References

[1] Y.-Y. Tang, B. K. Hölzel, and M. I. Posner, "The neuroscience of mindfulness meditation," *Nat. Rev. Neurosci.*, vol. 16, no. 4, pp. 213–225, Mar. 2015.

[2] B. K. Hölzel *et al.*, "Mindfulness practice leads to increases in regional brain gray matter density," *Psychiatry Res. Neuroimaging*, vol. 191, no. 1, pp. 36–43, Jan. 2011.

[3] H. A. Slagter, R. J. Davidson, and A. Lutz, "Mental Training as a Tool in the Neuroscientific Study of Brain and Cognitive Plasticity," *Front. Hum. Neurosci.*, vol. 5, 2011.

[4] "Meditation and the Brain," *MIT Technology Review*.

[5] R. J. Davidson and A. Lutz, "Buddha's brain: Neuroplasticity and meditation," *IEEE Signal Process. Mag.*, vol. 25, no. 1, p. 176, 2008.

[6] R. J. Davidson and B. S. McEwen, "Social influences on neuroplasticity: stress and interventions to promote well-being," *Nat. Neurosci.*, vol. 15, no. 5, pp. 689–695, Apr. 2012.

[7] K. A. Norman, S. M. Polyn, G. J. Detre, and J. V. Haxby, "Beyond mind-reading: multi-voxel pattern analysis of fMRI data," *Trends Cogn. Sci.*, vol. 10, no. 9, pp. 424–430, Sep. 2006.

[8] Y. Kamitani and F. Tong, "Decoding the visual and subjective contents of the human brain," *Nat. Neurosci.*, vol. 8, no. 5, pp. 679–685, 2005.

[9] K. A. Garrison *et al.*, "Real-time fMRI links subjective experience with brain activity during focused attention," *NeuroImage*, vol. 81, pp. 110–118, Nov. 2013.

[10] A. Lutz, H. A. Slagter, N. B. Rawlings, A. D. Francis, L. L. Greischar, and R. J. Davidson, "Mental Training Enhances Attentional Stability: Neural and Behavioral Evidence," *J. Neurosci.*, vol. 29, no. 42, pp. 13418–13427, Oct. 2009.

[11] A. Lutz, L. L. Greischar, N. B. Rawlings, M. Ricard, and R. J. Davidson, "Long-term meditators self-induce high-amplitude gamma synchrony during mental practice," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 101, no. 46, pp. 16369–16373, 2004.

[12] J. A. Brefczynski-Lewis, A. Lutz, H. S. Schaefer, D. B. Levinson, and R. J. Davidson, "Neural correlates of attentional expertise in long-term meditation practitioners," *Proc. Natl. Acad. Sci.*, vol. 104, no. 27, pp. 11483–11488, 2007.

[13] E. Baron Short *et al.*, "Regional Brain Activation during Meditation Shows Time and Practice Effects: An Exploratory FMRI Study," *Evid. Based Complement. Alternat. Med.*, vol. 7, no. 1, pp. 121–127, 2010.

[14] D. G. MacCoon *et al.*, "The validation of an active control intervention for Mindfulness Based Stress Reduction (MBSR)," *Behav. Res. Ther.*, vol. 50, no. 1, pp. 3–12, Jan. 2012.

[15] P. Malinowski, "Neural mechanisms of attentional control in mindfulness meditation," *Front. Neurosci.*, vol. 7, 2013.

[16] B. R. Cahn, A. Delorme, and J. Polich, "Event-related delta, theta, alpha and gamma correlates to auditory oddball processing during Vipassana meditation," *Soc. Cogn. Affect. Neurosci.*, vol. 8, no. 1, pp. 100–111, Jan. 2013.

[17] B. R. Cahn, A. Delorme, and J. Polich, "Occipital gamma activation during Vipassana meditation," *Cogn. Process.*, vol. 11, no. 1, pp. 39–56, Feb. 2010.

[18] V. Pizzella *et al.*, “Brain Activity of Buddhist Monks During Focused Attention Meditation: A MEG Study,” *NeuroImage*, vol. 47, p. S156, Jul. 2009.
[19] S. W. Lazar, G. Bush, R. L. Gollub, G. L. Fricchione, G. Khalsa, and H. Benson, “Functional brain mapping of the relaxation response and meditation,” *NeuroReport*, vol. 11, no. 7, pp. 1581–1585, May 2000.
[20] UNIVERSITY of WISCONSIN–MADISON, “The Dalai Lama and scientists unite to study meditation.”
[21] B. K. Holzel *et al.*, “Investigation of mindfulness meditation practitioners with voxel-based morphometry,” *Soc. Cogn. Affect. Neurosci.*, vol. 3, no. 1, pp. 55–61, Jun. 2007.
[22] F. Ferrarelli *et al.*, “Experienced Mindfulness Meditators Exhibit Higher Parietal-Occipital EEG Gamma Activity during NREM Sleep,” *PLoS ONE*, vol. 8, no. 8, p. e73417, Aug. 2013.
[23] A. Ahani, H. Wahbeh, H. Nezamfar, M. Miller, D. Erdogmus, and B. Oken, “Quantitative change of EEG and respiration signals during mindfulness meditation,” *J Neuroeng Rehabil*, vol. 11, no. 87, pp. 10–1186, 2014.
[24] R. Singla and N. Sharma, “Function Classification of EEG Signals Based on ANN,” *Int. J. Soft Comput. Eng.*, vol. 2, pp. 158–163, 2014.
[25] J. V. Haxby, “Multivariate pattern analysis of fMRI: The early beginnings,” *NeuroImage*, vol. 62, no. 2, pp. 852–855, Aug. 2012.
[26] Russell, Stuart J., Peter Norvig, and John Canny., *“Artificial Intelligence: A Modern Approach”. 2003.* .
[27] G. Rzevski, V. Soloviev, P. Skobelev, and O. Lakhin, “Complex adaptive logistics for the international space station,” *Int. J. Des. Nat. Ecodynamics*, vol. 11, no. 3, pp. 459–472, Jul. 2016.
[28] B. Hettige, A. S. Karunananda, and G. Rzevski, “A multi-agent solution for managing complexity in english to sinhala machine translation,” *Int. J. Des. Nat. Ecodynamics*, vol. 11, no. 2, pp. 88–96, Apr. 2016.
[29] J. Lagopoulos *et al.*, “Increased theta and alpha EEG activity during nondirective meditation,” *J. Altern. Complement. Med.*, vol. 15, no. 11, pp. 1187–1192, 2009.
[30] Courtney Humphries, “Brain Mapping : MIT Technology Review.”
[31] Nature, “Brain Mapping.”
[32] F. Pereira, T. Mitchell, and M. Botvinick, “Machine learning classifiers and fMRI: a tutorial overview,” *Neuroimage*, vol. 45, no. 1, pp. S199–S209, 2009.

Appendix A

Multi-Agent System

A.1 Introduction

This section provides screenshots of the multi agent system of the meditation solution and the jade configuration.

A.2 Multi agent system – Meditation solution

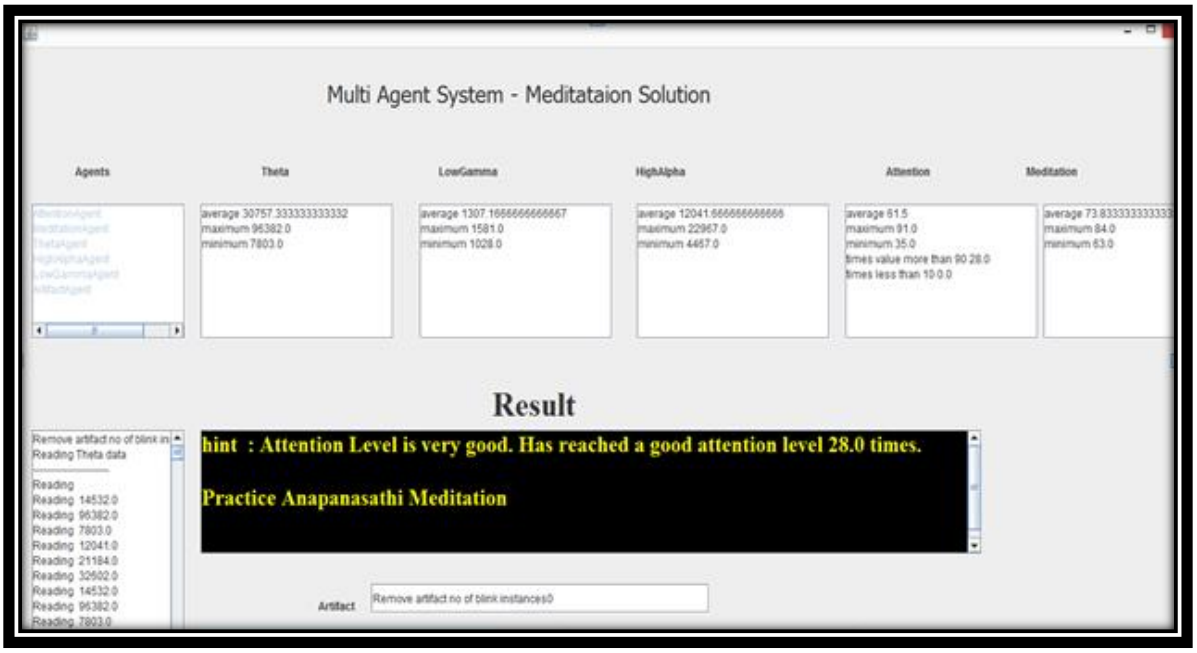
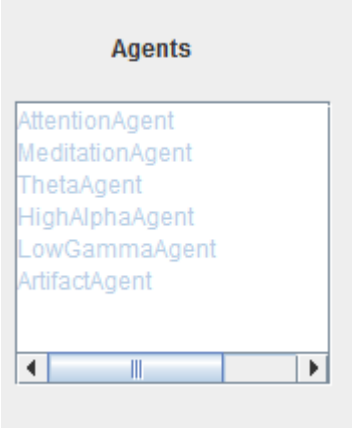
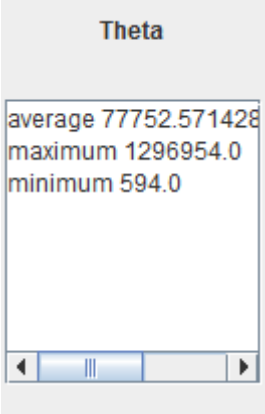


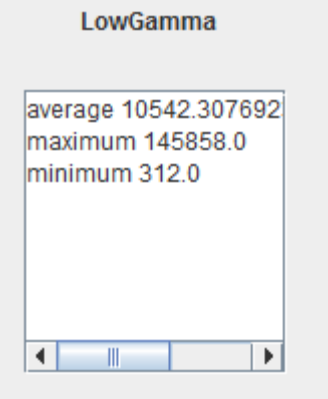
Figure 10.1: Multi agent system results

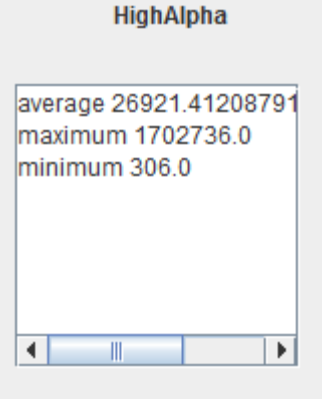
The activated agents are listed

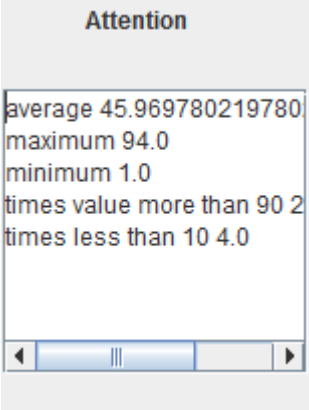


Various agents have data which they get from the environment. (From sensors)



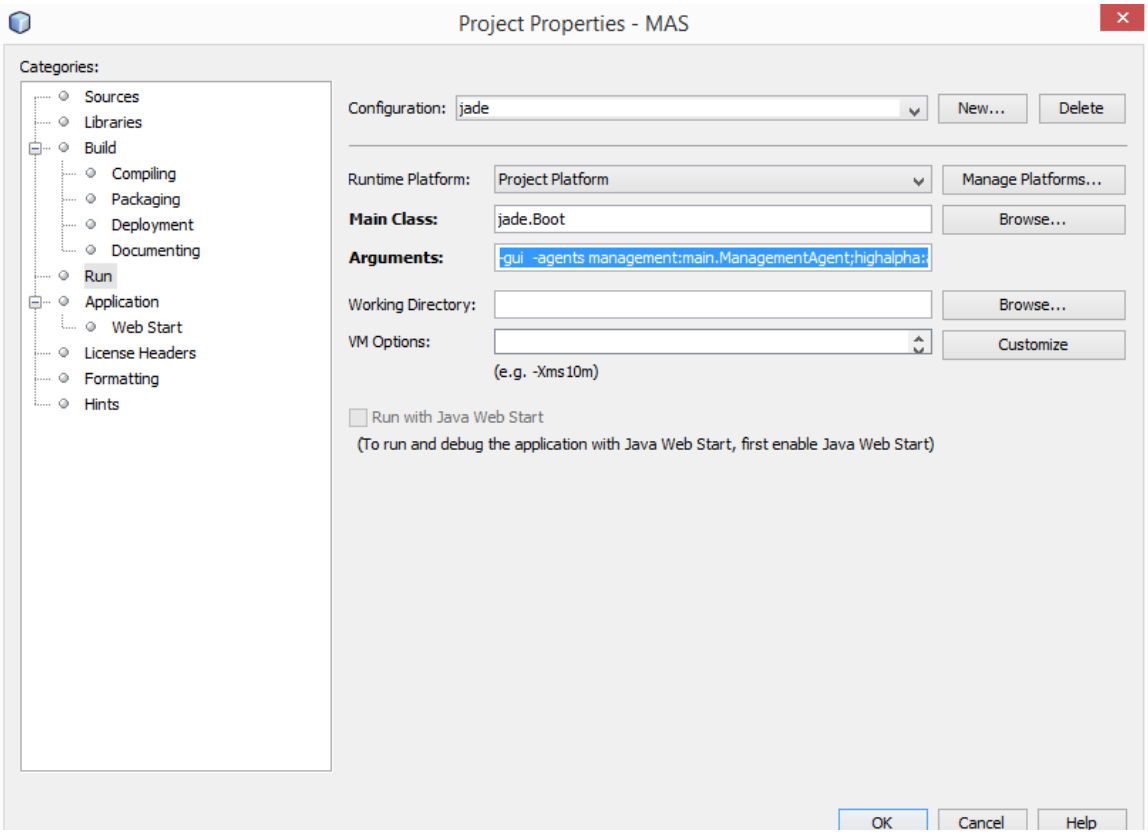






Setting up JADE

JADE (Java Agent Development Framework) is a software Framework fully implemented in the Java language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications. JADE is free software and is distributed by Telecom Italia, the copyright holder, in open source under the terms and conditions of the LGPL (Lesser General Public License Version 2) license.



-gui -agents management:main.ManagementAgent;highalpha:agents.HighAlphaAgent;theta:agents.ThetaAgent;decision:main.DecisionAgent;attention:agents.Attentionagent;meditation:agents.Meditationagent;art:agents.ArtifactAgent;lowgamma:agents.LowGammaAgent

Appendix B

Source code of the system

B.1 Introduction

This section will include source code which was used to develop the proposed system.

B.2 Source code of the proposed system

Code snippet from AttentionBehavior

```
private void Report() {
      AID r = new AID("management@" + myAgent.getHap(), AID.ISGUID);

      ACLMessage aclMessage = new ACLMessage(ACLMessage.INFORM);
      aclMessage.addReceiver(r);
      try {
         List<String> esense = TextFileReader.readData("attention");
         String eeg = TextFileReader.readAverage(esense);
         aclMessage.setContent("aaaaa" + eeg + "-" + TextFileReader.readMax(esense) +
"-"+TextFileReader.readMin(esense)+"-" + TextFileReader.getValueList(esense)+
               "-"+                              TextFileReader.moreThan90(esense)+"-"+
TextFileReader.lessThan10(esense));

      } catch (Exception e) {
         e.printStackTrace();
      }

      myAgent.send(aclMessage);
   }
```

Code snippet calculating average

```
public static HashMap<String, Integer> EegPowerAverage(List<JSONObject> obj)
throws JSONException {
```

```
        int deltasum = 0, thetaSum = 0, lowAlphaSum = 0, highAlphaSum = 0, lowBetaSum
= 0;
        int highGammaSum = 0, highBetaSum = 0, lowGammaSum = 0;
        int attentionSum = 0, meditationSum = 0;

        for (JSONObject obj1 : obj) {
            deltasum += (int) obj1.get("delta");
            thetaSum += (int) obj1.get("theta");
            lowAlphaSum += (int) obj1.get("lowAlpha");
            highAlphaSum += (int) obj1.get("highAlpha");
            lowBetaSum += (int) obj1.get("lowBeta");
            highGammaSum += (int) obj1.get("highGamma");
            highBetaSum += (int) obj1.get("highBeta");
            lowGammaSum += (int) obj1.get("lowGamma");
        }

        HashMap<String, Integer> hmap = new HashMap<>();
        hmap.put("deltaAvg", (deltasum / obj.size()));
        hmap.put("thetaAvg", (thetaSum / obj.size()));
        hmap.put("lowAlphaAvg", (lowAlphaSum / obj.size()));
        hmap.put("highAlphaAvg", (highAlphaSum / obj.size()));
        hmap.put("lowBetaAvg", (lowBetaSum / obj.size()));
        hmap.put("highGammaAvg", (highGammaSum / obj.size()));
        hmap.put("HighBetaAvg", (highBetaSum / obj.size()));
        hmap.put("LowGammaAvg", (lowGammaSum / obj.size()));

        return hmap;
    }
```

Retrieving minimum, maximum and average

```
public static String readAverage(List<String> obj) {

    int size = obj.size();
    double count = 0;
    for (int i = 0; i < size; i++) {
        count = count + Double.parseDouble(obj.get(i));
    }

    String avg = Double.toString(count / size);
    return avg;

}

public static String readMax(List<String> obj) {

    double max = 0;

    for (String obj1 : obj) {
        double val = Double.parseDouble(obj1);
        if (val > max) {
            max = val;
        }
    }
```

```
    return Double.toString(max);
}

public static String readMin(List<String> obj) {

    double min = Double.parseDouble(obj.get(0));
    for (String i : obj) {
        double temp = Double.parseDouble(i);
        min = min < temp ? min : temp;
    }

    return Double.toString(min);
}
```

Code from ANN Approach

loadTrianedANN

```
public void loadTrianedANN()
   {
      setTrainedNetwork(new BasicNetwork());
      eegDataTxtArea.append("Loading ANN\n");
      try
      {
         FileInputStream fin = new FileInputStream("meditationsol.dat");


         ObjectInputStream ois = new ObjectInputStream(fin);
         setTrainedNetwork((BasicNetwork) ois.readObject());
         ois.close();


         eegDataTxtArea.append("Loading ANN Completed\n");
      } catch (IOException | ClassNotFoundException e)
      {
         JOptionPane.showMessageDialog(null, e.getMessage());
      }
   }
```

startEEGMonitoring

```
            if (currentEEGReading.contains("eSense"))
            {

eegDataTxtArea.append(currentEEGReading.substring(currentEEGReading.indexOf("ee
gPower")) + "\n");
               eegDataTxtArea.setCaretPosition(eegDataTxtArea.getText().length());
            }
            else
            {
               eegDataTxtArea.append(currentEEGReading + "\n");
               eegDataTxtArea.setCaretPosition(eegDataTxtArea.getText().length());
            }



            double attention = 0;
            double meditation = 0;
            if (currentEEGReading.contains("eSense"))
            {
               double delta = 0, theta = 0, lowAlpha = 0, highAlpha = 0, lowBeta = 0,
highBeta = 0, lowGamma = 0, highGamma = 0, poorSignalLevel = 0;
               currentEEGReading = currentEEGReading.replace("\"", "");
               currentEEGReading = currentEEGReading.replace("{", "");
               currentEEGReading = currentEEGReading.replace("}", "");
               currentEEGReading = currentEEGReading.replace("eSense:", "");
               currentEEGReading = currentEEGReading.replace("eegPower:", "");
```

```
String[] eegValues = currentEEGReading.split(",");
for (String eegVal : eegValues)
{
   String[] eegReadings = eegVal.split(":");
   double tempReading = Double.parseDouble(eegReadings[1]);
   switch (eegReadings[0])
   {
      case "attention":
         attention = tempReading;
         break;
      case "meditation":
         meditation = tempReading;
         break;
      case "delta":
         delta = tempReading;
         break;
      case "theta":
         theta = tempReading;
         break;
      case "lowAlpha":
         lowAlpha = tempReading;
         break;
      case "highAlpha":
         highAlpha = tempReading;
```

```
            break;
        case "lowBeta":
            lowBeta = tempReading;
            break;
        case "highBeta":
            highBeta = tempReading;
            break;
        case "lowGamma":
            lowGamma = tempReading;
            break;
        case "highGamma":
            highGamma = tempReading;
            break;
        case "poorSignalLevel":
            poorSignalLevel = tempReading;
            break;
        default:
            break;
    }
}

if (poorSignalLevel < ACCEPTED_SIGNAL_LEVEL)
{
    INPUT[0] = delta;
    INPUT[1] = theta;
```

```
        INPUT[2] = lowAlpha;
        INPUT[3] = highAlpha;
        INPUT[4] = lowBeta;
        INPUT[5] = highBeta;
        INPUT[6] = lowGamma;
        INPUT[7] = highGamma;

        fullGamma += highGamma;
        iterations = iterations + 1;
    }

}
```

MeditationNEuroskySocketClient

```
/**
 *
 * @author Achala
 */
public class MeditationNEuroskySocketClient {

    /**
     * Logger for this class
     */
    private static final Logger logger =
Logger.getLogger(MeditationNEuroskySocketClient.class);
    public static final String DEFAULT_HOST = "127.0.0.1";
    public static final int DEFAULT_PORT = 13854;

    private static MeditationNEuroskySocketClient INSTANCE = null;
    private String host;
    private int port;
    private boolean connected;
    SocketChannel channel;
    Scanner in;

    private MeditationNEuroskySocketClient() {

        this.host = DEFAULT_HOST;
```

```
    this.port = DEFAULT_PORT;
    this.connected = false;

}

public MeditationNEuroskySocketClient(String host, int port) {

    this.host = host;
    this.port = port;
    this.connected = false;

}

public static MeditationNEuroskySocketClient getInstance() {
    if (INSTANCE == null) {
        INSTANCE = new MeditationNEuroskySocketClient();
    }

    return INSTANCE;
}

public String getHost() {
    return host;
}
```

```
    public void setHost(String host) {
        this.host = host;
    }

    public int getPort() {
        return port;
    }

    public void setPort(int port) {
        this.port = port;
    }

    public boolean isConnected() {
        return this.connected;
    }

    public void connect(boolean enableRawData) throws IOException {

        if (!this.connected) {

            this.channel = SocketChannel.open(new InetSocketAddress(this.host, this.port));

            CharsetEncoder enc = Charset.forName("US-ASCII").newEncoder();

            String jsonCommand = "{\"timestamp\":true,\"enableRawOutput\": false,
\"format\": \"Json\"}\n";
```

```
            this.channel.write(enc.encode(CharBuffer.wrap(jsonCommand)));


            this.in = new Scanner(channel);
            this.connected = true;


           System.out.println("Meditation solution is ocnnected with the device");
        } else {


            System.out.println("Already connected");
        }


    }


    public void startRecording() throws IOException {
        if (this.connected) {
            CharsetEncoder enc = Charset.forName("US-ASCII").newEncoder();


            String jsonCommand =
"{\"startRecording\":{\"rawEeg\":true,\"poorSignalLevel\":true,\"eSense\":true,\"eegPow
er\":true,\"blinkStrength\":true},\"applicationName\":\"ExampleApp\"}\n";
            this.channel.write(enc.encode(CharBuffer.wrap(jsonCommand)));


            JOptionPane.showMessageDialog(null, "Start Recording");


        } else {
            logger.debug("startRecording meditation session () - Not connected...");
```

```
    }

}

public void stopRecording() throws IOException {
    if (this.connected) {
        CharsetEncoder enc = Charset.forName("US-ASCII").newEncoder();

        String jsonCommand = "{\"stopRecording\":\"ExampleApp\"}\n";
        this.channel.write(enc.encode(CharBuffer.wrap(jsonCommand)));

        JOptionPane.showMessageDialog(null, "Stop Recording");

    } else {
        logger.debug("stopRecording meditation session() - Not connected...");
    }
}



public boolean isDataAvailable() {
    if (this.connected) {
        return this.in.hasNextLine();
    } else {
        return false;
```

```
        }
    }

    public String getData() {
        return this.in.nextLine();
    }

    public void close() throws IOException {

        if (this.connected) {

            this.in.close();
            this.channel.close();
            this.connected = false;
        }
    }

}
```

```
    /**
     * Logger for this class
     */
    private static final Logger logger =
Logger.getLogger(MeditationNEuroskySocketClient.class);
    public static final String DEFAULT_HOST = "127.0.0.1";
    public static final int DEFAULT_PORT = 13854;

    private static MeditationNEuroskySocketClient INSTANCE = null;
    private String host;
    private int port;
    private boolean connected;
    SocketChannel channel;
    Scanner in;

    private MeditationNEuroskySocketClient() {

        this.host = DEFAULT_HOST;
        this.port = DEFAULT_PORT;
        this.connected = false;

    }

    public MeditationNEuroskySocketClient(String host, int port) {

        this.host = host;
```

```
    this.port = port;
    this.connected = false;

}

public static MeditationNEuroskySocketClient getInstance() {
    if (INSTANCE == null) {
        INSTANCE = new MeditationNEuroskySocketClient();
    }

    return INSTANCE;
}

public String getHost() {
    return host;
}

public void setHost(String host) {
    this.host = host;
}

public int getPort() {
    return port;
}
```

```
    public void setPort(int port) {
        this.port = port;
    }

    public boolean isConnected() {
        return this.connected;
    }

    public void connect(boolean enableRawData) throws IOException {

        if (!this.connected) {

            this.channel = SocketChannel.open(new InetSocketAddress(this.host, this.port));

            CharsetEncoder enc = Charset.forName("US-ASCII").newEncoder();

            String jsonCommand = "{\"timestamp\":true,\"enableRawOutput\": false,
\"format\": \"Json\"}\n";
            this.channel.write(enc.encode(CharBuffer.wrap(jsonCommand)));

            this.in = new Scanner(channel);
            this.connected = true;

           System.out.println("Meditation solution is ocnnected with the device");
        } else {
```

```
            System.out.println("Already connected");
        }

    }

    public void startRecording() throws IOException {
        if (this.connected) {
            CharsetEncoder enc = Charset.forName("US-ASCII").newEncoder();

            String jsonCommand =
"{\"startRecording\":{\"rawEeg\":true,\"poorSignalLevel\":true,\"eSense\":true,\"eegPow
er\":true,\"blinkStrength\":true},\"applicationName\":\"ExampleApp\"}\n";
            this.channel.write(enc.encode(CharBuffer.wrap(jsonCommand)));

            JOptionPane.showMessageDialog(null, "Start Recording");

        } else {
            logger.debug("startRecording meditation session () - Not connected...");
        }

    }

    public void stopRecording() throws IOException {
        if (this.connected) {
            CharsetEncoder enc = Charset.forName("US-ASCII").newEncoder();
```

```
        String jsonCommand = "{\"stopRecording\":\"ExampleApp\"}\n";
        this.channel.write(enc.encode(CharBuffer.wrap(jsonCommand)));

        JOptionPane.showMessageDialog(null, "Stop Recording");

    } else {
        logger.debug("stopRecording meditation session() - Not connected...");
    }
}



public boolean isDataAvailable() {
    if (this.connected) {
        return this.in.hasNextLine();
    } else {
        return false;
    }
}

public String getData() {
    return this.in.nextLine();
}

public void close() throws IOException {
```

```
        if (this.connected) {

            this.in.close();
            this.channel.close();
            this.connected = false;
        }
    }

}
```