```java
package edu.seaaddicts.brockbutler.maps;

/**
 * Locate.java
 * Brock Butler
 * portion of Brock Butler.
 * Created by Thomas Nelson 2013-03-10
 * Copyright (c) 2013 Sea Addicts. All rights reserved.
 */

import java.util.List;
import android.content.Context;
import android.net.wifi.ScanResult;
import android.net.wifi.WifiManager;
import android.util.Log;

public class Locate {

  /**
   * Class variables
   */
  private static WifiManager      wifiMgr;
  private static List<ScanResult> scanResults;
  static Context parentContext;
  int[] answer = new int[10];
  /**
   * Wireless information containers
   */
  private static int[] sigStr  = new int[10];
  private static String[] address = new String[10];
  private static double[] addIn = new double[10];
  /**
   * Layers
   */
  private static final int inputs = 2;
  private static final int hidden = 8;
  private static final int output = 5;
  /**
   * Weights
   */
  private static double[][] W = new double[inputs][hidden];
  private static double[][] V = new double[hidden][output];
  private static double[] HB = new double[hidden];
  private static double[] OB = new double[output];
  /**
   * Neurons
   */
  private static double[] hiddenVal = new double[hidden];
  private static double[] outputVal = new double[output];
  private static double[][] inputVal = new double[10][inputs];

  /**
   * Constructor for the Locate class sets context and initializes weights
   * only once.
   * @param pc
   */
  public Locate (Context pc) {
    parentContext = pc;
    initWeights();
  }

  /**
   * Used by the mapping thread to get current user position, returns null if no position is found.
   * @return
   */
  public String getUserPosition ( ) {
    try {
      //getWirelessData(); // Use when you are testing on device
      initTestData(); // Use when testing on simulator testData1, testData2, or testData3
      //initData(); // Use when you are testing on device

      for(int i=0; i<10; i++) {
        calcNetwork(i);
        answer[i] = (int) (outputVal[0]*16 + outputVal[1]*8 + outputVal[2]*4 + outputVal[3]*2 + outputVal[4]*1);
      }

      int location = mode(answer);
      Log.i("LOCATE", "Node: " + location);
      switch(location) {
        case 1:
          return "J01";
        case 2:
          return "J02";
        case 3:
          return "J03";
        case 4:
          return "J04";
        case 5:
          return "J05";
        case 6:
          return "J06";
        case 7:
          return "J07";
        case 8:
          return "J08";
        case 9:
          return "J09";
        case 10:
          return "J10";
        case 11:
          return "J11";
        case 12:
          return "J12";
        case 13:
          return "J13";
        case 14:
          return "J14";
        case 15:
          return "J15";
        case 16:
          return "J16";
        case 17:
          return "J17";
        case 18:
          return "J18";
        case 19:
          return "J19";
        case 20:
          return "J20";
        case 21:
          return "J21";
        case 22:
          return "J22";
        case 23:
          return "J23";
        default:
          return "";
      }
    } catch (Exception err) {
      Log.e("LOCATE", err.getMessage());
    }
    return null;
  }

  /**
   * Gathers wireless information from the device for 10 wireless access points
   * currently in range. Gathers MAC address and received signal strength
   */
  @SuppressWarnings("unused")
  private static void getWirelessData() {
    wifiMgr = (WifiManager)parentContext.getSystemService(Context.WIFI_SERVICE);
      int x = 0;

      for(int num=0; num<10; num++) {
        wifiMgr.startScan();
        scanResults = wifiMgr.getScanResults();

        x = 0;
        sigStr  = new int[10];
        address = new String[10];

        for(ScanResult scanRes : scanResults) {
          if(x < 10) {
            address[x] = scanRes.BSSID;
            sigStr[x] = scanRes.level;
            x++;
          }
        }
      }
  }

  /**
   * This Method will return the sigmoid value of an argument
   * for the final node value.
   * @param x
   * @return
   */
  private static double sigmoid(double x) {
    return 1 / (1 + Math.exp(-x));
  }

  /**
   * The beans of this class, uses normalized wireless data to predict
   * user location based on an inputed input pattern.
   * @param pat
   */
  private static void calcNetwork(int pat) {
    for(int h=0; h<hidden; h++) {
      hiddenVal[h] = -HB[h];
      for(int i=0; i<inputs; i++) {
        hiddenVal[h] += (inputVal[pat][i] * W[i][h]);
      }
      hiddenVal[h] = sigmoid(hiddenVal[h]);
    }

    for(int o=0; o<output; o++) {
      outputVal[o] = -OB[o];
      for(int h=0; h<hidden; h++) {
        outputVal[o] += (hiddenVal[h] * V[h][o]);
      }
      outputVal[o] = sigmoid(outputVal[o]);

      if(outputVal[o] >= 0.5)
        outputVal[o] = 1;
        else if(outputVal[o] < 0.5)
          outputVal[o] = 0;
    }
  }

  /**
   * Searches through network output to find the most likely
   * user position.
   * @param a
   * @return
   */
  private static int mode(int a[]) {
    int maxValue=0, maxCount=0;

    for (int i = 0; i < a.length; ++i) {
      int count = 0;
      for (int j = 0; j < a.length; ++j) {
        if (a[j] == a[i]) ++count;
      }
      if (count > maxCount) {
        maxCount = count;
        maxValue = a[i];
      }
    }

    return maxValue;
  }

  private void initTestData() {
    inputVal[0][0] = 1; inputVal[0][1] = -65;
    inputVal[1][0] = 2; inputVal[1][1] = -60;
    inputVal[2][0] = 3; inputVal[2][1] = -64;
    inputVal[3][0] = 4; inputVal[3][1] = -64;
    inputVal[4][0] = 5; inputVal[4][1] = -68;
    inputVal[5][0] = 6; inputVal[5][1] = -69;
    inputVal[6][0] = 7; inputVal[6][1] = -69;
    inputVal[7][0] = 8; inputVal[7][1] = -72;
    inputVal[8][0] = 9; inputVal[8][1] = -74;
    inputVal[9][0] = 10; inputVal[9][1] = -74;
  }

  @SuppressWarnings("unused")
  private void initTestData2() {
    inputVal[0][0] = 1; inputVal[0][1] = -65;
    inputVal[1][0] = 1; inputVal[1][1] = -89;
    inputVal[2][0] = 6; inputVal[2][1] = -64;
    inputVal[3][0] = 8; inputVal[3][1] = -64;
    inputVal[4][0] = 11; inputVal[4][1] = -66;
    inputVal[5][0] = 12; inputVal[5][1] = -66;
    inputVal[6][0] = 13; inputVal[6][1] = -71;
    inputVal[7][0] = 14; inputVal[7][1] = -72;
    inputVal[8][0] = 15; inputVal[8][1] = -72;
    inputVal[9][0] = 16; inputVal[9][1] = -72;
  }

  @SuppressWarnings("unused")
  private void initTestData3() {
    inputVal[0][0] = 6; inputVal[0][1] = -58;
    inputVal[1][0] = 8; inputVal[1][1] = -58;
    inputVal[2][0] = 11; inputVal[2][1] = -72;
    inputVal[3][0] = 12; inputVal[3][1] = -72;
    inputVal[4][0] = 13; inputVal[4][1] = -75;
    inputVal[5][0] = 14; inputVal[5][1] = -66;
    inputVal[6][0] = 15; inputVal[6][1] = -77;
    inputVal[7][0] = 16; inputVal[7][1] = -75;
    inputVal[8][0] = 9; inputVal[8][1] = -69;
    inputVal[9][0] = 16; inputVal[9][1] = -72;
  }

  /**
   * Makes the wireless data usable and sets it up for
   * the network to use by putting the values between 0 and 1
   * with min/max normalization.
   */
  @SuppressWarnings("unused")
  private void initData() {
    for (int x=0; x<10; x++) {
      if (address[x].equalsIgnoreCase("00:0b:86:91:ce:a1"))
        addIn[x] = 1;
      else if (address[x].equalsIgnoreCase("00:0b:86:8a:8c:02"))
        addIn[x] = 2;
      else if (address[x].equalsIgnoreCase("00:1a:1e:fc:af:21"))
        addIn[x] = 3;
      else if (address[x].equalsIgnoreCase("00:0b:86:91:ce:a2"))
        addIn[x] = 4;
      else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:e2"))
        addIn[x] = 5;
      else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:e1"))
        addIn[x] = 6;
      else if (address[x].equalsIgnoreCase("00:0b:86:89:f6:e1"))
        addIn[x] = 7;
      else if (address[x].equalsIgnoreCase("00:1a:1e:fc:af:22"))
        addIn[x] = 8;
      else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b2:62"))
        addIn[x] = 9;
      else if (address[x].equalsIgnoreCase("00:0b:86:4d:8f:21"))
        addIn[x] = 10;
      else if (address[x].equalsIgnoreCase("00:0b:86:4d:8f:22"))
        addIn[x] = 11;
      else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:21"))
        addIn[x] = 12;
      else if (address[x].equalsIgnoreCase("00:1a:1e:a7:dc:22"))
        addIn[x] = 13;
      else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:22"))
        addIn[x] = 14;
      else if (address[x].equalsIgnoreCase("00:1a:1e:a7:dc:21"))
        addIn[x] = 15;
      else if (address[x].equalsIgnoreCase("00:0b:86:8a:8c:01"))
        addIn[x] = 16;
      else if (address[x].equalsIgnoreCase("00:1a:1e:a7:e4:c2"))
        addIn[x] = 17;
      else if (address[x].equalsIgnoreCase("00:1a:1e:a7:e4:c1"))
        addIn[x] = 18;
      else if (address[x].equalsIgnoreCase("00:1a:1e:fc:ac:82"))
        addIn[x] = 19;
      else if (address[x].equalsIgnoreCase("00:1a:1e:fc:ac:81"))
        addIn[x] = 20;
      else if (address[x].equalsIgnoreCase("00:0b:86:91:ce:a0"))
        addIn[x] = 21;
      else if (address[x].equalsIgnoreCase("00:0b:86:8a:8c:00"))
        addIn[x] = 22;
      else if (address[x].equalsIgnoreCase("00:1a:1e:fc:ac:80"))
        addIn[x] = 23;
      else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b2:61"))
        addIn[x] = 24;
      else if (address[x].equalsIgnoreCase("00:0b:86:42:de:80"))
        addIn[x] = 25;
      else if (address[x].equalsIgnoreCase("00:0b:86:42:de:82"))
        addIn[x] = 26;
      else if (address[x].equalsIgnoreCase("00:1a:1e:a7:e4:c0"))
        addIn[x] = 27;
      else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b2:60"))
        addIn[x] = 28;
      else
        addIn[x] = 0;
    }

    for (int x=0; x<10; x++) {
      inputVal[x][0] = (addIn[x] - 1) / 28;
      inputVal[x][1] = (sigStr[x] - -97) / 58;
    }
  }

  /**
   * Initializes the network with pre-defined weights, currently will only find a
   * position in JBlock.
   */
  private void initWeights ( ) {
    W[0][0] = -5.191953555370145;
    W[1][0] = 8.311119623052747;
    HB[0] = 13.645070679100112;
    V[0][0] = -1.4188817448241078;
    OB[0] = 8.485931304116875;
    V[0][1] = -0.12644482595532947;
    OB[1] = 7.676266130312892;
    V[0][2] = -0.8742897792429708;
    OB[2] = 7.71163044603109;
    V[0][3] = 8.174930730213324;
```

```
        OB[3] = 5.050964297399435;
        V[0][4] = -1.483533992384484;
        OB[4] = -0.9537318879960314;
        W[0][1] = 1.2747639658533194;
        W[1][1] = 35.85241447388153;
        HB[1] = 22.187068222811735;
        V[1][0] = 0.08863280595533382;
        OB[0] = 8.485931304116875;
        V[1][1] = -0.711381178420192;
        OB[1] = 7.676266130312892;
        V[1][2] = 0.47717648845370575;
        OB[2] = 7.71163044603109;
        V[1][3] = -0.9106351920878477;
        OB[3] = 5.050964297399435;
        V[1][4] = -21.858140121995646;
        OB[4] = -0.9537318879960314;
        W[0][2] = 72.6159523077509;
        W[1][2] = 62.83113577555713;
        HB[2] = 48.81891128647234;
        V[2][0] = -0.2564556186462736;
        OB[0] = 8.485931304116875;
        V[2][1] = -1.3189848572962326;
        OB[1] = 7.676266130312892;
        V[2][2] = -1.132371452477312;
        OB[2] = 7.71163044603109;
        V[2][3] = 6.23102460219137;
        OB[3] = 5.050964297399435;
        V[2][4] = 0.9547440817617039;
        OB[4] = -0.9537318879960314;
        W[0][3] = -12.93760218402065;
        W[1][3] = -27.44465213223537;
        HB[3] = -12.785991014176767;
        V[3][0] = -0.3900489003340711;
        OB[0] = 8.485931304116875;
        V[3][1] = -0.4526342230399839;
        OB[1] = 7.676266130312892;
        V[3][2] = -1.1900195982331039;
        OB[2] = 7.71163044603109;
        V[3][3] = 32.266073798358136;
        OB[3] = 5.050964297399435;
        V[3][4] = 0.8870702490085969;
        OB[4] = -0.9537318879960314;
        W[0][4] = 20.564192571900584;
        W[1][4] = 64.556122443447;
        HB[4] = 51.94209137066473;
        V[4][0] = -0.14542636219949362;
        OB[0] = 8.485931304116875;
        V[4][1] = -0.18725148477033168;
        OB[1] = 7.676266130312892;
        V[4][2] = -1.3082405929431982;
        OB[2] = 7.71163044603109;
        V[4][3] = 7.750490464388548;
        OB[3] = 5.050964297399435;
        V[4][4] = 7.381930098882;
        OB[4] = -0.9537318879960314;
        W[0][5] = 26.203506852804754;
        W[1][5] = 30.430410744252843;
        HB[5] = 30.471677989844203;
        V[5][0] = 0.6785829497339613;
        OB[0] = 8.485931304116875;
        V[5][1] = -0.7290230440401178;
        OB[1] = 7.676266130312892;
        V[5][2] = 0.01863788955350768;
        OB[2] = 7.71163044603109;
        V[5][3] = 13.290562812876107;
        OB[3] = 5.050964297399435;
        V[5][4] = -3.191226914646271;
        OB[4] = -0.9537318879960314;
        W[0][6] = -4.992559685945157;
        W[1][6] = 88.04743967482369;
        HB[6] = 50.39739242687603;
```

```
    V[6][0] = 0.0746839836828444;
    OB[0] = 8.485931304116875;
    V[6][1] = -0.5879731640198912;
    OB[1] = 7.676266130312892;
    V[6][2] = -0.25250658203018556;
    OB[2] = 7.71163044603109;
    V[6][3] = 1.4602581997464505;
    OB[3] = 5.050964297399435;
    V[6][4] = 18.463904409636967;
    OB[4] = -0.9537318879960314;
    W[0][7] = -4.74928985780168;
    W[1][7] = 40.78597102067532;
    HB[7] = 35.79898233956728;
    V[7][0] = -0.09439880889024627;
    OB[0] = 8.485931304116875;
    V[7][1] = -1.1230801185236317;
    OB[1] = 7.676266130312892;
    V[7][2] = -0.2674705623236563;
    OB[2] = 7.71163044603109;
    V[7][3] = 8.441254951069187;
    OB[3] = 5.050964297399435;
    V[7][4] = -20.04591503798929;
    OB[4] = -0.9537318879960314;
  }
}


```