```
package com.example.ble_basic;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;



import android.bluetooth.BluetoothGattCallback;
import android.annotation.TargetApi;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothManager;
import android.bluetooth.le.BluetoothLeScanner;
import android.bluetooth.le.ScanCallback;
import android.bluetooth.le.ScanFilter;
import android.bluetooth.le.ScanResult;
import android.bluetooth.le.ScanSettings;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.graphics.Color;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.os.ParcelUuid;
import android.text.TextUtils;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.Window;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import android.view.Menu;
import android.view.MenuItem;
import android.util.SparseArray;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCallback;
import android.bluetooth.BluetoothGattCharacteristic;
    /**
     * Created by Dave Smith
     * Double Encore, Inc.
     * BeaconActivity
     */
    @TargetApi(Build.VERSION_CODES.LOLLIPOP)
    public class MainActivity extends AppCompatActivity {
        private static final String TAG = "MainActivity";
        private static final String DEVICE_NAME = "RFduino";

        private BluetoothAdapter mBluetoothAdapter;
        private BluetoothLeScanner mBluetoothLeScanner;
        private BluetoothGatt mConnectedGatt;
```

```
    public static final ParcelUuid RFduino_SERVICE = ParcelUuid.fromString("00002220-0000-1000-8000-
00805f9b34fb");



    public static final ParcelUuid RFduino_READ_CHAR = ParcelUuid.fromString("00002221-0000-1000-8000-
00805f9b34fb");

    private SparseArray<BluetoothDevice> mDevices;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);


        requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
        setProgressBarIndeterminate(true);


        /*
         * Bluetooth in Android 4.3+ is accessed via the BluetoothManager, rather than
         * the old static BluetoothAdapter.getInstance()
         */
        BluetoothManager manager = (BluetoothManager) getSystemService(BLUETOOTH_SERVICE);
        mBluetoothAdapter = manager.getAdapter();
        mBluetoothLeScanner = mBluetoothAdapter.getBluetoothLeScanner();
        mDevices = new SparseArray<BluetoothDevice>();

    }

    @Override
    protected void onResume() {
        super.onResume();
        /*
         * We need to enforce that Bluetooth is first enabled, and take the
         * user to settings to enable it if they have not done so.
         */
        if (mBluetoothAdapter == null || !mBluetoothAdapter.isEnabled()) {
            //Bluetooth is disabled
            Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivity(enableBtIntent);
            finish();
            return;
        }

        /*
         * Check for Bluetooth LE Support.  In production, our manifest entry will keep this
         * from installing on these devices, but this will allow test devices or other
         * sideloads to report whether or not the feature exists.
         */
        if (!getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE)) {
            Toast.makeText(this, "No LE Support.", Toast.LENGTH_SHORT).show();
            finish();
            return;
        }

        //Begin scanning for LE devices
        startScan();
    }

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

```
        //Cancel scan in progress
        stopScan();
    }

    private void startScan() {
        Log.i(TAG, "Scanning ... ..");
        //Scan for devices advertising the thermometer service
        //setServiceUuid(TemperatureBeacon.THERM_SERVICE)
        //setDeviceName("RFduino")
        ScanFilter RFduinoFilter = new ScanFilter.Builder()
                .setDeviceName("RFduino")
                .build();
        ArrayList<ScanFilter> filters = new ArrayList<ScanFilter>();
        filters.add(RFduinoFilter);

        ScanSettings settings = new ScanSettings.Builder()
                .setScanMode(ScanSettings.SCAN_MODE_LOW_LATENCY)
                .build();

        mBluetoothLeScanner.startScan(mScanCallback);//filters, settings, mScanCallback);
    }

    private void stopScan() {
        mBluetoothLeScanner.stopScan(mScanCallback);
    }

    private ScanCallback mScanCallback = new ScanCallback() {
        @Override
        public void onScanResult(int callbackType, ScanResult result) {
            Log.d(TAG, "onScanResult");
            processResult(result);
        }


        @Override
        public void onScanFailed(int errorCode) {
            Log.w(TAG, "LE Scan Failed: "+errorCode);
        }

        private void processResult(ScanResult result) {
            Log.i(TAG, "New LE Device: " + result.getDevice().getName() + " @ " + result.getRssi());



            /*
             * We are looking for SensorTag devices only, so validate the name
             * that each device reports before adding it to our collection
             */
            if (DEVICE_NAME.equals(result.getDevice().getName())) {
                mDevices.put(result.getDevice().hashCode(), result.getDevice());
                //Update the overflow menu
                invalidateOptionsMenu();
            }


        }
    };

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Add the "scan" option to the menu
        getMenuInflater().inflate(R.menu.main, menu);
```

```
        //Add any device elements we've discovered to the overflow menu
        for (int i=0; i < mDevices.size(); i++) {
            BluetoothDevice device = mDevices.valueAt(i);
            menu.add(0, mDevices.keyAt(i), 0, device.getName());
        }

        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {

        switch (item.getItemId()) {
            case R.id.action_scan:
                mDevices.clear();
                startScan();
                return true;
            default:
                //Obtain the discovered device to connect with
                BluetoothDevice device = mDevices.get(item.getItemId());

                /*
                 * Make a connection with the device using the special LE-specific
                 * connectGatt() method, passing in a callback for GATT events
                 */
                mConnectedGatt = device.connectGatt(this, false, mGattCallback);
                Log.i(TAG, "Connected "+device.getName());
                return super.onOptionsItemSelected(item);
        }




    }

    private BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {
      //reading data
    };


}
```