JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

DEZSYS-L04 XML-RPC

Systemtechnik Labor
4CHIT 2016/17

Marvin Ertl

Version 1.0

Note:
Betreuer:

Begonnen am 2. Juni 2017
Beendet am 9. Juni 2017

Inhaltsverzeichnis

1 Einführung

Diese Übung soll die Funktionsweise und Implementierung von RPC mit Hilfe des Frameworks XML-RPC demonstrieren. Remote Procedure Call (RPC) ist neben InterProcessCommunication (IPC), Remote Objects (RMI) und Message Oriented Middleware (MOM) eine weitere Moeglichkeit um eine Kommunikation zwischen mehreren Rechnern umzusetzen.

1.1 Ziele

Das Ziel dieser Übung ist die Implementierung der Datenuebertragung von Windkraftanlagen mit Hilfe von Remote Procedure Calls. Nachdem das XML-RPC Demobeispiel durchgefuehrt werden kann, soll ein passendes Applikationsdesign zur Uebertragung der Windkraftdaten erarbeitet werden. Bei jeder Windkraftanlage laeuft ein XML-RPC Service, das XML-RPC Request empfaengt, Daten abfragt und zuruecksendet. Die Abfrage der Daten soll mit einem "Random Value Generatorßimuliert werden. Dabei sollen realistische Daten erzeugt werden.

1.2 Voraussetzungen

- Grundlagen Architektur von verteilten Systemen
- Grundlagen zur Remote Procedure Calls anhand von XML-RPC
- Verwendung des XML-RPC Frameworks (Download)
- Verwendung eines Zufallsgenerators in JAVA
- Anwendung von Sortieralgorithmen in JAVA

1.3 Aufgabenstellung

Ein Windpark besteht aus mehreren Windkraftanlagen. Jeder Windpark hat eine Masteranlage mit einem Zentralrechner, dem sogenannten "Parkrechner". Der Parkrechner sammelt die Daten der einzelnen Windparkanlagen und speichert diese in einer zentralen XML-Datei ab. Die Datenuebtragung erfolgt mit Hilfe von XML-RPC. Das Format der XML-Datei soll sinnvoll gewaehlt werden, wobei jede Windkraftanlage ein XML-Element darstellt und dieses beinhaltet die entsprechende Winddaten.

Folgende Daten werden pro Windkraftanlage erzeugt:

- aktuelle Stromerzeugung
- Blindstrom
- Windgeschwindigkeit
- Drehzahl des Windrades

- Temparatur
- Blattposition

Zu den Daten sollen die Einheiten der Werte in der XML-Datei gespeichert werden. Die Abfrage der Daten bei einer Windkraftanlage soll mit einem "Random Value Generatorßimuliert werden. Dabei sollen Daten mit sinnvollen Werten erzeugt werden.

2 Ergebnisse

2.1 Beispiel ausführen

Es wird ein neues Java Projekt erstellt, in dieses kann dann über die Projekt Properties, Java Build Path die externen Libaries einbinden. Dies sieht wie folgt aus:

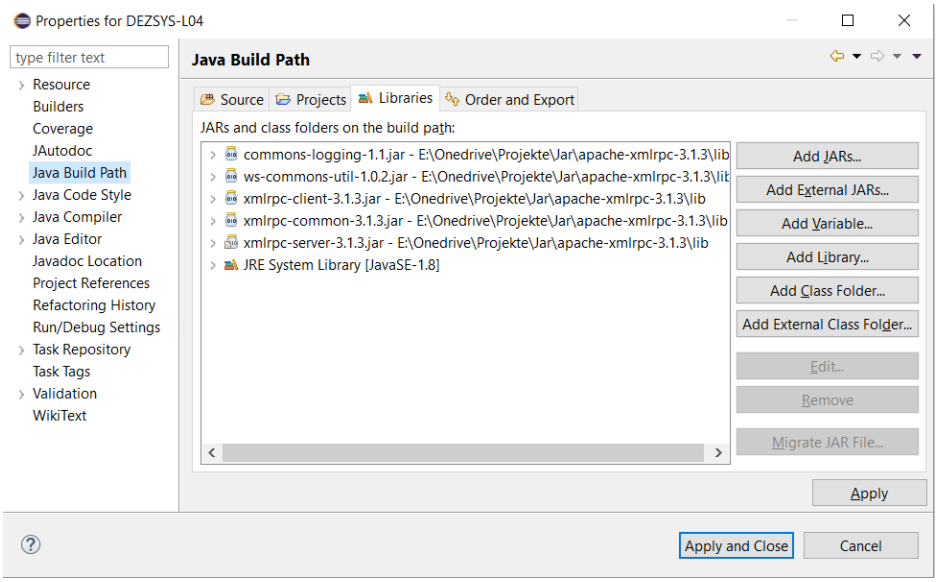

Abbildung 1: Java Build Path

Nun kann man den Server starten und anschließend den Client, nun wird das Ergebnis der Rechnung in der Konsole angezeigt.

2.2 Netzwerksanalyse

Zur Netzwerkanalyse wurde das Linux Tool snort verwendet, dieses muss als erstes mittels äpt-get installïnstalliert werden. Anschließend muss man das Verzeichnis slog erstellen, dies kann man mittels den Befehl "mkdir slog"machen.

Das Tool kann dann mittels folgenden Befehl auf den Loopback Interface gestartet werden.

```
1 sudo snort -vde -i lo -l ./slog
```

Listing 1: Starten der Netzwerkanalyse

Mittels Strg + C kann man die Überwachung beenden. Das erstellte Logfile kann man nun mittels nachfolgenden Befehl in ein normales txt File speichern.

```
1 sudo snort -vde -r ./slog/snort.log.13335132405 > log.txt
```

Listing 2: Speichern in txt Datei

Nun kann man in der Datei den Verlauf der Transaktion der XML Datei verfolgen. In nachfolgenden Packet sieht man das die Werte 99 und 20 gesendet worden sind.

```
06/04-12:19:38.364376 00:00:00:00:00:00 -> 00:00:00:00:00:00 type:0x800 len:0x10B
127.0.0.1:45294 -> 127.0.0.1:8080 TCP TTL:64 TOS:0x0 ID:61978 IpLen:20 DgmLen:253 DF
***AP*** Seq: 0x5BA01AF5  Ack: 0x6CA4A149  Win: 0x156  TcpLen: 32
TCP Options (3) => NOP NOP TS: 697712657 1592454245
3C 3F 78 6D 6C 20 76 65 72 73 69 6F 6E 3D 22 31  <?xml version="1
2E 30 22 20 65 6E 63 6F 64 69 6E 67 3D 22 55 54  .0" encoding="UT
46 2D 38 22 3F 3E 3C 6D 65 74 68 6F 64 43 61 6C  F-8"?><methodCal
6C 3E 3C 6D 65 74 68 6F 64 4E 61 6D 65 3E 43 61  l><methodName>Ca
6C 63 75 6C 61 74 6F 72 2E 61 64 64 3C 2F 6D 65  lculator.add</me
74 68 6F 64 4E 61 6D 65 3E 3C 70 61 72 61 6D 73  thodName><params
3E 3C 70 61 72 61 6D 3E 3C 76 61 6C 75 65 3E 3C  ><param><value><
69 34 3E 39 39 3C 2F 69 34 3E 3C 2F 76 61 6C 75  i4>99</i4></valu
65 3E 3C 2F 70 61 72 61 6D 3E 3C 70 61 72 61 6D  e></param><param
3E 3C 76 61 6C 75 65 3E 3C 69 34 3E 32 30 3C 2F  ><value><i4>20</
69 34 3E 3C 2F 76 61 6C 75 65 3E 3C 2F 70 61 72  i4></value></par
61 6D 3E 3C 2F 70 61 72 61 6D 73 3E 3C 2F 6D 65  am></params></me
74 68 6F 64 43 61 6C 6C 3E                       thodCall>
```

Listing 3: Senden der Werte

In einem späteren Packet kann die Antwort auf das erste XML File sehen. In diesem File ist nun das Ergebnis der Rechnung 119 drinnen.

```
06/04-12:19:38.389416 00:00:00:00:00:00 -> 00:00:00:00:00:00 type:0x800 len:0x133
127.0.0.1:8080 -> 127.0.0.1:45294 TCP TTL:64 TOS:0x0 ID:19136 IpLen:20 DgmLen:293 DF
***AP*** Seq: 0x6CA4A149  Ack: 0x5BA01BBE  Win: 0x167  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1592454252 697712657
48 54 54 50 2F 31 2E 31 20 32 30 30 20 4F 4B 0D  HTTP/1.1 200 OK.
0A 53 65 72 76 65 72 3A 20 41 70 61 63 68 65 20  .Server: Apache
58 4D 4C 2D 52 50 43 20 31 2E 30 0D 0A 43 6F 6E  XML-RPC 1.0..Con
6E 65 63 74 69 6F 6E 3A 20 63 6C 6F 73 65 0D 0A  nection: close..
43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20 74 65  Content-Type: te
78 74 2F 78 6D 6C 0D 0A 43 6F 6E 74 65 6E 74 2D  xt/xml..Content-
4C 65 6E 67 74 68 3A 20 31 33 30 0D 0A 0D 0A 3C  Length: 130....<
3F 78 6D 6C 20 76 65 72 73 69 6F 6E 3D 22 31 2E  ?xml version="1.
30 22 20 65 6E 63 6F 64 69 6E 67 3D 22 55 54 46  0" encoding="UTF
2D 38 22 3F 3E 3C 6D 65 74 68 6F 64 52 65 73 70  -8"?><methodResp
6F 6E 73 65 3E 3C 70 61 72 61 6D 73 3E 3C 70 61  onse><params><pa
72 61 6D 3E 3C 76 61 6C 75 65 3E 3C 69 34 3E 31  ram><value><i4>1
31 39 3C 2F 69 34 3E 3C 2F 76 61 6C 75 65 3E 3C  19</i4></value><
2F 70 61 72 61 6D 3E 3C 2F 70 61 72 61 6D 73 3E  /param></params>
3C 2F 6D 65 74 68 6F 64 52 65 73 70 6F 6E 73 65  </methodResponse
3E                                               >
```

Listing 4: Empfangen des Ergebnis

2.3 Design

Der Parkrechner soll möglichst flexibel auf neue Windräder sowie auf Verbindungsverluste zu Windräder regieren. Um dies Umzusetzen werden die Windräder in einer ArrayList verwaltet.

In einem eigenen Thread durchläuft der Parkrechner alle 500ms die Liste der Windräder und holt sich die aktuellen Informationen der Windräder. Diese Informationen werden dann in eine xml Datei geschrieben.

Wird ein neues Windrad hinzugefügt, werden die aktuellen Informationen des Windrads im nächsten Listendurchgang abgerufen.

Verliert ein Windrad die Verbindung kommt es zu einer TimeOutException, auf diese wird regiert indem das Windrad aus der Liste entfernt wird.

Nachfolgendes UML Diagramm visualisiert die beschriebene Struktur, wobei der Parkrechner der Client ist und jedes Windrad ein Server ist.

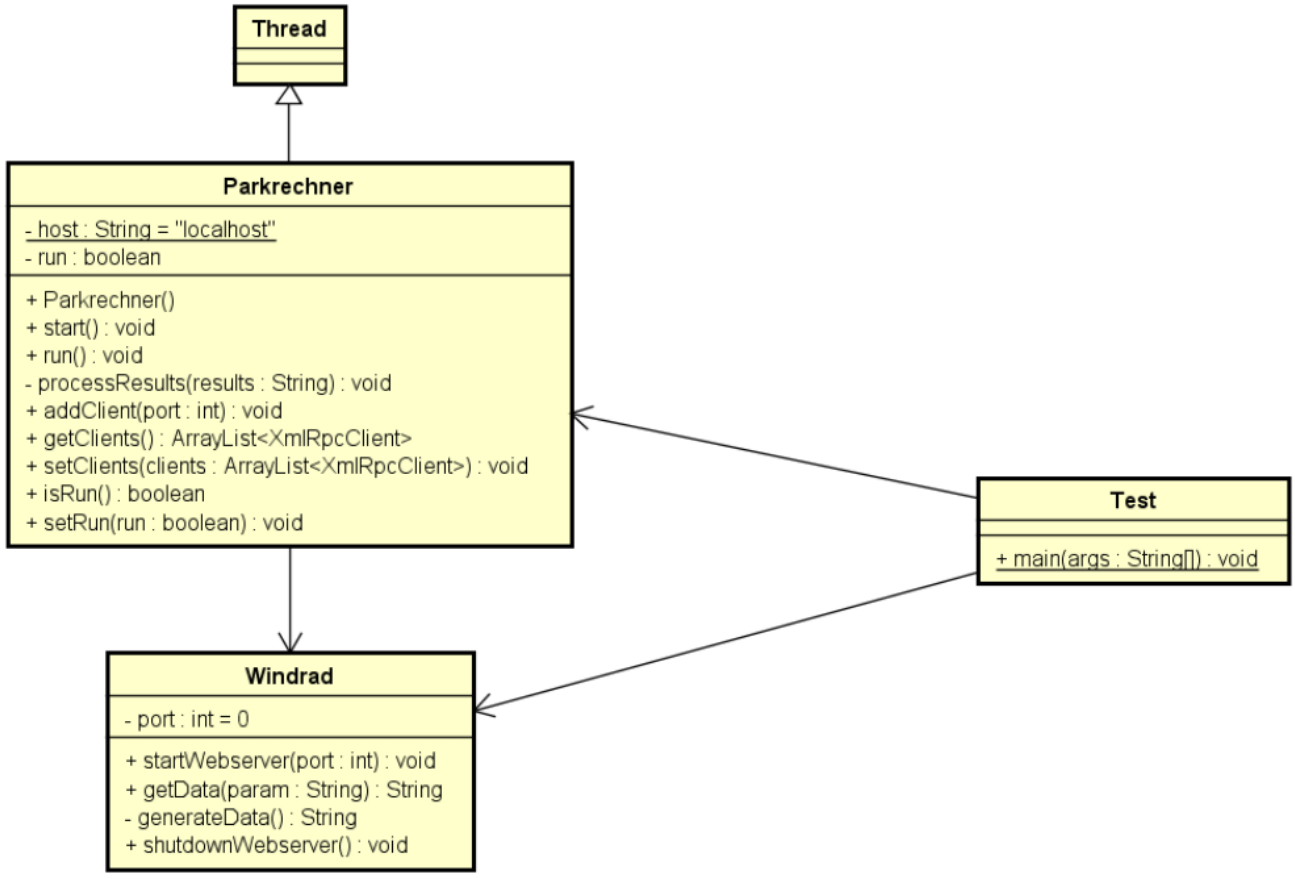

Abbildung 2: UML Diagramm

2.4 Windrad

Das Windrad hat zwei Klassenattribute, einen Webserver und die Portnummer.

```
private WebServer webserver;
private int port = 0;
```

Listing 5: Klassenattribute vom Windrad

Um ein Windrad zu starten muss die Methode startWebserver aufgerufen werden, diese benötigt den Port an dem das Windrad den Webserver startet. An den Property Handler wird die Klasse Windrad hinzugefügt um dann die Methoden der Klasse vom Client aus aufzurufen.

```
public void startWebserver(int port) throws XmlRpcException, IOException {
    this.port = port;

    this.webserver = new WebServer(this.port);
    XmlRpcServer xmlRpcServer = this.webserver.getXmlRpcServer();

    PropertyHandlerMapping phm = new PropertyHandlerMapping();
    phm.addHandler("Windrad", Windrad.class);
    xmlRpcServer.setHandlerMapping(phm);

    this.webserver.start();
    System.out.println("Windrad Port: " + this.port + " gestartet");
}
```

Listing 6: Starten des Windrades

Die Daten die von dem Client geholt werden, werden in der Methode generateData generiert. Hierfür wurden folgende Einheiten und Zahlenbereiche festgelegt.

Information	Wertebereich	Einheit
aktuelle Stromerzeugung	2 bis 5	MW
Blindstrom	0 bis 10	var
Windgeschwindigkeit	6 bis 10	m/s
Blattposition	0 - 180	°
Temperatur	-10 bis + 35	°C
Umdrehungen	5 bis 18	upm

Tabelle 1: Lorem ipsum dolor sit amet

Diese werden wie folgt generiert:

```
private String generateData() {
    StringBuilder data = new StringBuilder();

    /*
    * Data: aktStrom#value#Einheit#Blindstrom#value#Einheit#Wind..#Blatz..#Temp..#Um..
    */
    Random rand = new Random();
    data.append("aktStrom#" + (rand.nextDouble()*3 + 2) + "#" + "MW" + "#");
    data.append("Blindstrom#" + (rand.nextDouble()*3 + 2)*(rand.nextDouble()*0.1) + "#" + "var" + "#");
    data.append("Windgeschwindigkeit#" + (rand.nextDouble()*4 + 6) + "#" + "ms" + "#");
    data.append("Blattposition#" + rand.nextDouble()*180 + "#" + "Grad" + "#");
    data.append("Temperatur#" + (rand.nextDouble()*45 - 10) + "#" + "Grad C" + "#");
```



```
13      data.append("Umdrehungen#" + (rand.nextDouble()*13 + 5) + "#" + "upm");

15      return data.toString();
    }
```

Listing 7: Generien von Random Daten

Abschließend gibt es noch die Methode shutdownWebserver, diese Methode fährt den Webserver herunter und gibt somit den Port wieder frei.

```
public void shutdownWebserver() {
2       this.webserver.shutdown();
    }
```

Listing 8: Shutdown des Webservers

2.5 Parkrechner

Der Parkrechner hat als Klassenattribute eine ArrayList von XmlRpcClient, diese List wird zum Verwalten von den Windrädern verwendet. Weiters gibt es einen hostname und einen boolean Wert run, dieser verwaltet den Thread.

```
1   private ArrayList<XmlRpcClient> clients;
    private static final String host = "localhost";
3   private boolean run;
```

Listing 9: Klassenattribute des Parkrechners

Mittels der addClient Methode können nun neue Windräder hinzugefügt werden. Hierfür wird eine XmlRpcClientConfigImpl erzeugt, diese bekommt als URL den Hostname plus den übergebenen Port. Anschließend wird ein neuer XmlRpcClient erzeugt, dieser bekommt die Config. Der Client wird dann in der ArrayListe abgespeichert.

```
1   public void addClient(int port) {
        XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
3       try {
            config.setServerURL(new URL("http://"+ host + ":" + port + "/xmlrpc") );
5       } catch (MalformedURLException e) {
            System.out.println("Angegebenes Windrad nicht vorhanden!");
7       }
        XmlRpcClient client = new XmlRpcClient();
9       client.setConfig(config);

11      this.clients.add(client);
    }
```

Listing 10: Hinzufügen eines Windrades zum Parkrechner

Mittels den Aufruf von start kann der Thread gestartet werden. In der run Methode wird nun solange der boolean Wert run true ist eine Schleife durchgeführt. Diese Schleife fragt alle Windräder aus der ArrayListe nach den aktuellen Informationen ab. Nach einen Durchgang wird 500 ms gewartet. Die aktuellen Informationen werden dann in einer Methode verarbeitet und in eine Datei gespeichert.

```
    @Override
2   public void run() {
        super.run();
4       while (this.run) {
            StringBuilder results = new StringBuilder();
```

```
        for (int i = 0; i < this.clients.size(); i++) {
            XmlRpcClient client = this.clients.get(i);
            Object[] params = new Object[]{""};
            String result = "";
            try {
                result = (String) client.execute("Windrad.getData", params);
                results.append(result + "\n");
            } catch (XmlRpcException e) {
                this.clients.remove(i);
            }
        }
        this.processResults(results.toString());
        System.out.println("Windrad Informationen aktualisiert");
        try {
            sleep(500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Listing 11: Abfragen der aktuellen Informationen der Windräder

Die Methode processResults verarbeitet nun die Informationen zu einem gültigen XML Format und speichert dieses anschließend in die Datei output.xml.

```
private void processResults(String results) {
    String[] windrad = results.split("\n");

    try {
        XMLOutputFactory xof = XMLOutputFactory.newInstance();
        XMLStreamWriter xtw = null;
        xtw = xof.createXMLStreamWriter(new FileOutputStream("output.xml"), "utf-8");

        xtw.writeStartDocument("utf-8", "1.0");
        xtw.writeCharacters("\n");
        xtw.writeStartElement("windpark");
        xtw.writeCharacters("\n");
        for (String result : windrad) {
            xtw.writeCharacters("\t");
            xtw.writeStartElement("windrad");
            xtw.writeCharacters("\n");

            String[] values = result.split("#");
            for (int i = 0; i < values.length; i += 3) {
                xtw.writeCharacters("\t\t");
                xtw.writeStartElement(values[i]);
                xtw.writeCharacters("\n\t\t\t");
                xtw.writeStartElement("value");
                xtw.writeCharacters(values[i+1]);
                xtw.writeEndElement();
                xtw.writeCharacters("\n\t\t\t");
                xtw.writeStartElement("unit");
                xtw.writeCharacters(values[i+2]);
                xtw.writeEndElement();
                xtw.writeCharacters("\n\t\t");
                xtw.writeEndElement();
                xtw.writeCharacters("\n");
            }
            xtw.writeCharacters("\t");
            xtw.writeEndElement();
            xtw.writeCharacters("\n");
        }
        xtw.writeEndElement();
        xtw.writeCharacters("\n");

        xtw.writeEndDocument();
    } catch (FileNotFoundException | XMLStreamException e) {
        System.out.println("Exception beim Schreiben der XML Datei!");
    }
}
```

Listing 12: Speichern der Informationen als XML

Nachfolgend ist die XML Datei, welche bei 2 Windräder erzeugt wird.

```
<?xml version="1.0" encoding="utf-8"?>
<windpark>
    <windrad>
        <aktStrom>
            <value>3.4209517485778758</value>
            <unit>MW</unit>
        </aktStrom>
        <Blindstrom>
            <value>0.12679542568118224</value>
            <unit>var</unit>
        </Blindstrom>
        <Windgeschwindigkeit>
            <value>7.2858879662333695</value>
            <unit>ms</unit>
        </Windgeschwindigkeit>
        <Blattposition>
            <value>56.30030357654295</value>
            <unit>Grad</unit>
        </Blattposition>
        <Temperatur>
            <value>27.62347930027377</value>
            <unit>Grad C</unit>
        </Temperatur>
        <Umdrehungen>
            <value>7.38533598646653</value>
            <unit>upm</unit>
        </Umdrehungen>
    </windrad>
    <windrad>
        <aktStrom>
            <value>2.6101590270165276</value>
            <unit>MW</unit>
        </aktStrom>
        <Blindstrom>
            <value>0.3090288948779763</value>
            <unit>var</unit>
        </Blindstrom>
        <Windgeschwindigkeit>
            <value>7.484818149729225</value>
            <unit>ms</unit>
        </Windgeschwindigkeit>
        <Blattposition>
            <value>154.72016810106754</value>
            <unit>Grad</unit>
        </Blattposition>
        <Temperatur>
            <value>-8.071407574288642</value>
            <unit>Grad C</unit>
        </Temperatur>
        <Umdrehungen>
            <value>16.495754033033748</value>
            <unit>upm</unit>
        </Umdrehungen>
    </windrad>
</windpark>
```

Listing 13: Informationen zweier Windräder als XML

2.6 Testen

Folgendes File wurde genutzt um die Funktionalität der Klassen zu gewährleisten.

Als erstes werden 4 Windräder erstellt, diese werden zum Parkrechner hinzugefügt. Nach 2 Sekunden wird ein Windrad entfernt, nach 5 Sekunden werden abermals ein Windrad entfernt und nach wieder 5 Sekunden wird ein neues Windrad hinzugefügt.

```
1  public class Test {
       private static ArrayList<Windrad> windrad;
3
       public static void main(String[] args) throws XmlRpcException, IOException {
5          windrad = new ArrayList<>();
           Windrad windrad1 = new Windrad();
7          windrad1.startWebserver(8081);
           windrad.add(windrad1);
9          windrad1 = new Windrad();
           windrad1.startWebserver(8082);
11         windrad.add(windrad1);
           windrad1 = new Windrad();
13         windrad1.startWebserver(8083);
           windrad.add(windrad1);
15         windrad1 = new Windrad();
           windrad1.startWebserver(8084);
17         windrad.add(windrad1);

19         Parkrechner parkrechner = new Parkrechner();
           parkrechner.addClient(8081);
21         parkrechner.addClient(8082);
           parkrechner.addClient(8083);
23         parkrechner.addClient(8084);
           parkrechner.start();
25
           try{
27             Thread.sleep(2000);
           }catch(InterruptedException e){
29             e.printStackTrace();
           }
31
           windrad.get(1).shutdownWebserver();
33
           try{
35             Thread.sleep(5000);
           }catch(InterruptedException e){
37             e.printStackTrace();
           }
39
           windrad.get(2).shutdownWebserver();
41
           try{
43             Thread.sleep(5000);
           }catch(InterruptedException e){
45             e.printStackTrace();
           }
47
           windrad1 = new Windrad();
49         windrad1.startWebserver(8085);
           windrad.add(windrad1);
51         parkrechner.addClient(8085);
       }
53 }
```

Listing 14: Testen der funktionalität der Klassen

Listings

Abbildungsverzeichnis