ELEC3607 Project Group 9: A Wireless Car Park Reservation System

Final Report

Team members: *Qishen Guan 450549016*
Mingyuan Xia 450549223

June/2017

Project Implementation Summary Form

Category	List of items, with web link or page number where item is described
Open source compliance **Are you publishing your code as open source? Explain what license you are using, or link to where you have published your project.**	**Yes, we publish it on github as open source, include all the code and schematic. We use** `MIT License, then everyone can use our code if they mention us.` `Link:` **https://github.com/qgua2322/ELEC3607_carparkproject**
Platforms **List the platforms you have used, including processor architecture (ARM, AVR, x86, etc.), programming languages (Python, C, C++, etc.), and IDEs. If you are not using Arduino Due, justify your choice of platform.**	**Arduino due(ARM): C++** **Android phone(x86/ARM): Java**
Sensors and inputs **List your hardware inputs (push buttons, analog sensors, I2C or SPI sensors including accelerometers, etc.) and mention on which page each input is described.**	**IR sensor(page5)**
Outputs **List any mechanical, visual or other type of outputs controlled by your hardware (motors, magnets, LED, 7-Segs, LCD, etc.) and mention on which page each input is described.**	**IR LED, Servo motor, LCD (page 5-7)**

Connectivity **List any protocols used for communicating between your main controller and any other microcontroller excluding sensors or actuators (USART, UART over USB, Bluetooth 2, Bluetooth 4, Wi-Fi, ZigBee, etc.) and mention on which page each input is described.**	**Bluetooch 4.0 and Bluetooth2.0(page5-7)**
Graphical User Interface **List any GUI used for interfacing users (local LCD, Phone app, Desktop app, Web app, Web dashboard, etc.) and mention on which page each input is described.**	**LCD and custom App on Android (page5-7 for LCD, Page 8 for custom App on Android)**
Algorithms / Logic **List substantial control algorithms (state-machines, real-time operating system, filesystems, feedback algorithms, mathematical transformations, etc.) and mention on which page each input is described.**	**FSM (page6-7)**
Physical case and other mechanical consideration **List what casing or mechanical systems have been used in your project (3d prints, pipes, cardboard mechanics, etc.) and mention on which page each input is described.**	**NO**

1. Abstract

The following project we designed and built is a wireless car parking/billing system. An android device is allowed to communicate with the prototype via Bluetooth 4.0 by using our customized android app. At the car parking lot, a LCD screen will display the number of available car parking, this information is sent to paired android phone as well. On the other hand, the phone app plays the role of a remote controller, it has “reserve”, “leave”, “pay”, “disconnect” buttons and fulfill these functions correspondingly. This will be discussed in detail in Design section. After we built the prototype, we ran tests to test whether it achieved our goal. We found that it was hard to synchronize Arduino Due and the phone, so we always lose a word or two upon receiving messages. This will be expanded in Trouble-shooting section. After the tests, although we approached our expectations, there is much more thing can be done to improve, such as using WIFI instead of BT; but due to time limitation, we left our prototype as what it was. This project is extremely helpful in regarding to gain a better understanding of communications between embedded system & others, and simple real world embedded system design.

2. Aim and difficult:

Live in city like Sydney, a daily headache is to find a parking spot at your travel destination. The time of finding a spot occupies a great percentage of your commuting time, if you live in city, the percentage is even higher. Therefore we came up with an idea of a wireless parking/billing system that can check availability and reserve a parking spot before your arrival. Furthermore, you can also view your bill on the phone in real time and pay your bill when you are leaving.

3.Design:

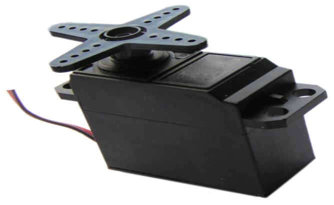
3.1: Components list:

Equipment :	amount
Arduino due	1
Bluetooth 2.0 shield	1
Bluetooth 4.0 shield	1
Servo motor	1
IR LED	1
IR sensor	1
LCD display	1
Resistor 220 (ohm)	2

3.2: Use of each components:
Arduino Due board is the core of this embedded system. All the other components are connected on it. It process the message that is sent from user's App, then output the response to LCD and user's App at the same time. And control the parking space's gate open or off by reading IR signal.

Bluetooth 2.0/4.0 shield provide a wireless connection between embedded system and user's phone App. (Our project used BT2.0 at first, then switch to 4.0 later, Check 3.7 How does this project work to see how to use different version of BT)

Servo motor is used to open/close the gate of our parking space



IR LED and IR sensor is used to detect whether customer's car has parked in or not, in order to close the gate automatically when car has parked in or close the door when customer has left. The reason that choose infrared as detection light is: infrared light is not common in environment light, so it will not be impact very easy by environment. And it is invisible for human.

LCD screen always display the number of free space is remaining, and display the information at each stage. For example: display how long does user's car have been parked in/ How much does user have to pay.

3.3 Hardware schematic: Appendix 1

3.4 Software flowchart: Appendix 2

3.5 Arduino code: Appendix 3

3.6 Android App's code: Appendix 4

3.7 How does this project work:

This system uses a **Finite state machine(FSM)** as the control mechanism, because in FSM we don't need to worry about customer mispress some wrong button(Except "Disconnect"),then the system get into an unknown situation. For example, nothing will happen when customer press "pay" before "leave".Things only be processed in a sequence that we want.

Bluetooth is used as wireless connection, but Arduino Due board does not support SoftwareSerial library, so we need to implement hardware serial to Bluetooth, in order to get it work. (Bluetooth's TX connect with Arduino's Serial2's RX, bluetooth's RX connect with Arduino's Serial2's TX, this is same for both BT2.0 and 4.0. If choose Bluetooth 4.0, uncommand setupBlueToothConnection4() in function void setup() in fsm.ino , command setupBlueToothConnection(), and vice versa.
)

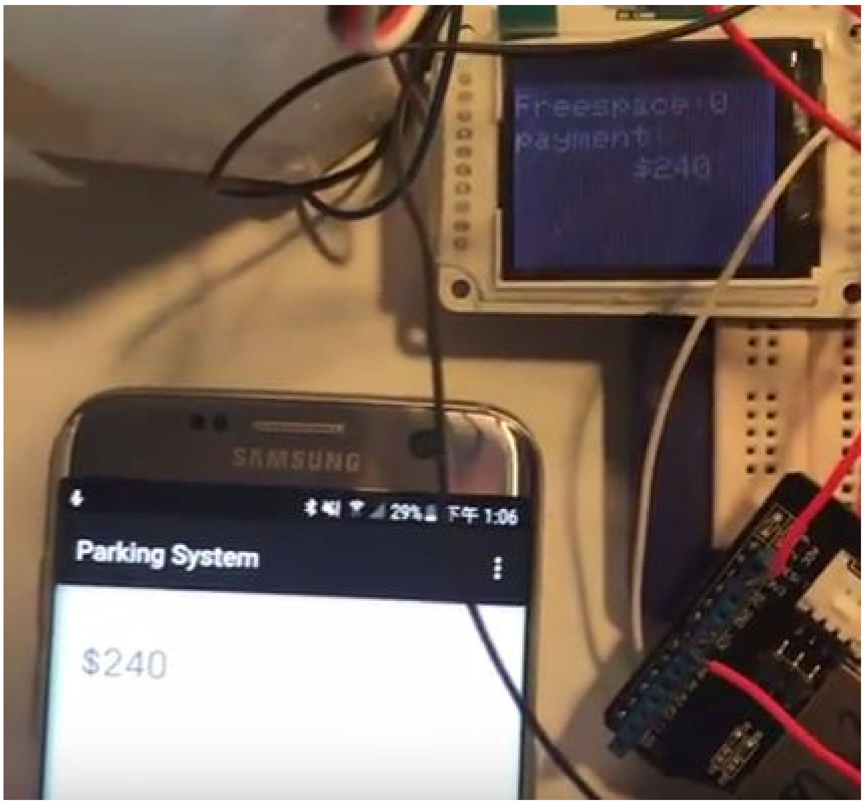
This embedded system implement two **Timer counters(TC)** to achieve time counting and generate a squarewave. We turn TC0 on to generate a 36K Hz square wave signal on digital pin 2, because Arduino Due does not support Tone library. We generate such high frequency signal because it isn't common in environment light, so it is not easy to be affected by surrounding circumstance. TC1 is used to count how long does user's car stay.

Expected operating procedure: When user press "reverse" button on the phone, our embedded system will control the servo motor to open the gate, and update number of free space on LCD. The system will not count time immediately, it will wait until IR sensor detect user's car has parked in, and the gate is fully closed. After that embedded system will update the time to LCD and user's phone simultaneously. When user decide to leave, all he needs to do is to press "leave", then the bill will come up on LCD as well as user's phone. Once user press “pay”, the user is good to go. System will control motor to open the gate, and it will only close after IR sensor detects IR light(meaning the car has left).

3.8 Design verification and result.

IR LED and Sensor: IR light is invisible to human eye, so we used an android phone’s camera with a IR filter to check whether it is operating, And we see a purple light show on IR LED when it is operating. To test IR sensor, we put IR LED face with IR sensor, when there is nothing a ‘0’ show on monitor. After that we put a book in between, a ‘1’ show on monitor, when we remove it ‘0’ come back.

Servo motor: myservo.write(30); move forward, myservo.write(120); move backward
myservo.write(90) stop. Work as expected.
LCD: We print a “hello world” on LCD by TFTscreen.text("Freespace:.", 0, 40); then change the second or third argument of this function to change the position on the LCD screen. Change first argument to output different string. It all work as we expected.



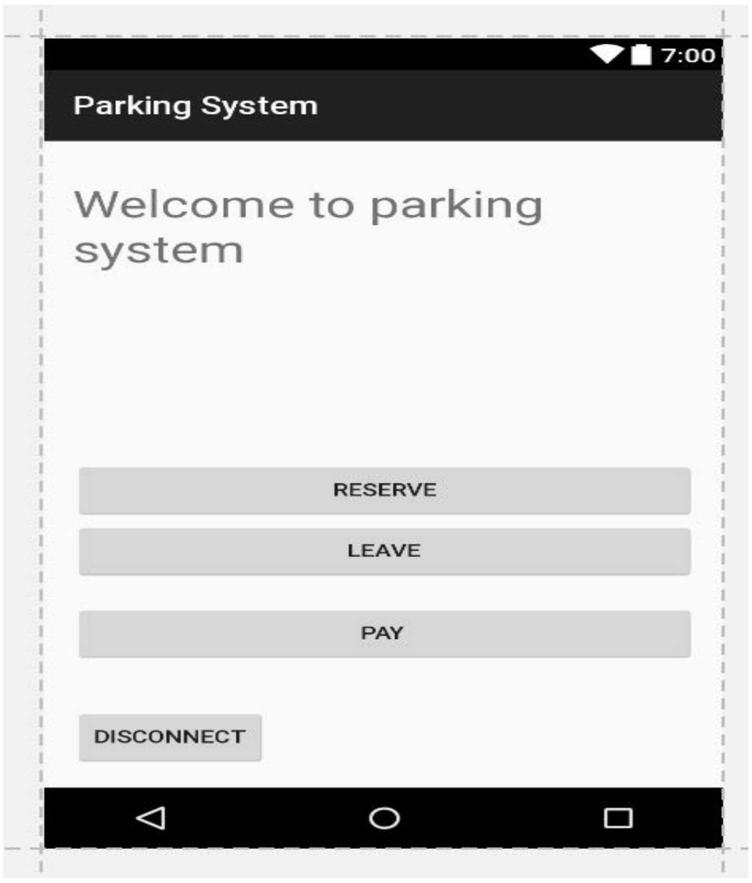
Bluetooth 4.0 and App: Using off the shelf App: Bluetooth Terminal to send and receive some Strings from embedded system. No characters lost in sending or receiving String. Using our customised app, no charate lost in Sending message to embedded system, some character lost in receiving String from embedded system. We decide to add space buffer to the String being sent from embedded system. No chartered lost anymore.

FSM: we test our FSM by pressing the button on the App in unexpected sequence. Like press “pay” before “leave”. No matter what we did, the system only operated in a sequence that we expected. (Unless someone reset the board,or disconnect the power, then that App may break or receive a error message.)
All parts are being verified as what we expected.

App design:

The android app is designed base on an android Bluetooth app tutorial using android developer tool called ***Android studio***, this tutorial is in the reference named Android control app.(Ref 10) The tutorial provides a detailed procedure of creating an app that uses Bluetooth as communication tool. We used the tutorial as a learning material, and create our app follow the tutorial structure. Then we modified our code based on it to suit our needs.

We added more buttons to the user interface and complete their functions correspondingly. We add a textview widget to continuously refresh information displayed on the phone, this is done by using MyAsyncTask(). What it does is to create another thread that will constantly read input string from Bluetooth socket in background. A Timer() is introduced to delay the initialization of MyAsyncTask in order to keep the welcome screen for few seconds. The Timer also controls the time between every execution of MyAsyncTask;therefore the information displayed is refreshed at certain frequency without being interrupted by other button operations. The app happened to crash several times during testing, we managed to sort this out somehow, but as this project is not focused on android we didn't go deep into it and stopped at where we thought it was enough.



Welcome to parking system

4.Trouble-shooting:

There were both hardware and software problems occurred during this project:

1. In order to communicate with phone via Bluetooth, we add a Bluetooth shield to Arduino Due board. Normally, when adding a Bluetooth shield to Arduino, we plug the whole shield on top of Arduino board to let pins on the Bluetooth shield inherits the pins on Arduino. But for this particular Arduino Due board, if we had done the connection this

way, many digital pins on the Due board will be blocked by the shield; and we do need those pins to fulfill our other functions, for instance the pins to connect LCD's MISO,MOSI and SCK were blocked by the shield. Therefore we decided to power up the shield via wires and connect output/input pins as well. Although this connection takes more space, it provides a large range of accessible pins to suit our needs.

2. A proper installation of LCD screen onto breadboard could not be done due to the width of LCD screen. The LCD screen happened to be a bit wider than the width of pin matrix on breadboard; therefore we could not plug in both pin vectors on breadboard. Fortunately, after researching, we found that only connect the pin vector on left is enough to drive the LCD screen. So we plugged in the left half and let the right half hang at edge of the breadboard.

3. In regarding to the detection of whether the car is in parking lot, we used a pair of IR LED and IR sensor. Upon testing if this pair is working, we found that our sensor could not detect the IR light from IR LED. This meant either our LED was not working or our sensor was broken. To test the IR LED, we used a phone camera that turns IR filter off and we saw nothing is emitting from the LED.So we went back to double check our code which drives LED and as well as our circuit connection. We used oscilloscope to measure voltage across LED, the result showed it was in right frequency but Vpp was too low. Due to this, we reduced the resistance of series resistor (voltage divider) to 230 ohms to allow more voltage drop across the LED. After this we used camera again and observed a purple-ish light was emitted from LED but only at a certain angle. We corrected the LED position to ensure the IR light will propagate to IR sensor and successfully detected the IR light.

4. The synchronization issue was the main software problem encountered in this project. In our design, the Arduino Due will send messages to the phone via Bluetooth, and the app will pick up the messages then display on the phone screen. But as they were not synchronized, every time the massage was refreshed on screen, it might lose some part of the original message. For example, Arduino sends “freespace: 1” but only an “eespace: 1” message will appear on the phone. In order to solve this, rather than dig deep to understand both devices' operating rates, we decided to solve this in a much simpler and less time consuming way: adding buffer. Instead of sending “freespace: 1”, we modified our code to send with buffer “ freespace: 1 ”. On the other hand, the code of app was modified to display everything that is not a “space”. For instance, if the Arduino sent “   freespace: 1   ” and the phone received “ freespace: 1   ”, the buffer at front did it work and maintain the main message. After automatically deleting spaces, a message of “freespace: 1” will be displayed on screen. Although this solution is not perfect, it satisfies our need at this stage.

5. Improvements

Our project has a great potential in terms of improvements：
The first that comes to our mind is to use WIFI instead of Bluetooth for communication; this will improves the communication range by a large factor to make this project more practical and utilized.

Secondly, as mentioned in trouble-shooting, dig into the synchronization issue and solve it from a deeper level. This will make our app react faster and improve the stability under load.

Additional improvement for app would be a better looking user interface which is user friendly. In terms of the prototype, we could add more components to achieve more functionality. For example, we could add red and green LED to indicate if there is any available parking spot; use LCD to display more information and even images; create more parking spots; add temperature sensor to detect fire emergency and a speaker to warn people if emergency occurs.

6.Cost:

Component:	Price	quantity	Total
Arduino due	50	1	50
Bluetooth 2.0 shield	20	1	20
Bluetooth 4.0 shield	59	1	59
Servo motor	16	1	16
IR LED	1	1	1
IR sensor	2.5	1	2.5
LCD display	22	1	22
Resistor 220 (ohm)	0.1	2	0.2
Total			170.7(AUD)

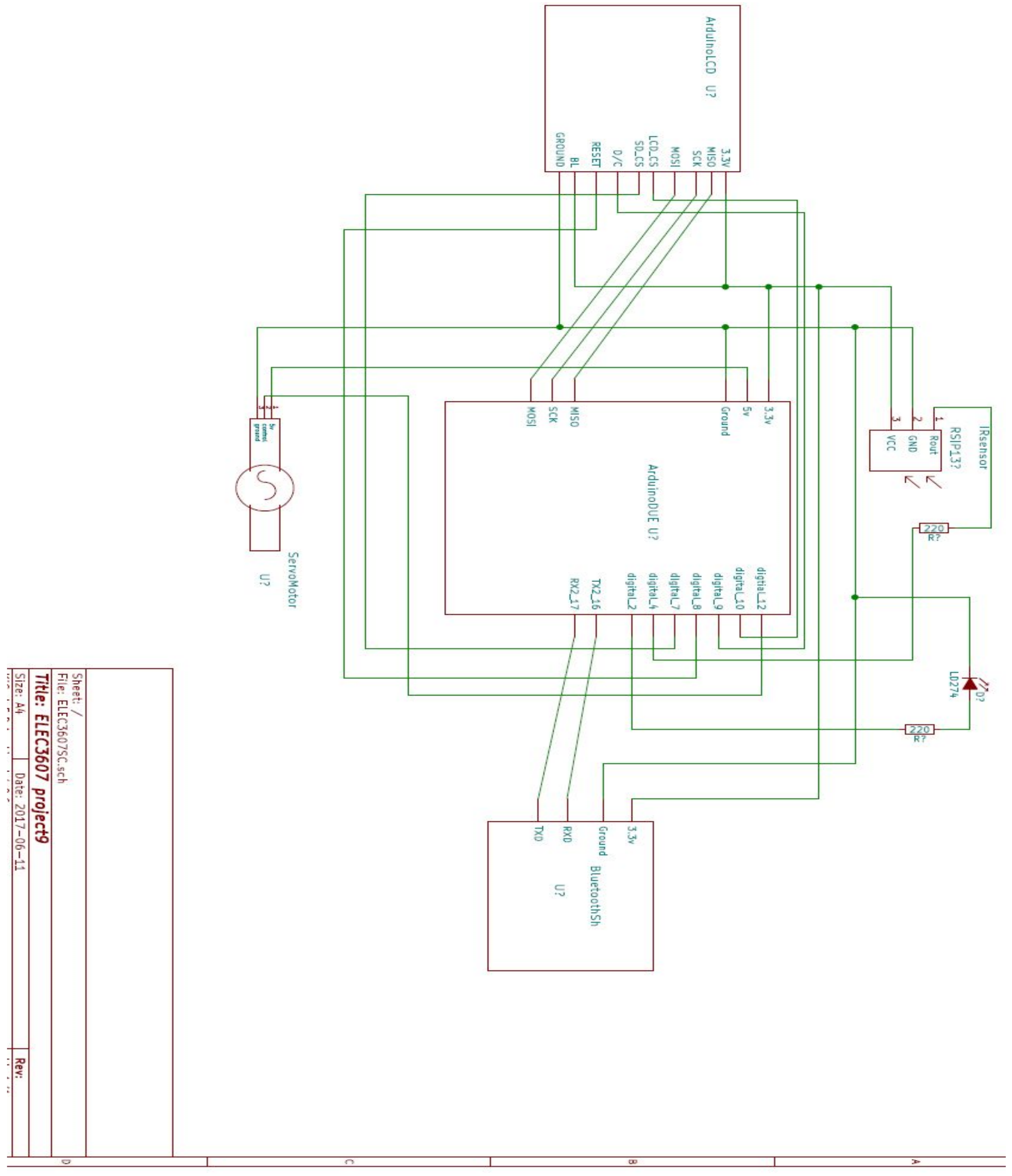
7. Conclusion

In conclusion, we successfully implement a wireless car parking/billing system using Arduino Due and an android phone. Our prototype completes most of our expectations and at some point it even exceeds our plan. (“Pay button” in app replaces the hardware button in the original plan) We worked hard and put efforts into this project as it continuously give us positive feedbacks and joy. This experience is valuable to us because it is like a bridge links the theorem to reality. By designing and completing this project, we have gain experience on how to apply our knowledge to a real world problem/application. Although this project is rather simple than anything that is worthy to talk about, it is a great start for understanding embedded systems. Additionally, this project reinforced our team-work skills such as communication skills as it requires a team work.

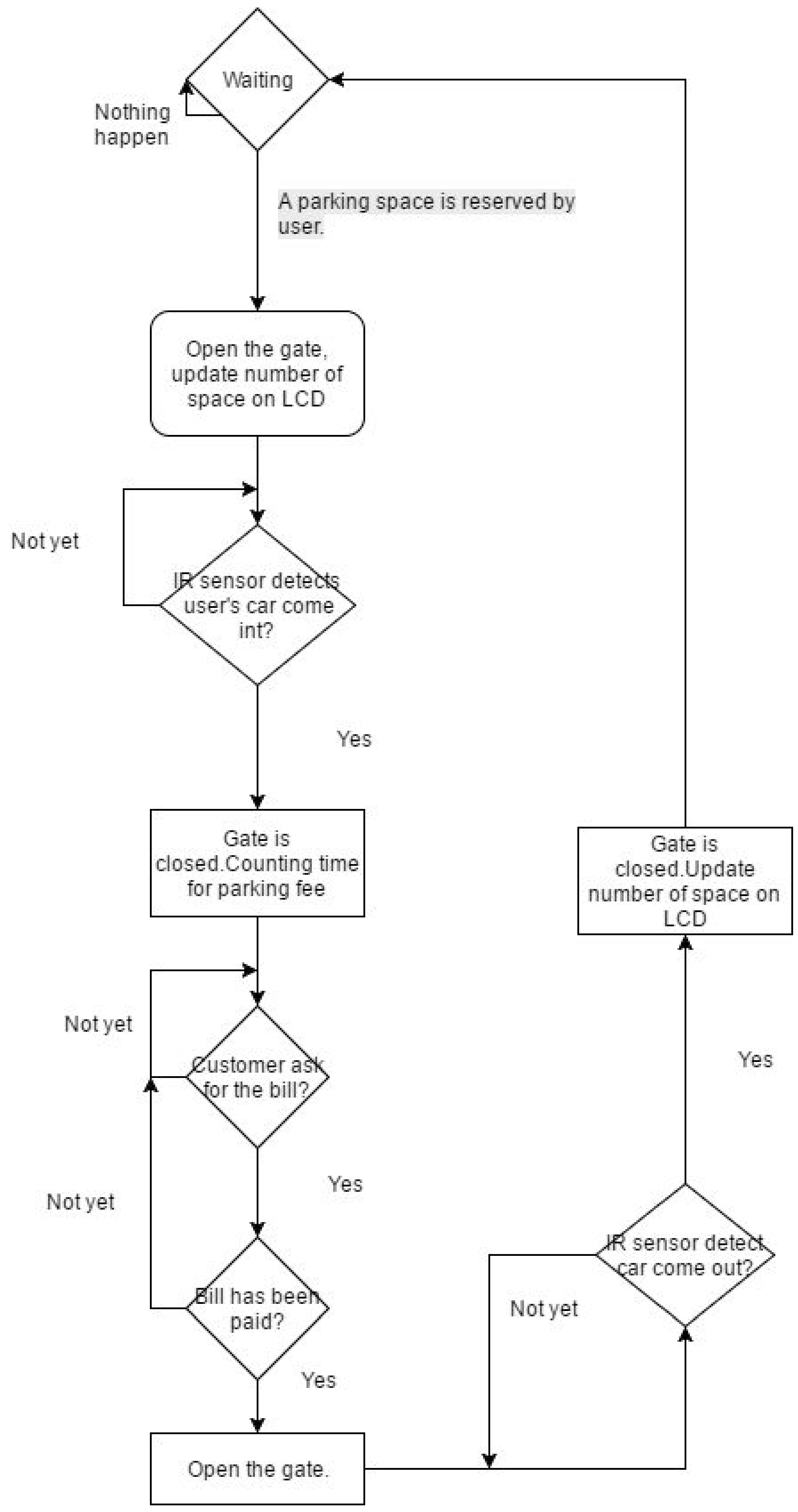
Open Source to GITHUB : https://github.com/qgua2322/ELEC3607_carparkproject

Reference :
1.Arduino pin map: https://www.arduino.cc/en/Main/ArduinoBoardDue
2.Arduino due schematic:https://www.arduino.cc/en/uploads/Main/arduino-Due-schematic.pdf
3.LCD connection:https://www.arduino.cc/en/Guide/TFTtoBoards
4.LCD display:https://www.arduino.cc/en/Reference/TFTLibrary
5.FSM : https://flip.ee.usyd.edu.au/seiewiki/Labs/interrupts
6.Bluetooth2.0: http://wiki.seeedstudio.com/wiki/Bluetooth_Shield
7.Bluetooth4.0:
http://linksprite.com/wiki/index.php5?title=Bluetooth_4.0_BLE_Pro_Shield_for_Arduino_(Master/Slave_and_iBeacon)
8.Kicad tutorial: https://www.youtube.com/watch?v=o6IWdPpZ4mc
9.Android tutorial:https://developer.android.com/training/index.html
10.Android control APP: http://www.instructables.com/id/Android-Bluetooth-Control-LED-Part-2/

Appendix 1 : Schematic by Kicad
(Please go to github for a better version)



Appendix 2: software flow chart


Waiting
Nothing happen
A parking space is reserved by user.
Open the gate, update number of space on LCD
Not yet
IR sensor detects user's car come int?
Yes
Gate is closed.Counting time for parking fee
Not yet
Customer ask for the bill?
Yes
Not yet
Bill has been paid?
Yes
Open the gate.
Gate is closed.Update number of space on LCD
Yes
IR sensor detect car come out?
Not yet

Appendix 3: Arduino code:

```
/*
 *   This is a Car Parking Embedded System by using Arduino Due. This system is able to
 *   communicate with customer by Bluetooth(2.0 or 4.0). In our design, customers can open their own's Apps
 *   to check how many spaces are still available/reserve a parking space/ how long did they parked
 *   in/Pay the bill. So this embedded system need to be able to response customer's command, and manage
 *   this car park automatically. For example, when a customer is reserve a space, this system need to
 *   update the number of spaces is remain on LCD and customer's phone, and open the gate of that parking space,
 *   and the gate will not close until the customer's car park in.
 *
 *   System design:
 *   This system is using a Finite state machine(FSM) as the control mechanism.
 *   Input: Bluetooth and IR sensor;
 *   Output: IR LED, servo motor,bluetooth, LCD display.
 *
 *   Using different version of Bluetooth:
 *   First,Bluetooth's RX connects to Serial2 TX, Bluetooth's TX connects to Serial2 RX. (For any version)
 *   Second, if choose Bluetooth 4.0, uncommand setupBlueToothConnection4(), command setupBlueToothConnection()
 *           if choose Bluetooth 2.0, uncommand setupBlueToothConnection(), command setupBlueToothConnection4().
 *
 *
 *   Created 10/06/2017 by QISHEN GUAN
 */



//Include Arduino library
#include <TFT.h>
#include <SPI.h>
#include <Servo.h>

// pin definition for the LCD
#define cs   10
#define dc   9
#define rst  8

//Bluetooth definition
#define blueToothSerial Serial2

#define XTC TC1
#define XCHAN 1
#define XID ID_TC4

//IR LED signal definition
#define   XTC1     TC0        // TC number
#define   XCHAN1   0          // TC channel
#define   XID1     ID_TC0     // Instance ID
#define   PINCHAN   25       // Digital pin 2 which is on PIOB channel 25

//Create Servo and TFT object
Servo myservo;
TFT TFTscreen = TFT(cs, dc, rst);


/*  FSM States */
typedef enum
{
  IDLE = 0U,
  RESERVESend,
  RESERVECheck,
  RESERVEClose,
  TIMECounting,
  LEAVE,
  PAY,
  PAYCHECK,
  PAYCLOSE,
} ParkingSystem_State_t;


/* FSM variables */
ParkingSystem_State_t txState = IDLE; // State register
int statusreg = 0;  //FSM intrreput  signal

int freespace = 1;
char spaceN[7];
```

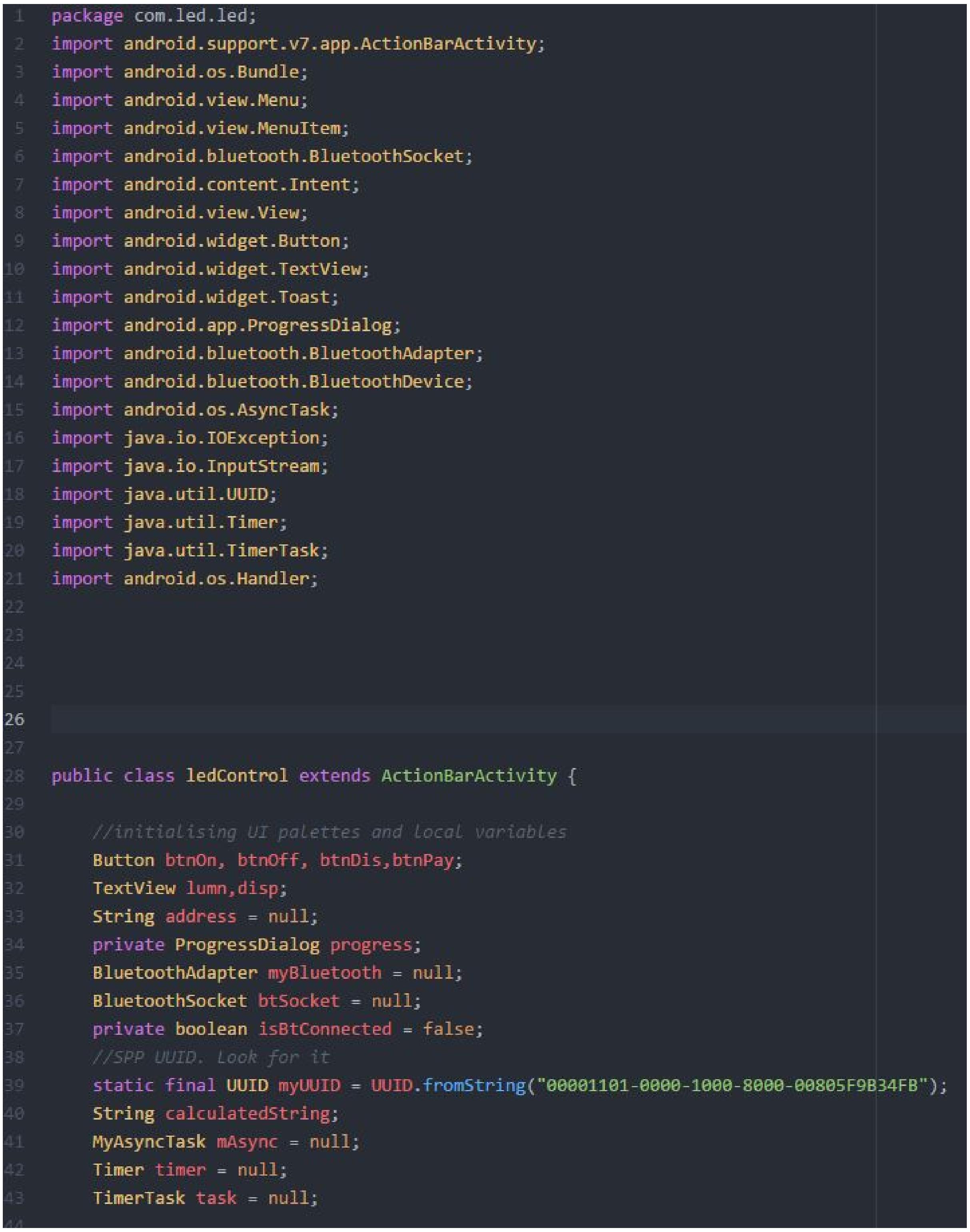
```
String spacebuff ="";

int timeL = 0 ;
String timebuff = "";
char timeLC[7];

int payment  = 0;
String paybuff = "";
char payA[10];

int detection  = 0 ;


/*  We choose FSM as control mechanism of this system, because in FSM we dont need to
 *  worry about customer mispress some wrong button(Except "Disconnect"),then the system
 *  get into a unkown situation. Nothing will happen when customer press "pay" before "leave".
 *  Thing only procese in a sequence that we wanted.
 */
void ParkingSystem_state_machine(){
  switch(txState){

    case RESERVESend:
        blueToothSerial.print("    Freespace:");
        blueToothSerial.print(freespace);
        blueToothSerial.println("  Counter time after u park in ");
        opendoor();
        txState = RESERVECheck;
        break;

     case RESERVECheck:
        detection = digitalRead(4);
        txState = detection !=1 ? RESERVECheck : RESERVEClose; //Goto next state when car parks in
        break;

    case RESERVEClose:
        closedoor();

       [illegible]
       REG_TC1_CCR1 = 1<<2; //Reset counter register
       TC_Start(XTC,XCHAN); //Begin counting
       txState = TIMECounting;
       break;

   case TIMECounting:
       timeL = double(TC_ReadCV(XTC,XCHAN))/VARIANT_MCK*128;     //Read time from timer counter register
       timebuff ="";
       timebuff += timeL;
       timebuff.toCharArray(timeLC,7);    //Cast it into LCD form.

       TFTscreen.text("Time:", 0, 40); //Write on LCD
       TFTscreen.text(timeLC, 100, 40);
       blueToothSerial.print("    Time:"); //Write on Customer's App
       blueToothSerial.print(timeL);
       blueToothSerial.println("  ");

       txState = statusreg != 2 ? TIMECounting : LEAVE; //Goto next state when customer press "leave"
       break;

   case LEAVE:

       payment = 10*timeL;
       paybuff ="  $";
       paybuff += payment;
       paybuff.toCharArray(payA,10);

       TFTscreen.text("payment:", 0, 40);
       TFTscreen.text(payA, 50, 60);
       blueToothSerial.print("    Payment:");
       blueToothSerial.println(payA);
       txState =statusreg != 3 ? LEAVE : PAY;
       [illegible]
```

```
case PAY:

    blueToothSerial.print("       ");
    blueToothSerial.println("Payment procesing");
    blueToothSerial.print("       ");
    blueToothSerial.println("Payment comfired. Good to go.");
    opendoor();
    TFTscreen.text("Thank for.", 0, 40);
    TFTscreen.text("coming.", 60, 60);
    txState = PAYCHECK;

    break;

case PAYCHECK:
    detection = digitalRead(4);
    txState = detection != 0 ? PAYCHECK : PAYCLOSE;
    break;

case PAYCLOSE:
     blueToothSerial.print("   ");
     blueToothSerial.println("Thank for coming");
     closedoor();
     txState = IDLE;
     break;

case IDLE:
    blueToothSerial.print("    Freespace: ");
    blueToothSerial.print(freespace);
    blueToothSerial.println("  ");

    Serial.println("IDLE");
    txState = statusreg != 1 ? IDLE : RESERVESend;
    break;
    default:
        break;
  }
}




void setup()
{
    Serial.begin(9600);

    pmc_set_writeprotect(false);

    setupBlueToothConnection(); //Bluetooth 2.0
    //setupBlueToothConnection4(); //Bluetooth 4.0

    TFTscreen.begin();
    TFTscreen.background(0, 0, 0);   // clear the screen with a black background
    TFTscreen.stroke(255, 255, 255); // set the font color to white
    TFTscreen.setTextSize(2);        // set the font size

    //Set up timer counter to counte how long does customer parked in.
    pmc_enable_periph_clk(XID);
    TC_Configure(XTC,XCHAN,TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC | TC_CMR_TCCLKS_TIMER_CLOCK4);

    //Connect digital pin2 to another timer counter to generate a square wave
    unsigned int chanmask = (1 << PINCHAN);
    REG_PIOB_PDR = chanmask;
    REG_PIOB_ABSR = chanmask;
    REG_PIOB_MDDR = chanmask;
    squarewave(36000);

    //Setup for IR sensor
    pinMode(4,INPUT);
}
```

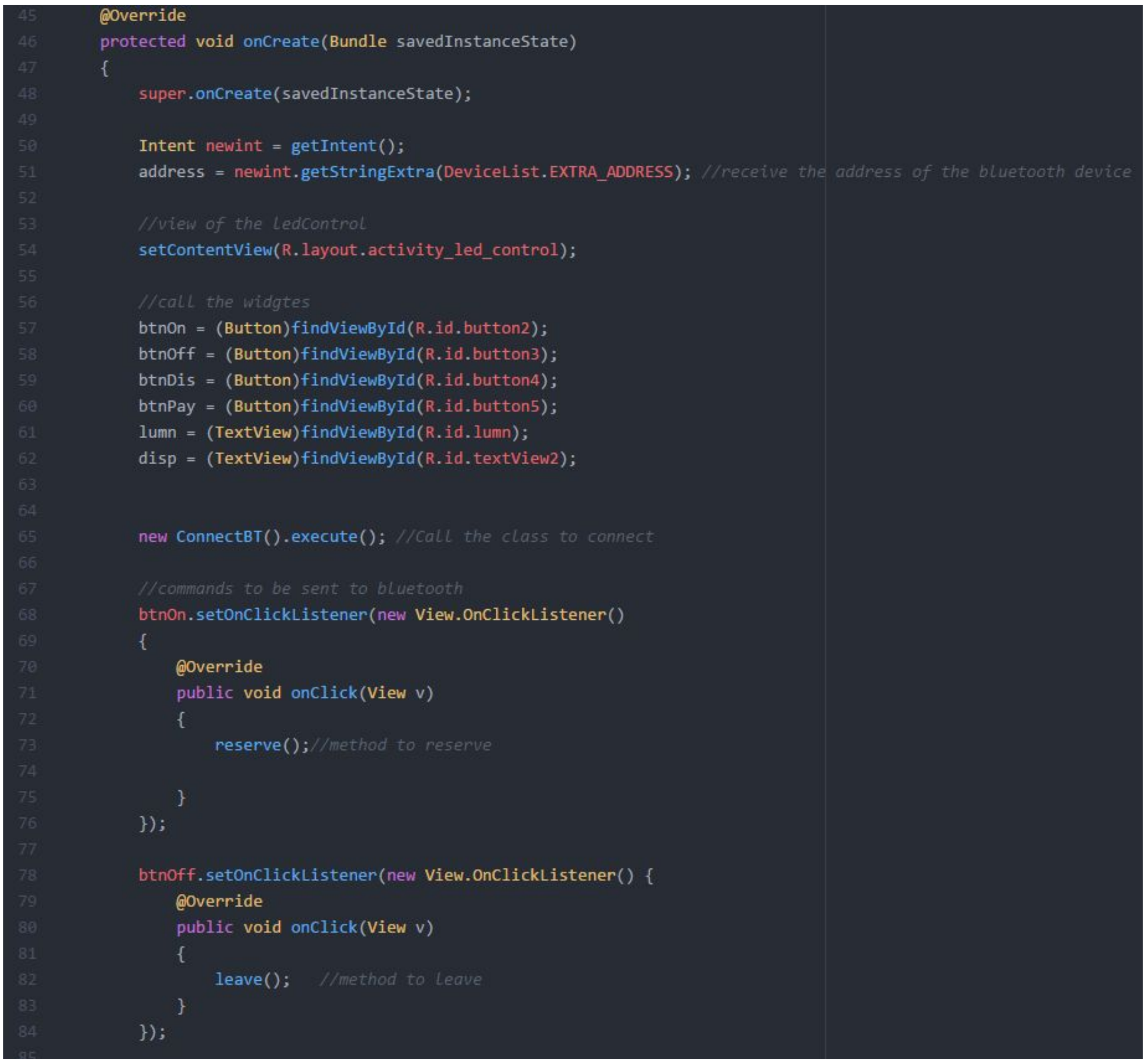
```
void loop()
{
    String recvStr;
    String sendStr;


    while(1){
        if(blueToothSerial.available()){
          //check if there's any data sent from the remote bluetooth shield
            recvStr = blueToothSerial.readStringUntil(',');
            Serial.println(recvStr);
            //Change the interrput signal  when customer press a button
            if(recvStr =="reserve"){
              freespace -= 1;
              statusreg =1;
            }

            if(recvStr == "leave"){
              statusreg = 2;
            }

            if(recvStr == "pay"){
              freespace += 1;
              statusreg = 3 ;
            }


        }
        if(Serial.available()){ //For admin to send some message to customer
          sendStr = Serial.readString();
          blueToothSerial.print("");
          blueToothSerial.println(sendStr);
        }

        TFTscreen.stroke(255, 255, 255);
        spacebuff += freespace;
        spacebuff.toCharArray(spaceN,7);
        TFTscreen.text("Freespace:", 0, 20);  //keep on showing free spaces on LCD
        TFTscreen.text(spaceN, 120, 20);

        ParkingSystem_state_machine(); //Call the FSM
        delay(1000);

        TFTscreen.background(0,0,0);
        // erase the text you just wrote
        TFTscreen.stroke(0, 0, 0);
    }
}

/*
 * Bluetooth 2.0 setup
 */
void setupBlueToothConnection(){
    blueToothSerial.begin(38400);                          // Set BluetoothBee BaudRate to default baud rate 38400
    blueToothSerial.print("\r\n+STWMOD=0\r\n");            // set the bluetooth work in slave mode
    blueToothSerial.print("\r\n+STNA=ProjectDemo\r\n");   // set the bluetooth name as "SeeedBTSlave"
    blueToothSerial.print("\r\n+STOAUT=1\r\n");            // Permit Paired device to connect me
    blueToothSerial.print("\r\n+STAUTO=0\r\n");            // Auto-connection should be forbidden here
    delay(2000);                                           // This delay is required.
    blueToothSerial.print("\r\n+INQ=1\r\n");               // make the slave bluetooth inquirable
    Serial.println("The slave bluetooth is inquirable!");
    delay(2000);                                           // This delay is required.
    blueToothSerial.flush();
}


 /*
  * Bluetooth 4.0 setup (AT mode)
  */
 void setupBlueToothConnection4(){
     blueToothSerial.begin(38400);                        // Set BluetoothBee BaudRate to default baud rate 38400
     blueToothSerial.print("AT+ROLE=0");                  //slave mode
     blueToothSerial.print("AT+BAUD8");
     blueToothSerial.print("AT+PIN1234");                 //set pin 1234
     blueToothSerial.print("AT+NAMEBT4");                  // set the bluetooth name as "BT4"
     delay(2000);                                         // This delay is required.
     blueToothSerial.flush();

 }


 /*This function is control servo motor turn backward by half second to open the gate.
  *
  */
 void opendoor(){
   myservo.attach(12);  //Motor is connected with digital pin 12
   myservo.write(120);  //Turn backward
   delay(500);
   myservo.write(90);  //Stop the motor
 }

 /*This function is control servo motor turn forward by half second to open the gate.
  *
  */
 void closedoor(){
   myservo.attach(12);
   myservo.write(30);
   delay(500);
   myservo.write(90);
 }
```

```
/* This function is to generate a certain frequency squarewave signal by using timer counter.
 *     _ _ _ _         _ _ _ _
 *    |       |       |       |
 * ___|       |_______|       |__
 *
 *    |<tcclk>|                     tcclk = Time frequency/desired frequency
 *                                  Beacuase CLOCK1 = MCK/2, and square wave only on for half cycle.
 *                                  tcclk = (MCK/2)/freq/2
 *
 *                                  TC_CMR_WAVE: waveform mode on
 *                                  TC_CMR_WAVSEL_UP_RC: UP mode with automatic trigger on RC Compare
 *                                  TC_CMR_TCCLKS_TIMER_CLOCK1: Time frequency = MCK/2
 *                                  TC_CMR_ACPC_TOGGLE: RC compare effect on TIO
 */
void squarewave(unsigned int freq) {
  pmc_enable_periph_clk(XID1);
  unsigned int  tcclk = VARIANT_MCK / freq / 4;
  TC_Configure(XTC1, XCHAN1, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC | TC_CMR_TCCLKS_TIMER_CLOCK1 | TC_CMR_ACPC_TOGGLE);
  TC_SetRC(XTC1, XCHAN1, tcclk);
  TC_Start(XTC1, XCHAN1);
}
```

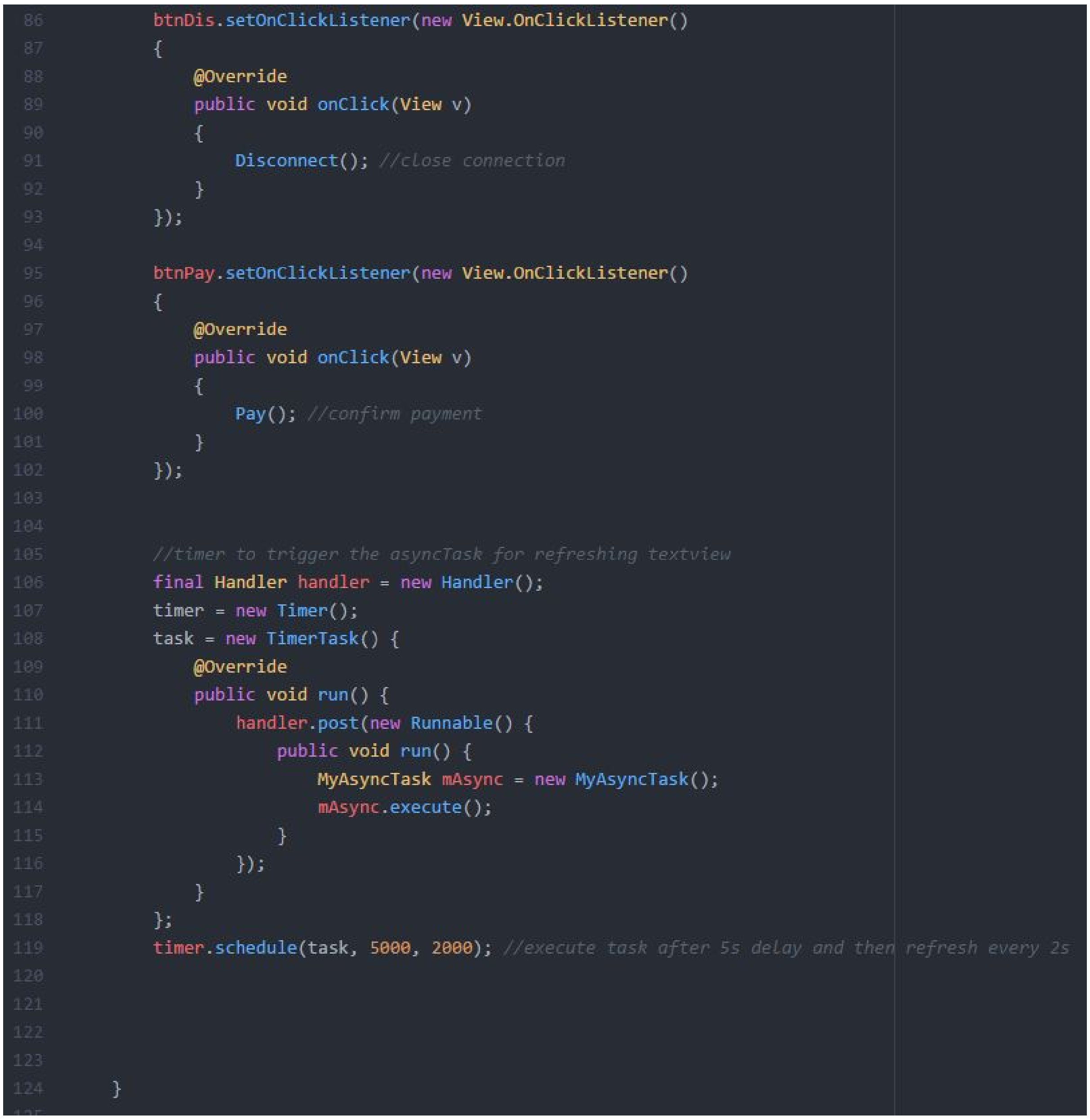
Appendix 4: Android code: Control.java



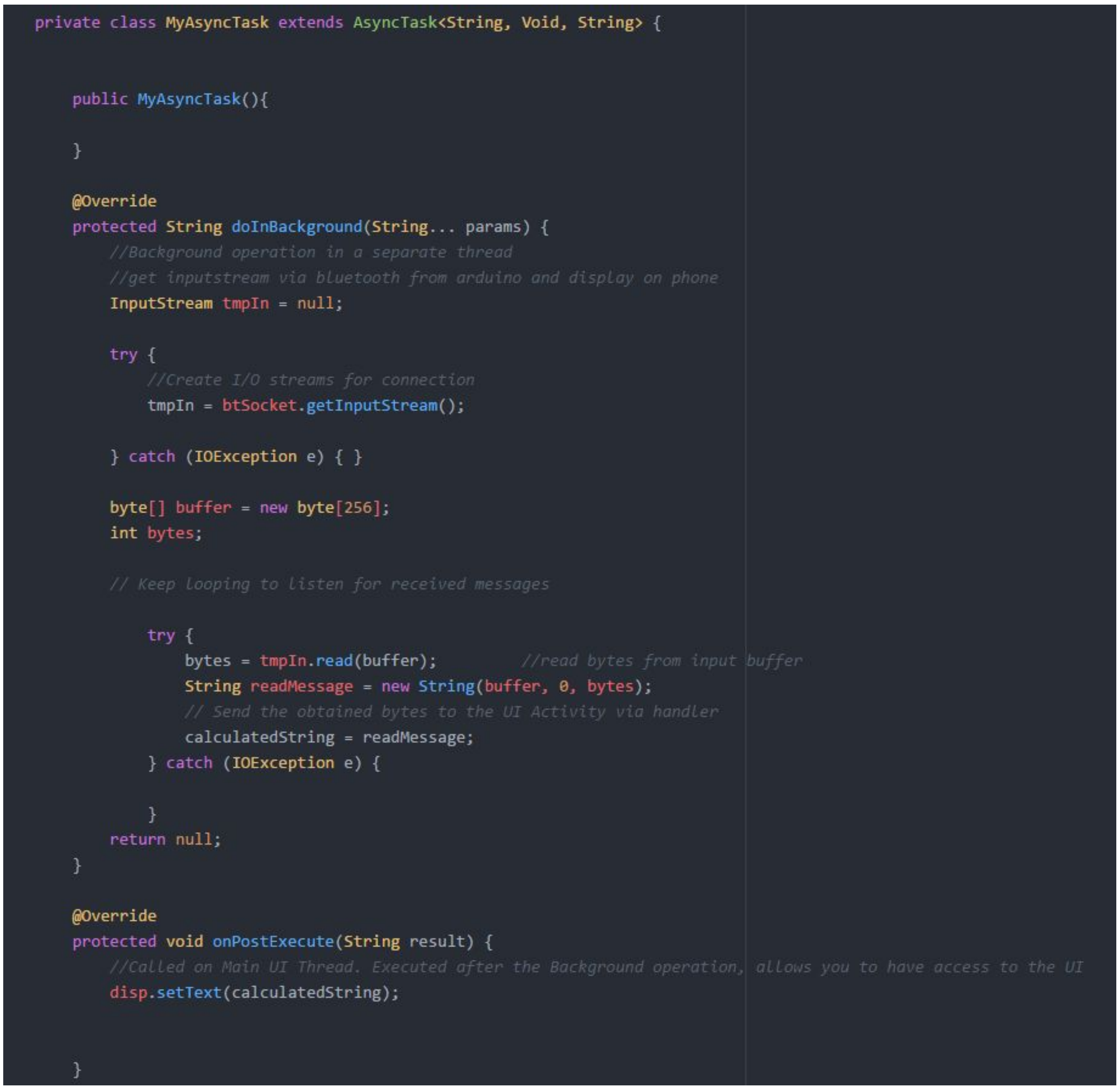
```java
package com.led.led;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.os.AsyncTask;
import java.io.IOException;
import java.io.InputStream;
import java.util.UUID;
import java.util.Timer;
import java.util.TimerTask;
import android.os.Handler;






public class ledControl extends ActionBarActivity {

    //initialising UI palettes and local variables
    Button btnOn, btnOff, btnDis,btnPay;
    TextView lumn,disp;
    String address = null;
    private ProgressDialog progress;
    BluetoothAdapter myBluetooth = null;
    BluetoothSocket btSocket = null;
    private boolean isBtConnected = false;
    //SPP UUID. Look for it
    static final UUID myUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
    String calculatedString;
    MyAsyncTask mAsync = null;
    Timer timer = null;
    TimerTask task = null;
```



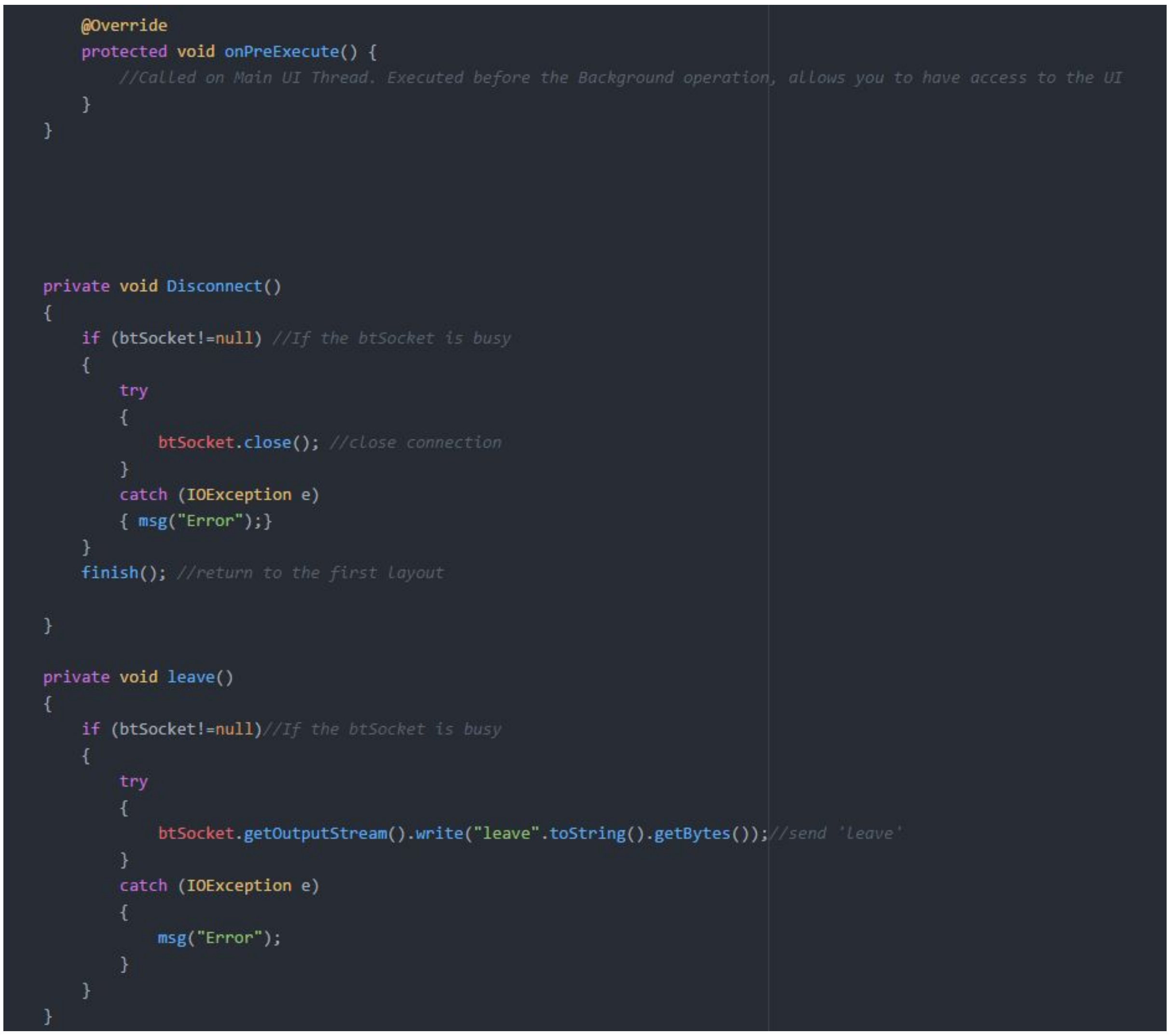
```
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        Intent newint = getIntent();
        address = newint.getStringExtra(DeviceList.EXTRA_ADDRESS); //receive the address of the bluetooth device

        //view of the ledControl
        setContentView(R.layout.activity_led_control);

        //call the widgtes
        btnOn = (Button)findViewById(R.id.button2);
        btnOff = (Button)findViewById(R.id.button3);
        btnDis = (Button)findViewById(R.id.button4);
        btnPay = (Button)findViewById(R.id.button5);
        lumn = (TextView)findViewById(R.id.lumn);
        disp = (TextView)findViewById(R.id.textView2);


        new ConnectBT().execute(); //Call the class to connect

        //commands to be sent to bluetooth
        btnOn.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                reserve();//method to reserve

            }
        });

        btnOff.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v)
            {
                leave();   //method to leave
            }
        });
```



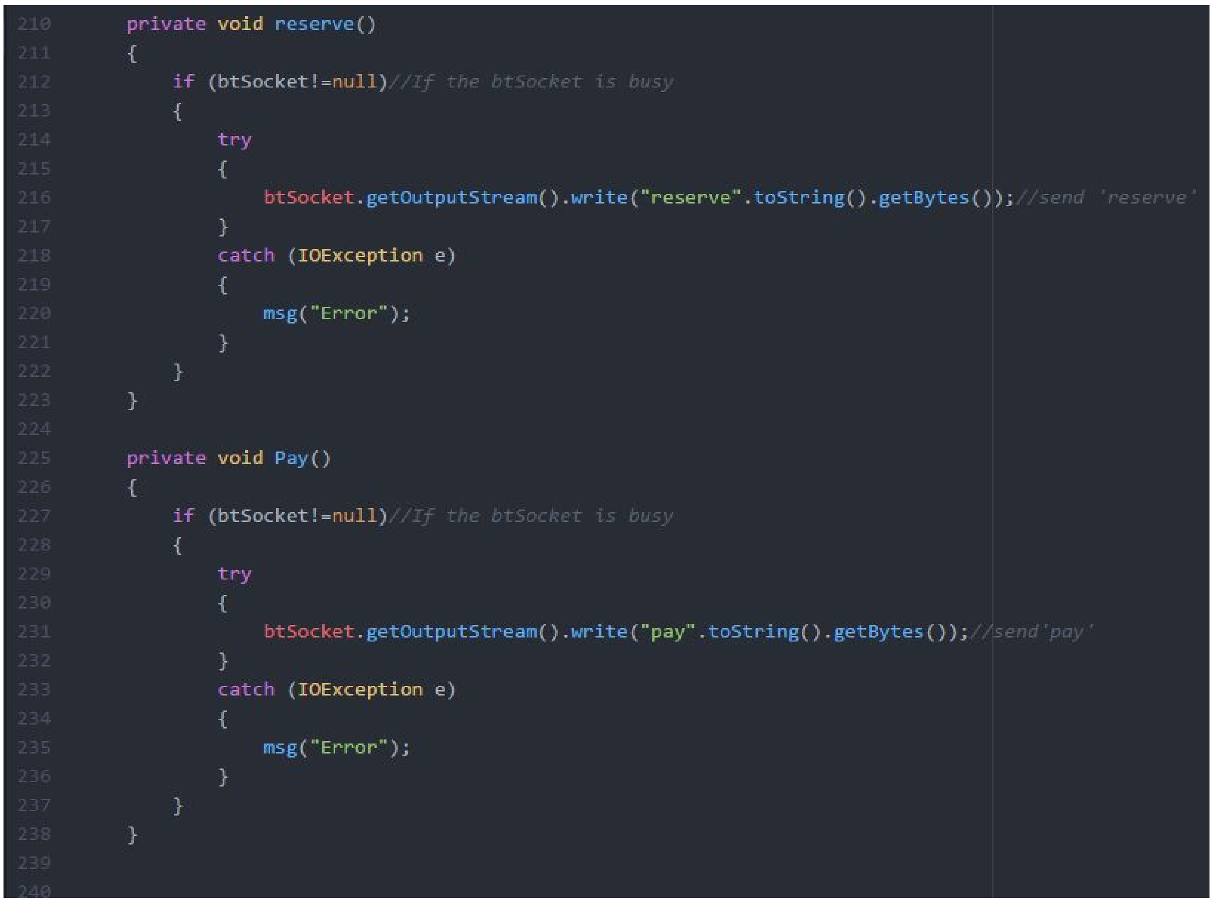
```java
        btnDis.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                Disconnect(); //close connection
            }
        });

        btnPay.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                Pay(); //confirm payment
            }
        });


        //timer to trigger the asyncTask for refreshing textview
        final Handler handler = new Handler();
        timer = new Timer();
        task = new TimerTask() {
            @Override
            public void run() {
                handler.post(new Runnable() {
                    public void run() {
                        MyAsyncTask mAsync = new MyAsyncTask();
                        mAsync.execute();
                    }
                });
            }
        };
        timer.schedule(task, 5000, 2000); //execute task after 5s delay and then refresh every 2s




    }
```



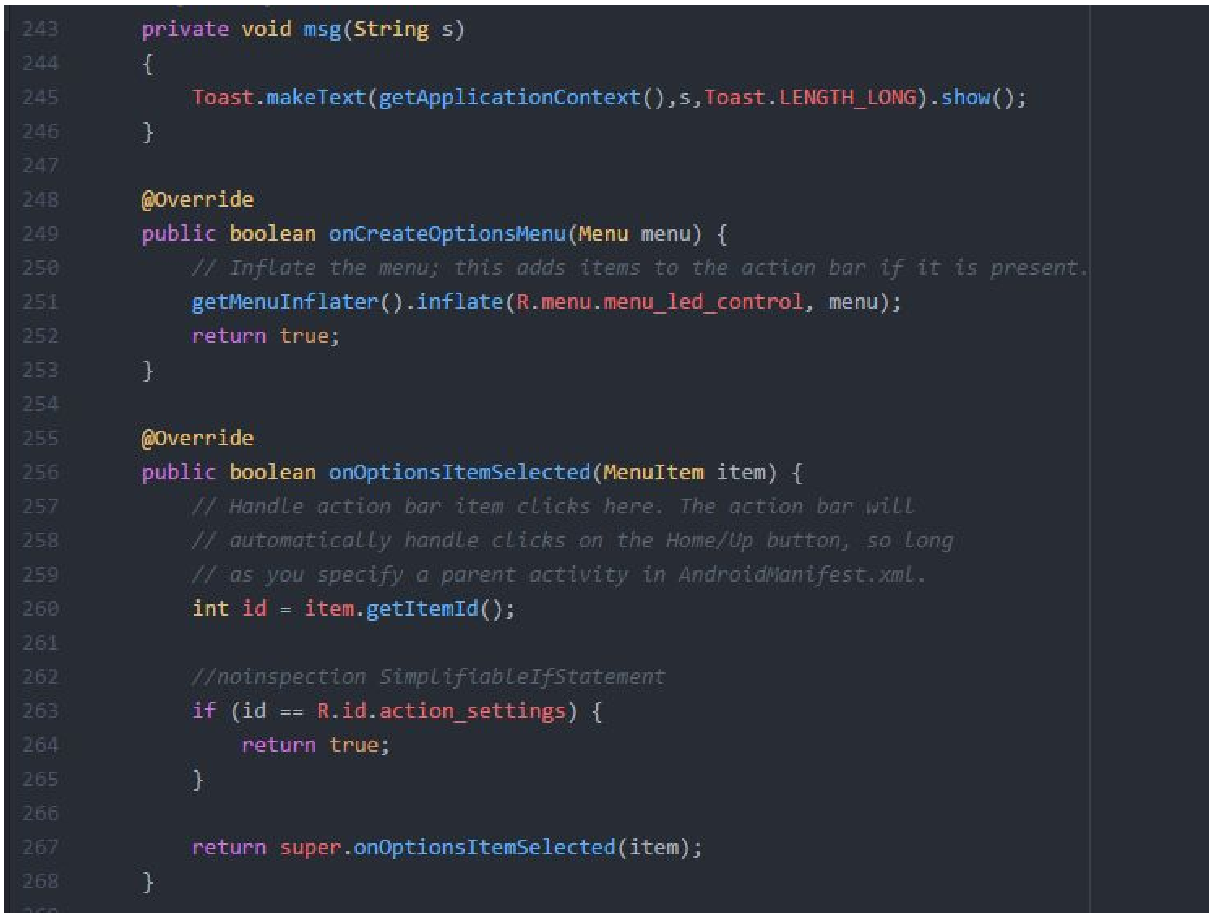
```java
private class MyAsyncTask extends AsyncTask<String, Void, String> {


    public MyAsyncTask(){

    }

    @Override
    protected String doInBackground(String... params) {
        //Background operation in a separate thread
        //get inputstream via bluetooth from arduino and display on phone
        InputStream tmpIn = null;

        try {
            //Create I/O streams for connection
            tmpIn = btSocket.getInputStream();

        } catch (IOException e) { }

        byte[] buffer = new byte[256];
        int bytes;

        // Keep looping to listen for received messages

            try {
                bytes = tmpIn.read(buffer);         //read bytes from input buffer
                String readMessage = new String(buffer, 0, bytes);
                // Send the obtained bytes to the UI Activity via handler
                calculatedString = readMessage;
            } catch (IOException e) {

            }
        return null;
    }

    @Override
    protected void onPostExecute(String result) {
        //Called on Main UI Thread. Executed after the Background operation, allows you to have access to the UI
        disp.setText(calculatedString);


    }
```



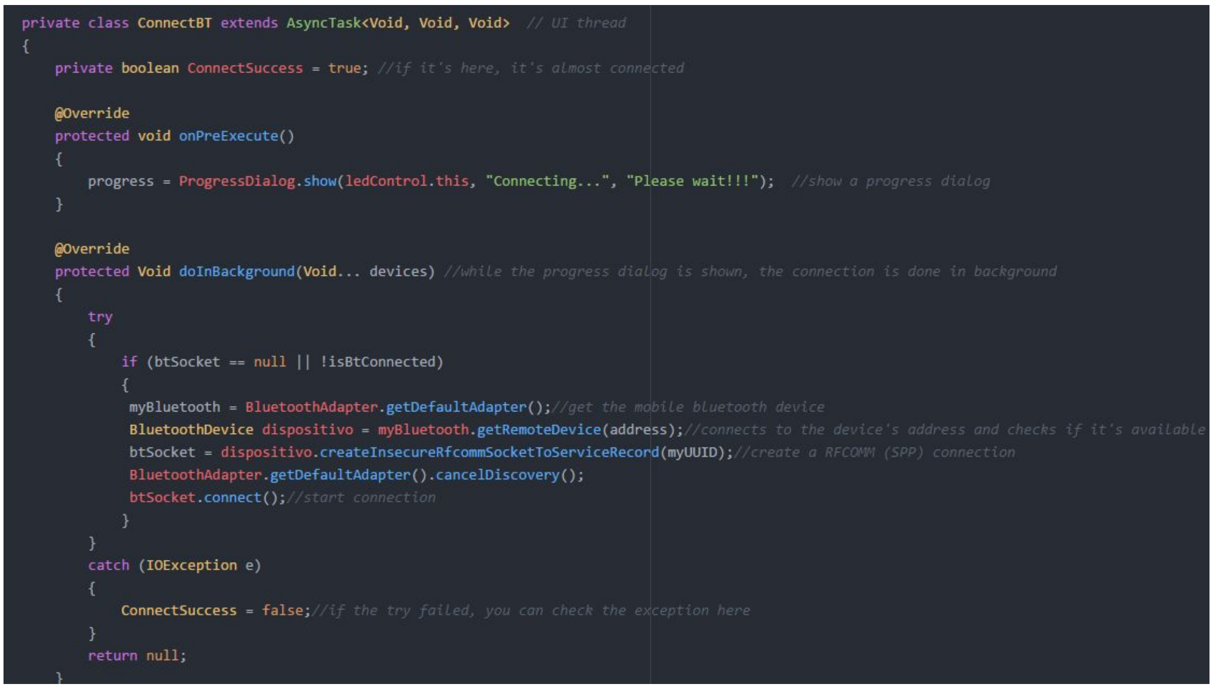
```
        @Override
        protected void onPreExecute() {
            //Called on Main UI Thread. Executed before the Background operation, allows you to have access to the UI
        }
    }




    private void Disconnect()
    {
        if (btSocket!=null) //If the btSocket is busy
        {
            try
            {
                btSocket.close(); //close connection
            }
            catch (IOException e)
            { msg("Error");}
        }
        finish(); //return to the first layout

    }

    private void leave()
    {
        if (btSocket!=null)//If the btSocket is busy
        {
            try
            {
                btSocket.getOutputStream().write("leave".toString().getBytes());//send 'leave'
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }
    }
```



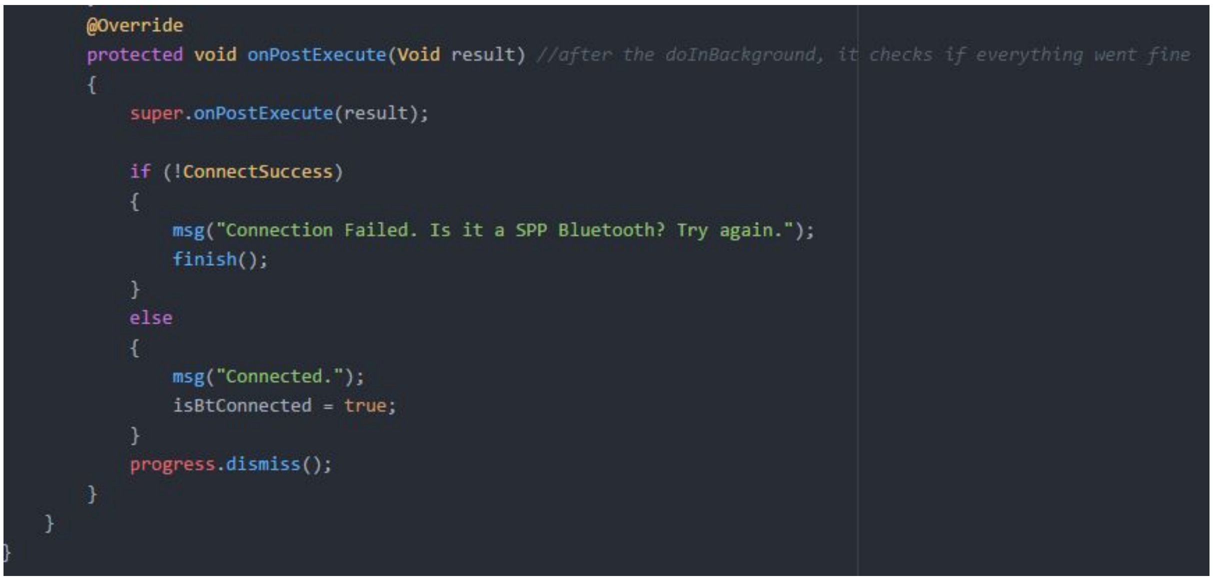
```java
    private void reserve()
    {
        if (btSocket!=null)//If the btSocket is busy
        {
            try
            {
                btSocket.getOutputStream().write("reserve".toString().getBytes());//send 'reserve'
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }
    }

    private void Pay()
    {
        if (btSocket!=null)//If the btSocket is busy
        {
            try
            {
                btSocket.getOutputStream().write("pay".toString().getBytes());//send'pay'
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }
    }

```



```java
    private void msg(String s)
    {
        Toast.makeText(getApplicationContext(),s,Toast.LENGTH_LONG).show();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_led_control, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
```

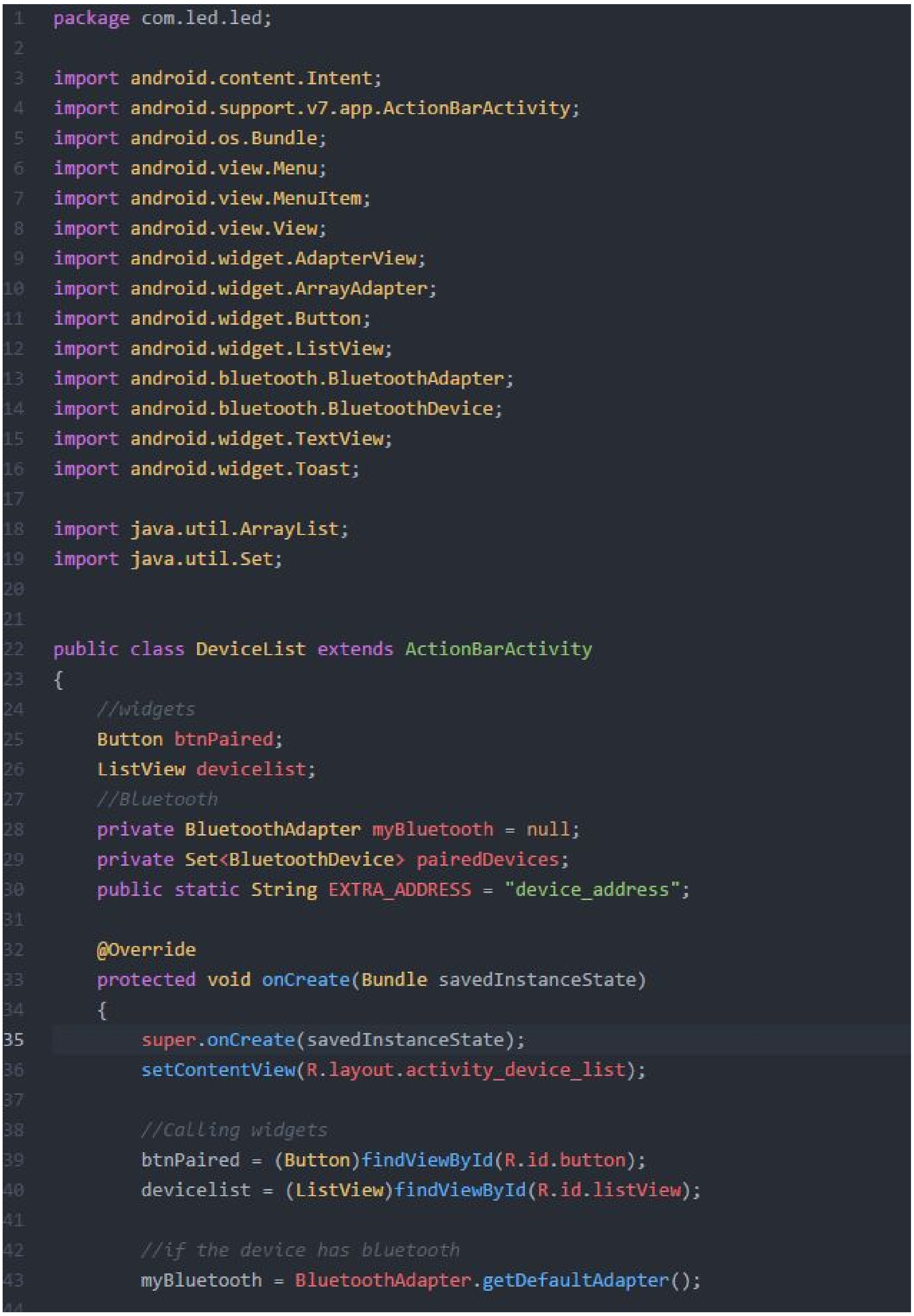


```java
private class ConnectBT extends AsyncTask<Void, Void, Void>  // UI thread
{
    private boolean ConnectSuccess = true; //if it's here, it's almost connected

    @Override
    protected void onPreExecute()
    {
        progress = ProgressDialog.show(ledControl.this, "Connecting...", "Please wait!!!");  //show a progress dialog
    }

    @Override
    protected Void doInBackground(Void... devices) //while the progress dialog is shown, the connection is done in background
    {
        try
        {
            if (btSocket == null || !isBtConnected)
            {
             myBluetooth = BluetoothAdapter.getDefaultAdapter();//get the mobile bluetooth device
             BluetoothDevice dispositivo = myBluetooth.getRemoteDevice(address);//connects to the device's address and checks if it's available
             btSocket = dispositivo.createInsecureRfcommSocketToServiceRecord(myUUID);//create a RFCOMM (SPP) connection
             BluetoothAdapter.getDefaultAdapter().cancelDiscovery();
             btSocket.connect();//start connection
            }
        }
        catch (IOException e)
        {
            ConnectSuccess = false;//if the try failed, you can check the exception here
        }
        return null;
    }
```

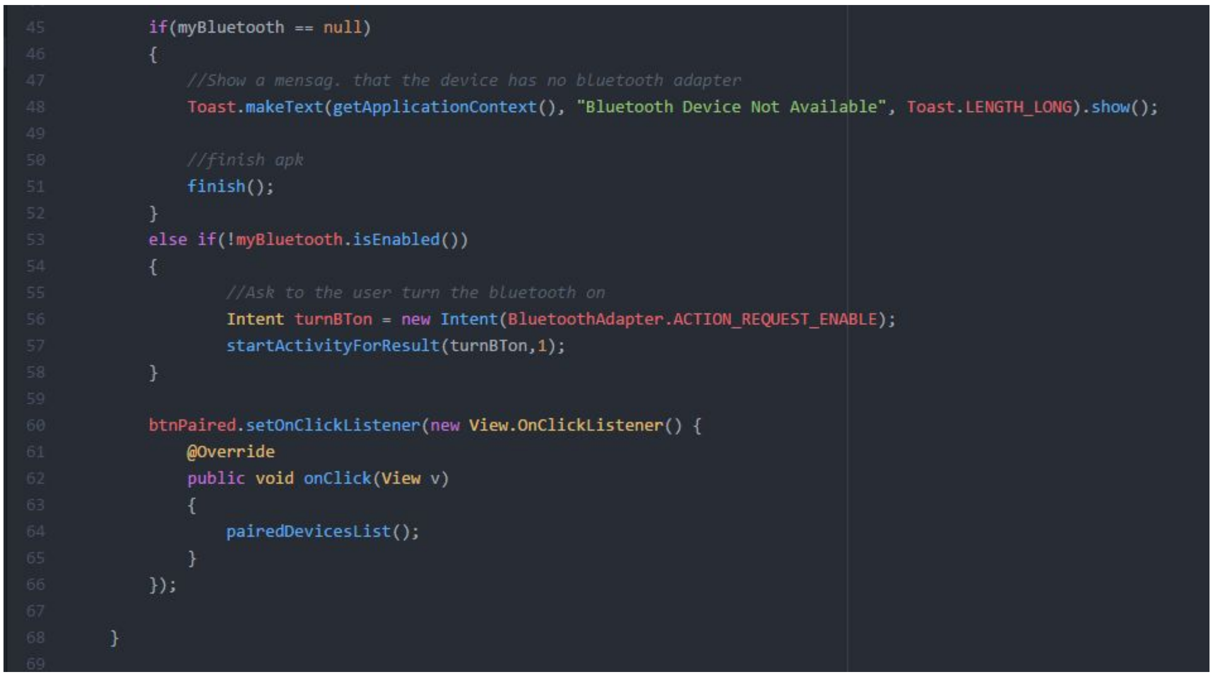


```java
    @Override
    protected void onPostExecute(Void result) //after the doInBackground, it checks if everything went fine
    {
        super.onPostExecute(result);

        if (!ConnectSuccess)
        {
            msg("Connection Failed. Is it a SPP Bluetooth? Try again.");
            finish();
        }
        else
        {
            msg("Connected.");
            isBtConnected = true;
        }
        progress.dismiss();
    }
}
}
```

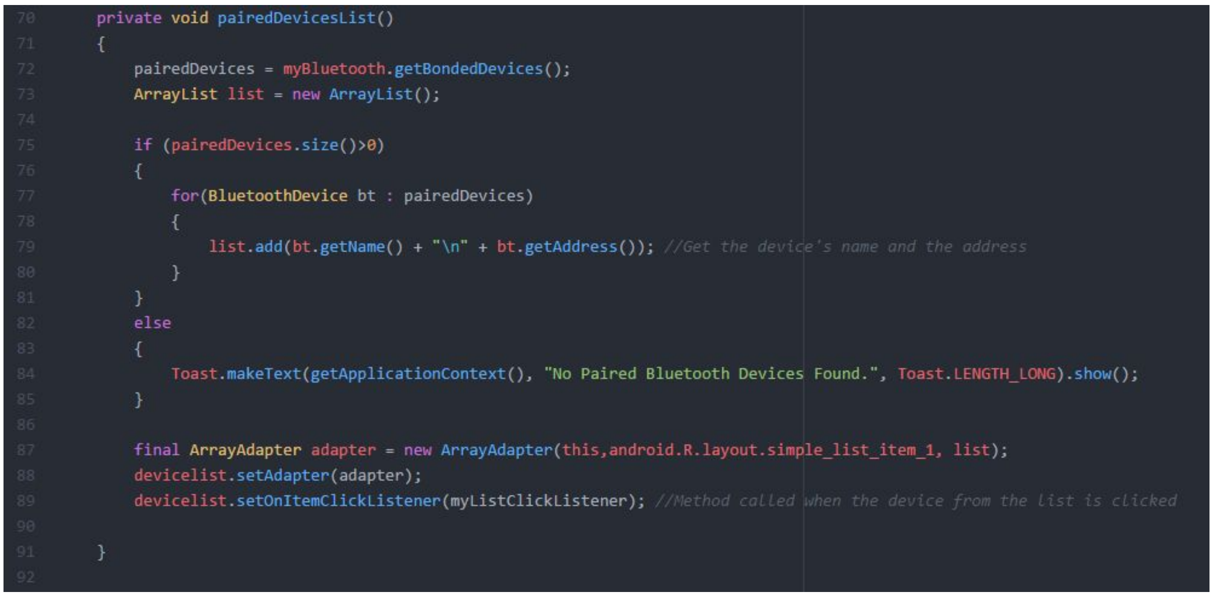
Android code: DeviceList.java



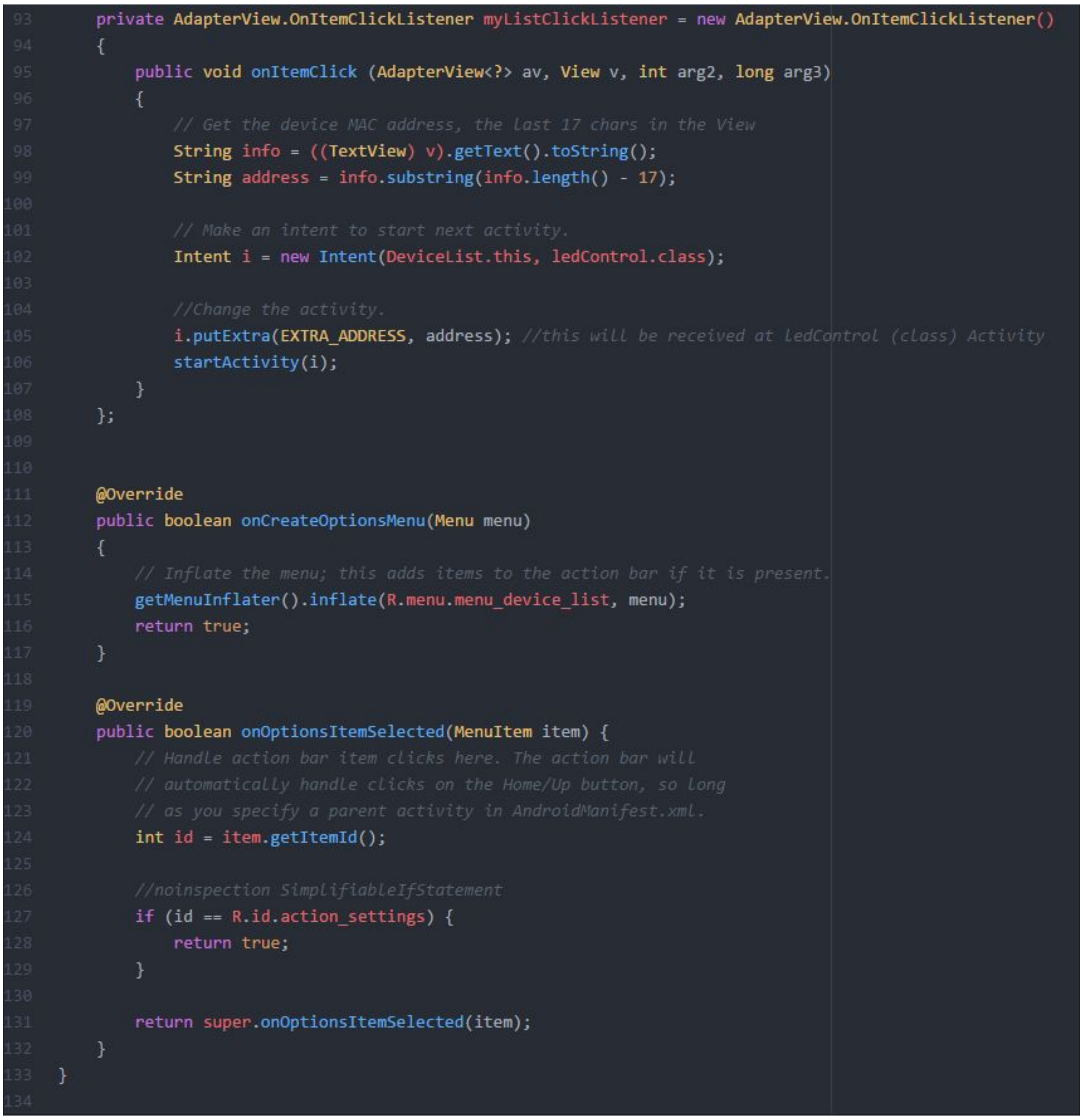
```java
package com.led.led;

import android.content.Intent;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.Set;


public class DeviceList extends ActionBarActivity
{
    //widgets
    Button btnPaired;
    ListView devicelist;
    //Bluetooth
    private BluetoothAdapter myBluetooth = null;
    private Set<BluetoothDevice> pairedDevices;
    public static String EXTRA_ADDRESS = "device_address";

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

        //Calling widgets
        btnPaired = (Button)findViewById(R.id.button);
        devicelist = (ListView)findViewById(R.id.listView);

        //if the device has bluetooth
        myBluetooth = BluetoothAdapter.getDefaultAdapter();
```



```
        if(myBluetooth == null)
        {
            //Show a mensag. that the device has no bluetooth adapter
            Toast.makeText(getApplicationContext(), "Bluetooth Device Not Available", Toast.LENGTH_LONG).show();

            //finish apk
            finish();
        }
        else if(!myBluetooth.isEnabled())
        {
                //Ask to the user turn the bluetooth on
                Intent turnBTon = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(turnBTon,1);
        }

        btnPaired.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v)
            {
                pairedDevicesList();
            }
        });

    }

```



```
    private void pairedDevicesList()
    {
        pairedDevices = myBluetooth.getBondedDevices();
        ArrayList list = new ArrayList();

        if (pairedDevices.size()>0)
        {
            for(BluetoothDevice bt : pairedDevices)
            {
                list.add(bt.getName() + "\n" + bt.getAddress()); //Get the device's name and the address
            }
        }
        else
        {
            Toast.makeText(getApplicationContext(), "No Paired Bluetooth Devices Found.", Toast.LENGTH_LONG).show();
        }

        final ArrayAdapter adapter = new ArrayAdapter(this,android.R.layout.simple_list_item_1, list);
        devicelist.setAdapter(adapter);
        devicelist.setOnItemClickListener(myListClickListener); //Method called when the device from the list is clicked

    }

```



```java
    private AdapterView.OnItemClickListener myListClickListener = new AdapterView.OnItemClickListener()
    {
        public void onItemClick (AdapterView<?> av, View v, int arg2, long arg3)
        {
            // Get the device MAC address, the last 17 chars in the View
            String info = ((TextView) v).getText().toString();
            String address = info.substring(info.length() - 17);

            // Make an intent to start next activity.
            Intent i = new Intent(DeviceList.this, ledControl.class);

            //Change the activity.
            i.putExtra(EXTRA_ADDRESS, address); //this will be received at ledControl (class) Activity
            startActivity(i);
        }
    };


    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_device_list, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```