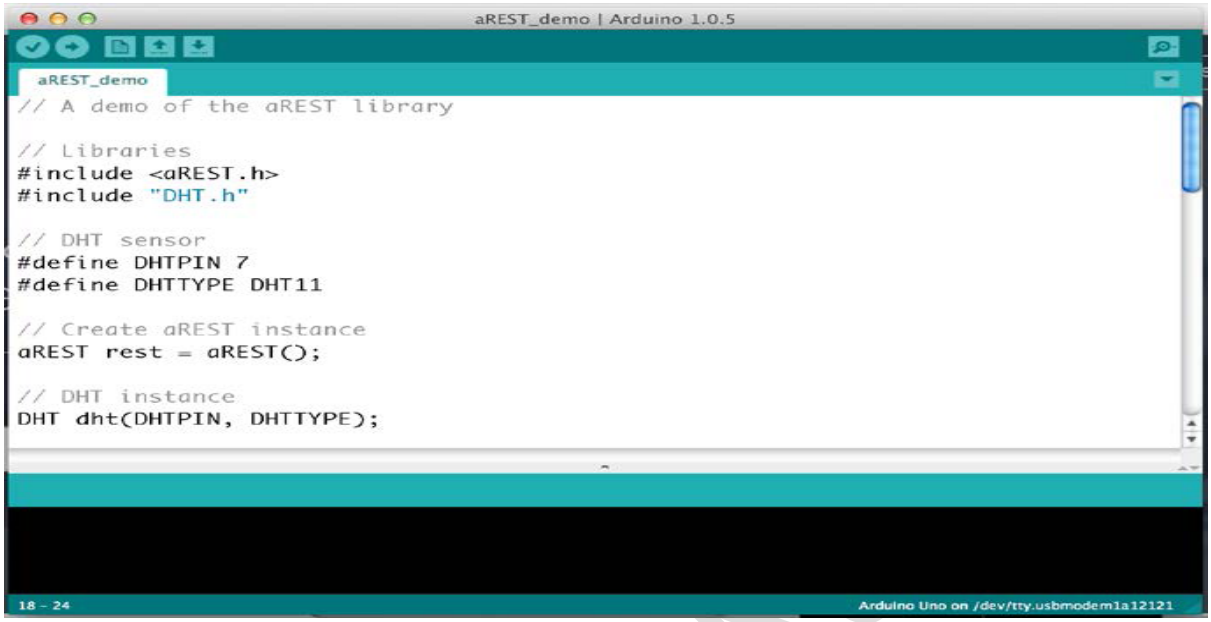


Using aREST library

- The aREST library will allow us to simply control the Arduino board externally using the same commands, whether it is using an USB cable, Bluetooth, or Wi-Fi.
- The complete documentation on the aREST library is available at https://github.com/marcoschwartz/aREST

Arduino IDE

The main window of the Arduino IDE is where we enter the code to program the Arduino board.
Arduino code files are called sketches. The following screenshot is of the Arduino IDE with the code



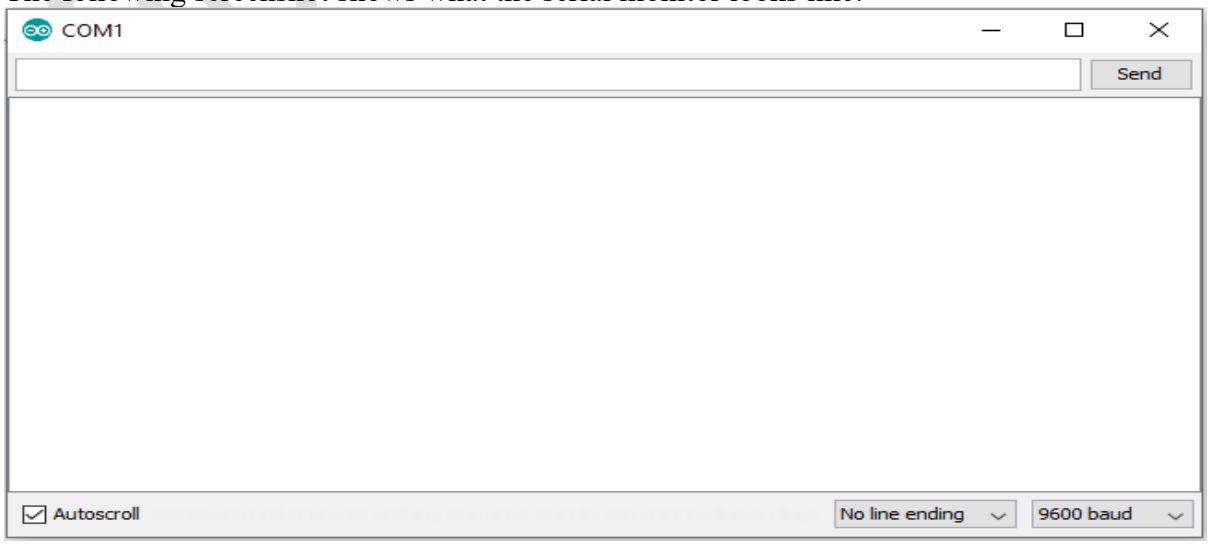
We basically use two buttons that your can find on the left-hand side of the toolbar.

- The first one, with the check sign, can be used to compile the code
- The second one will be used to upload the code to the Arduino board.

Note: If the code has not been compiled yet, the upload button will also compile the code before uploading

- The second important window of the Arduino IDE is called the serial monitor, where we can monitor what our Arduino project is doing
- By using the Serial.print() statements in the code to generate debug output.
- We can access it by clicking on the top-right icon of the Arduino IDE main window.

The following screenshot shows what the serial monitor looks like:



Arduino Sketch for the project:

```
// Libraries (The Arduino sketch starts by importing the required libraries for the project
#include <aREST.h>
#include "DHT.h"
// DHT sensor (Define on which pin the DHT11 sensor is connected to and which is the type
//of the sensor
#define DHTPIN 7
#define DHTTYPE DHT11
// Create aREST instance
aREST rest = aREST();
// Create an instance of DHT11 sensor so that we can measure data from it
DHT dht(DHTPIN, DHTTYPE);
// Create two variables that will contain our measurements
int temperature;
int humidity;
void setup(void) {
// Start Serial port (with 115200 as the baud rate)
Serial.begin(115200);
// Expose our two measurement variables so that we can access them via the serial //port
using the aREST library. (pass references not values)
rest.variable("temperature",&temperature);
rest.variable("humidity",&humidity);
// Give ID and Name to our project
rest.set_id("001");
rest.set_name("arduino_project");
// Start DHT11 sensor
dht.begin();
}
void loop() {
// Measure from DHT11 sensor, and convert these measurements to
//integers (as aREST library support only integers)
float h = dht.readHumidity();
float t = dht.readTemperature();
temperature = (int)t;
humidity = (int)h;
// Handle any requests coming from the outside
rest.handle(Serial);
}
```

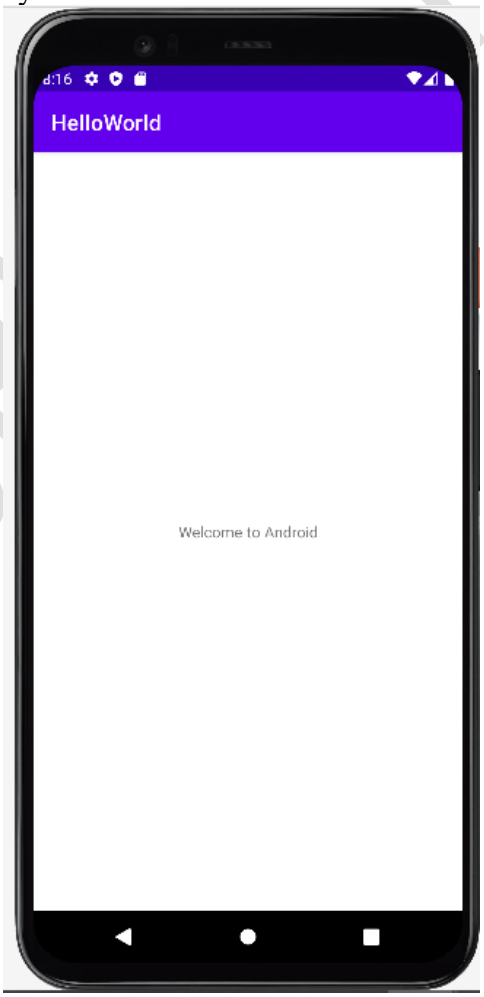
aREST library commands:

Command	Output
/id	{"id": "001", "name": "arduino_project", "connected": true}
/mode/8/o	{"message": "Pin D8 set to output", "id": "001", "name": "arduino_project", "connected": true}
/digital/8/1	Activate the relay

/digital/8/0	Switch off the relay
/temperature	{"temperature": 28, "id": "001", "name": "arduino_project", "connected": true}
/humidity	{"humidity": 35, "id": "001", "name": "arduino_project", "connected": true}

Creating First Android Project

Step-1: Click on 'New Project' when Android Studio launches
Step-2: Select 'Phone and Tablet' template and then select 'Empty Activity' and then click on 'next'
Step-3: Give Name, Package name. Select 'Save location' and 'Java' as Language. Select Minimum SDK and then click on 'Finish'
Step-4: Select Tools->AVD Manager->Create Device and create a virtual device on Emulator (Eg: Pixel 4 API 33)
Step-5: Select the virtual device on which we want to run the application and click on 'play' button
Step-6: We may connect a physical device using USB cable and select the physical device(Eg: realme RMX3085) and click on 'play' button
The Output will be display as shown below



Project-2: Wi-Fi Remote Security Camera

Hardware Requirements

- The Wi-Fi remote security camera project is based around the Arduino Yun board.
- The Arduino Yun is a powerful Arduino board with integrated Wi-Fi and an onboard Linux machine based on a very small Linux distribution called OpenWrt.
- It also has a USB port so that you can connect hard drives, cameras, or other USB devices

The following is an image shows Arduino YUN Board

- A USB camera to stream live video with the Yun(Compatible with USB Video Class)
- A micro USB cable to power the Yun.
- A micro SD card to save the recorded data (optional)

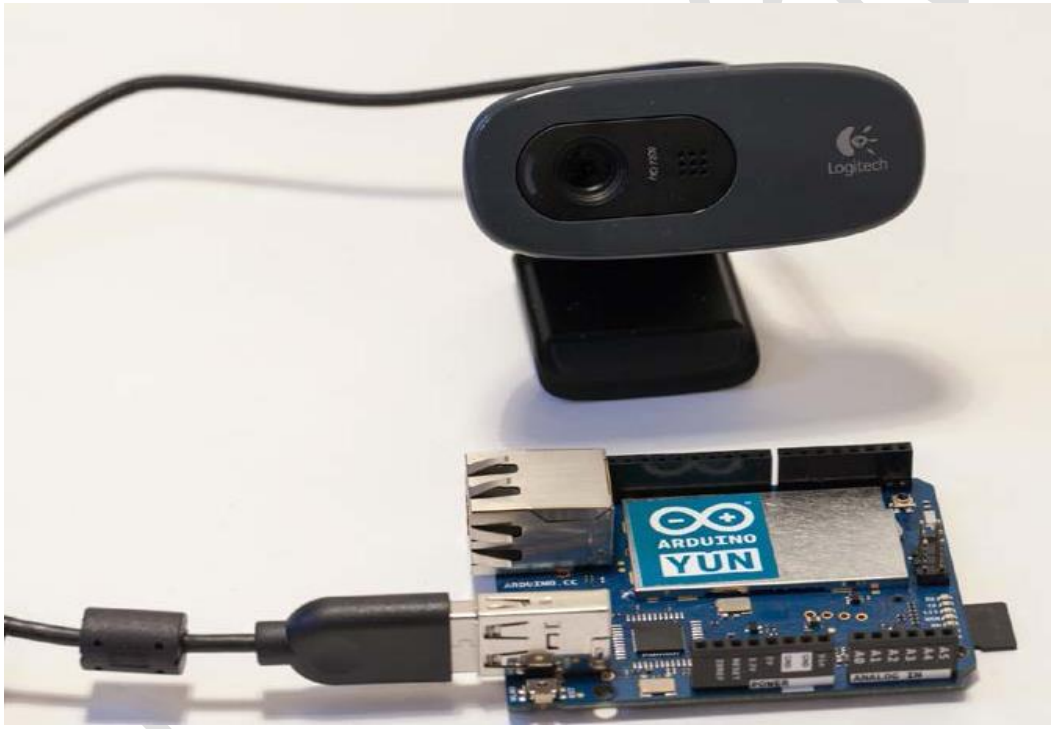
Software Requirements (Arduino Yun)

- Install PuTTY or OpenSSH software to get a terminal
- Type the command 'ssh root@yourYunName.local' (name given during configuring Yun board)
- Enter the password (password is defined during the Yun's configuration step)
- Type the command 'opkg update' to update the list of available packages
- Type the command 'opkg install kmod-video-uvc mjpg-streamer'

Hardware Configuration

Step-1: Insert the formatted microSD card into the Arduino Yun SD card reader as shown in the following screenshot

Step-2: Connect the USB camera to the host USB port of the Yun, as shown in the following screenshot:



Step-3: Connect the board to power via the micro USB port

Setting up video streaming

Step-1: Login to your Arduino Yun using the following command: ssh root@yourYunName
Step-2: Type the command
mjpg_streamer -i "input_uvc.so -d /dev/video0 -r 640x480 -f 25" -o "output_http.so -p 8080 -w /www/webcam" &
(Start the streaming at resolution of 640x480, at 25 frames per second, on the 8080 port)
Step-3: Go to web browser and type yourYunName.local:8080, which will open the main streaming interface.
Step-4: To access the steam itself for a test, go to http://arduinoYun.local:8080/stream.html

Live Streaming Coming from Arduino Yun to PC:

Implementing a full screen stream player on Android

Step-1: Switch on the 'Auto-Import' function within your Android preferences. (File->Settings->Editor->Auto-Import)

Step-2: Create a New Project with the following setup:

- Name: Android Yun Security
- Minimum SDK: 15
- Project: Empty Activity
- Activity Name: StreamActivity

Step-3: Create three Java classes

- StreamActivity (the main activity that is created upon the start of a new project)
- MjpegInputStream
- MjpegView

(To create a new class, you will need to go to app->src->main->java->com.domainofyourchoice.androidyunsecurity)
Right click on the package name and select 'new->Java Class' and give the specific classname

Step-4: Add the following code below the package name in 'AndroidManifest.xml' file:

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

<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"
(Add these lines of code below <manifest> tag)

Important Packages Used in MjpegView.java class:

Package android.context.Context

public abstract class Context extends Object
This is an abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc.

package android.graphics.Bitmap

public final class Bitmap extends Object implements Parcelable
A bitmap is a digital image composed of a matrix of dots. When viewed at 100%, each dot corresponds to an individual pixel on a display. In a standard bitmap image, each dot can be assigned a different color.

package android.util.AttributeSet

public interface AttributeSet
This interface provides an efficient mechanism for retrieving data from compiled XML files
package android.view.SurfaceHolder

public interface SurfaceHolder

This interface Allows you to control the surface size and format, edit the pixels in the surface, and monitor changes to the surface. This interface is typically available through the SurfaceView class

package android.view.SurfaceView

public class SurfaceView extends View
Provides a dedicated drawing surface embedded inside of a view hierarchy. You can control the format of this surface and its size. The SurfaceView takes care of placing the surface at the correct location on the screen

public static interface SurfaceHolder.Callback

A client may implement this interface to receive information about changes to the surface.

Code Snapshot for MjpegView.java:

```
package in.co.jntuanantapur.arduinoyunsecurity;
    import android.content.Context;
    import android.graphics.Bitmap;
    import android.util.AttributeSet;
    import android.view.SurfaceHolder;
    import android.view.SurfaceView;
```

```
public class MjpegView extends SurfaceView implements SurfaceHolder.Callback
{
   public class MjpegViewThread extends Thread
        {
                public MjpegViewThread(SurfaceHolder surfaceHolder, Context context)
                {
                }
private void init(Context context)
                {
                }
                public void run()
                {
                }
               public void setSurfaceSize(int width, int height)
                {
                }
                 private Bitmap makeFpsOverlay(Paint p, String text)
                {
                }
               public void startPlayback()
                {
                }
              public void stopPlayback()
                {
                }
        }
        public MjpegView(Context context, AttributeSet attrs)
        {
        }
        public MjpegView(Context context)
        {
        }
        public void surfaceCreated(SurfaceHolder holder)
        {
        }
        public void surfaceChanged(SurfaceHolder holder, int f, int w, int h)
        {
        }
        public void surfaceDestroyed(SurfaceHolder holder)
        {
        }
        public void showFps(boolean b)
        {
        }
        public void setSource(MjpegInputStream source)
        {
        }
        public void setOverlayPaint(Paint p)
        {
        }
        public void setOverlayTextColor(int c)
        {
```

```
        public void setOverlayBackgroundColor(int c)
        {

        }
        public void setOverlayPosition(int p)
        {

        }
        public void setDisplayMode(int s)
        {

        }
        }
}
```

Important Packages Used in MjpegInputStream.java class

Package java.io.DataInputStream

public class DataInputStream extends FilterInputStream implements DataInput
A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way. An application uses a data output stream to write data that can later be read by a data input stream.

Code Snapshot for MjpegInputStream.java:

```
package in.co.jntuanantapur.arduinoyunsecurity;
import java.io.DataInputStream;
import java.io.IOException;
public class MjpegInputStream extends DataInputStream
{
   public MjpegInputStream(InputStream in)
   {
   }
   private int getEndOfSeqeunce(DataInputStream in, byte[] sequence) throws IOException
   {
   }
private int getStartOfSequence(DataInputStream in, byte[] sequence) throws IOException
   {
   }
   private int parseContentLength(byte[] headerBytes) throws IOException,
NumberFormatException
   {

   }
   public Bitmap readMjpegFrame() throws IOException
   {

   }
}
```

Important Packages Used in StreamActivity.java class:

Package android.os.AsyncTask<Params, Progress, Result>

public abstract class AsyncTask extends Object
AsyncTask was intended to enable proper and easy use of the UI thread

Package android.os.Bundle

public final class Bundle extends BaseBundle implements Cloneable, Parcelable
Bundle is a utility class that lets you store a set of name-value pairs. You will always find this import along with the import for Activity class because both onCreate() and onFreeze() methods take Bundle as a parameter. Into a Bundle object, you can put integers, longs, strings, arrays, etc along with the keys to identify them. When needed, these values can be obtained by using those keys.

Package android.util.Log

public final class Log extends Object
This class allows you to log messages categorized based severity; each type of logging message has its own message.

Package android.view.Window

public abstract class Window extends Object
Abstract base class for a top-level window look and behavior policy. An instance of this class should be used as the top-level view added to the window manager. It provides standard UI policies such as a background, title area, default key processing, etc.

Package android.view.WindowManager

public interface WindowManager implements ViewManager
The interface that apps use to talk to the window manager. Each window manager instance is bound to a Display.

Package org.apache.http.HttpResponse

public interface HttpResponse extends HttpMessage
After receiving and interpreting a request message, a server responds with an HTTP response message

Package java.net.URI

public final class URI extends Object implements Comparable<URI>, Serializable
Represents a Uniform Resource Identifier (URI) reference

Code for StreamActivity.java

```
package com.example.lalitha.myapplication;
import android.app.Activity;
    import android.os.AsyncTask;
    import android.os.Bundle;
```

```
import android.util.Log;
import android.view.Window;
import android.view.WindowManager;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;

import java.io.IOException;
import java.net.URI;
public class StreamActivity extends Activity {
  private static final String TAG = "MjpegActivity";
  private MjpegView mv;

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    //sample public cam
    String URL = "http://10.0.1.6:8080/?action=stream";

    requestWindowFeature(Window.FEATURE_NO_TITLE);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN);
      mv = new MjpegView(this);
    setContentView(mv);

    new DoRead().execute(URL);
  }

  public void onPause() {
    super.onPause();
    mv.stopPlayback();
  }

  public class DoRead extends AsyncTask<String, Void, MjpegInputStream> {
    protected MjpegInputStream doInBackground(String... url) {
      //TODO: if camera has authentication deal with it and don't just not work
      HttpResponse res = null;
      DefaultHttpClient httpclient = new DefaultHttpClient();
      Log.d(TAG, "1. Sending http request");
      try {
        res = httpclient.execute(new HttpGet(URI.create(url[0])));
        Log.d(TAG, "2. Request finished, status = " + res.getStatusLine().getStatusCode());
        if(res.getStatusLine().getStatusCode()==401){
          //You must turn off camera User Access Control before this will work
          return null;
        }
        return new MjpegInputStream(res.getEntity().getContent());
      }
      HttpResponse res = null;
      DefaultHttpClient httpclient = new DefaultHttpClient();
      Log.d(TAG, "1. Sending http request");
```

```
            try {
                res = httpclient.execute(new HttpGet(URI.create(url[0])));
                Log.d(TAG, "2. Request finished, status = " + res.getStatusLine().getStatusCode());
                if(res.getStatusLine().getStatusCode()==401){
                    //You must turn off camera User Access Control before this will work
                    return null;
                }
                return new MjpegInputStream(res.getEntity().getContent());
            catch (ClientProtocolException e) {
                e.printStackTrace();
                Log.d(TAG, "Request failed-ClientProtocolException", e);
                //Error connecting to camera
            } catch (IOException e) {
                e.printStackTrace();
                Log.d(TAG, "Request failed-IOException", e);
                //Error connecting to camera
            }

            return null;
        }       }
protected void onPostExecute(MjpegInputStream result) {
            mv.setSource(result);
            mv.setDisplayMode(MjpegView.SIZE_BEST_FIT);
            mv.showFps(true);
        }
    }
}
```