Multi PC Controller with Java

Various things can be accomplished by connecting a PC to Arduino. However, if you have more than one machine, you may want to manage all of them from a single point. In this example, a PC is used to control two other PCs (connected via a hub). One thing that is worth noting is that each client can have a different function. One of the client is connected to Arduino. We will activate the Arduino's built-in LED via the main computer. A Java library, *Ardulink*, is used for this purpose. The second PC client will only receive some text messages.

Here are the code files for the PCs:

mainmachine.java

```
import java.io.*;
import java.net.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class MainMachine {
    public static void main(String[] args) throws IOException, InterruptedException {
        Machine1Thread thr1 = new Machine1Thread('169.254.139.169', 38);
        Machine2Thread thr2 = new Machine2Thread('169.254.141.92', 994);

        thr1.start();
        thr2.start();
    }
}
```

```
class Machine1Thread extends Thread {
    private Socket socket = null;
    private PrintWriter out = null;

    public Machine1Thread(String ip, int port) {
        try {
            socket = new Socket(ip, port);
            out = new PrintWriter(socket.getOutputStream(), true);
        } catch (UnknownHostException e) {
            System.err.println(e);
            System.exit(1);
        } catch (IOException e) {
            System.err.println(e);
            System.exit(1);
        }
    }

    public void run(){
        try {
            out.println('ON');
            Thread.sleep(10000);
            out.println('OFF');

            out.close();
            socket.close();
        } catch (InterruptedException ex) {

        } catch (IOException ex) {

        }
    }
}


class Machine2Thread extends Thread {
    private Socket socket = null;
    private PrintWriter out = null;

    public Machine2Thread(String ip, int port) {
        try {
            socket = new Socket(ip, port);
            out = new PrintWriter(socket.getOutputStream(), true);
        } catch (UnknownHostException e) {
            System.err.println(e);
            System.exit(1);
        } catch (IOException e) {
            System.err.println(e);
            System.exit(1);
        }
    }

    public void run(){
        try {
            out.println('Test2a');
            out.println('Test2b');
            out.println('Bye');

            out.close();
            socket.close();
        } catch (IOException ex) {

        }
    }
}
```

Explanation

The file *mainmachine.java* is the controller. It runs on the main PC and builds a point-to-point communication channel with each of the clients. Through the use of threads, it can handle multiple clients

simultaneously. The interaction is performed via a *Socket* object that takes two parameters, the IP address of the client and an unused port number.

To find out the unused port number of the clients, the following program can be used:

portscanner.java

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.net.*;
import java.util.ArrayList;
import java.util.concurrent.TimeUnit;
import javax.swing.*;

public class PortScanner extends JFrame{
    static JTextArea ta=null;
    JScrollPane sp=null;

    public PortScanner(){
        super('Port Scanner');
        setSize(400,400);
        setLayout(new BorderLayout());

        JButton b=new JButton('Start');
        b.addActionListener(new ButtonHandler());
        add('North',b);

        ta=new JTextArea(20,20);
        sp=new JScrollPane(ta);
        add('Center',sp);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    public static void main(String[] args){
        new PortScanner();
    }
}

class ButtonHandler implements ActionListener{
    public void actionPerformed(ActionEvent e){
        int startPort=0;
        int stopPort=1024; //49151 - 65536

        CheckPort[] cp=new CheckPort[(stopPort+1)-startPort];

        for(int i=startPort;i<=stopPort;i++){
            cp[i]=new CheckPort(i);
            cp[i].start();
        }
    }
}

class CheckPort extends Thread{
    int curPort=0;

    public CheckPort(int x){
        curPort=x;
    }

    public void run(){
        try{
            Socket sok=new Socket('127.0.0.1',curPort);
            PortScanner.ta.append('Port: '+curPort+' (in use) ');
            sleep(1000);
            sok.close();
        }catch(Exception e){
            PortScanner.ta.append('Port: '+curPort+' (not in use) ');
```

```
        }
    }
}
```

submachine1.java

```
import java.net.*;
import java.io.*;
import org.zu.ardulink.Link;
import org.zu.ardulink.protocol.IProtocol;

public class SubMachine {
    public static void main(String[] args) throws IOException, InterruptedException {
        ServerSocket serverSocket = null;
        Socket socket = null;

        String connectionPort = 'COM4';
        int baudRate = 115200;
        Link link = Link.getDefaultInstance();
        boolean connected = link.connect(connectionPort, baudRate);

        if(connected){
            try {
                serverSocket = new ServerSocket(38);
                socket = serverSocket.accept();
            }catch (IOException e) {
                System.err.println('IOException:' + e);
                System.exit(1);
            }catch(Exception e) {
                System.err.println('Exception:' + e);
                System.exit(1);
            }

            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            String strLine = null;

            while (true){
                strLine = in.readLine();
                if(strLine != null){
                    System.out.println(strLine);
                    if(strLine.equals('ON')){
                        link.sendPowerPinSwitch(13,IProtocol.POWER_HIGH);
                    }else if(strLine.equals('OFF')){
                        link.sendPowerPinSwitch(13,IProtocol.POWER_LOW);
                        break;
                    }
                }
            }
            link.disconnect();

            in.close();
        }

        socket.close();
        serverSocket.close();
    }
}
```

Explanation

The above is the code for the PC that connects to Arduino. Keep in mind that in this example, it is the clients that wait, listening to the socket for the main computer to make a connection request. Run the PC clients first before running the main PC.

```
String connectionPort = 'COM4';
        int baudRate = 115200;
Link link = Link.getDefaultInstance();
boolean connected = link.connect(connectionPort, baudRate);
```

The above lines of code deal with *Ardulink*. With the help of the library, communication between the client PC and Arduino can be performed.

```
serverSocket = new ServerSocket(38);
socket = serverSocket.accept();
```

The PC client functions as the listener. In this case, it listens on port 38. The *ServerSocket* class from *java.net* implements the server side of the socket connection.

```
BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
```

When the connection is accepted, it can get messages from the main computer by reading from the socket.

```
if(strLine.equals('ON')){
    link.sendPowerPinSwitch(13,IProtocol.POWER_HIGH);
}else if(strLine.equals('OFF')){
    link.sendPowerPinSwitch(13,IProtocol.POWER_LOW);
    break;
}
```

The above is the code for activating and deactivating the LED light.

submachine2.java

```
import java.net.*;
import java.io.*;

public class SubMachine {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = null;
        Socket socket = null;

        try {
            serverSocket = new ServerSocket(994);
            socket = serverSocket.accept();
        } catch (IOException e) {
            System.err.println('IOException: '+e);
            System.exit(-1);
        } catch (Exception e) {
            System.err.println('Exception: '+e);
            System.exit(-1);
        }

        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        String strLine = null;

        while (true){
            strLine = in.readLine();
            if(strLine.equals('Bye')){
                break;
            }else if(strLine != null){
                System.out.println(strLine);
            }
        }

        in.close();
        socket.close();
        serverSocket.close();
    }
}
```

Explanation

This program is very similar with the previous one. It runs on the second PC client. 994 is the port to which the PC is listening.