Software Prototyping & Visualisierung
Projekt

Sommersemester 2015

Name:	______________	Abgabetermin:	28.05.2015 23:55
Mat.Nr.:	______________	Punkte:	______
Aufwand in h:	______	korrigiert:	___

Dieser Übungszettel ist in Zweiergruppen zu lösen. Es reicht, wenn ein Gruppenmitglied die Arbeit im Moodle abgibt.

Teil 1 (32 Punkte) Projekt - Applikation

Entwickeln Sie eine WPF Anwendung, die Sensordaten empfängt und verarbeitet. Sie haben freie Hand welche Sensoren Sie anbinden wollen. Sie können eine beliebige Schnittstelle verwenden, um auf die Daten zuzugreifen.

Entscheiden Sie sich für eine von 2 Möglichkeiten, was Sie mit den Sensordaten machen:

a) **Visualisierung der Sensordaten**
   Stellen Sie die Sensordaten graphisch dar. Achten Sie je nach Sensor auf eine entsprechend sinnvolle Darstellungsform. Die Sensordaten sollen so zeitnah wie möglich dargestellt werden.
b) **Interpretation der Sensordaten**
   Die Sensordaten sollen in der Form interpretiert werden, dass eine Aktion ausgelöst wird. Liefern sie zusätzlich auch ein Testprogramm, in dem man erkennt, was Sie steuern können/könnten.

Sie dürfen für diese Übung auch externe Libraries (z.B. aus Nuget) und fremden Code verwenden. Ziel ist es, einen funktionierenden Prototyp zu entwickeln.

Teil 2 (16 Punkte) Projekt - Präsentation

Zusätzlich zur Abgabe im Moodle müssen Sie ihr Projekt auch in der Übung, am 21.05.2015 präsentieren. Jede Gruppe hat dafür 10min Zeit. Nutzen Sie diese 10 min nicht nur um die Funktionalität ihrer Applikation zu präsentieren, sondern auch, um die interessantesten Code-Passagen zu erklären. Zum Zeitpunkt der Präsentation muss der Prototyp noch nicht fertig sein, jedoch schon so weit entwickelt, dass Sie (Teil-)Funktionalitäten präsentieren können.

Sie können für ihre Präsentation maximal 16 Punkte erhalten. Es wird die Gruppe bewertet und keine Einzelpersonen.

Allgemeine Hinweise: Legen Sie bei der Erstellung Ihrer Übung großen Wert auf eine **saubere Strukturierung** und auf eine **sorgfältige Ausarbeitung**! **Dokumentieren** Sie alle Schnittstellen und versehen Sie Ihre Algorithmen an entscheidenden Stellen ausführlich mit **Kommentaren**! **Testen** Sie ihre Implementierungen ausführlich! Geben Sie **Lösungsideen** an!

1 Dokumentation

1.1 Server Client Kommunikation

Die Kommunikation basiert auf TCP/IP. Kommuniziert wird zwischen einem Server (WPF App) und einem Client (Android App oder C# Testclient). Die Daten werden in XML-Format geschickt. Das Format wird durch eine Klasse, die die Sensorwerte beinhaltet, bestimmt. Diese Klasse wird verwendet um die Daten in XML zu serialisieren bzw. zu deserialisieren.

1.2 Android App

Die Android App hat folgende Bedienelemente:

- Feld für Server IP Adresse
- Feld für Server Port
- Connect Button
- Disconnect Button
- Feld für Anzeige der Sensorwerte

Mithilfe eines Sensor Managers werden die Sensordaten des Androidgerätes in periodischen Abständen ausgelesen. Welcher Sensor ausgelesen werden soll, kann eingestellt werden. Es können auch mehrere Sensoren ausgelesen werden.

1.3 WPF App

Die WPF App hat folgende Elemente:

- Ip Adresse des Servers
- Port des Servers
- Start Button
- Stop Button
- Anzeige eines 3D Objektes

In einem Backgroundworker läuft ein Server welcher auf Daten von einem Client wartet. Sobald Daten vorhanden sind werden die entsprechenden Properties, welche and die GUI gebunden sind (zum Beispiel Winkelwerte), gesetzt. Die 3D Darstellung des Androidgerätes wurden mit einem ModelVisual3D in einem Viewport realisiert.

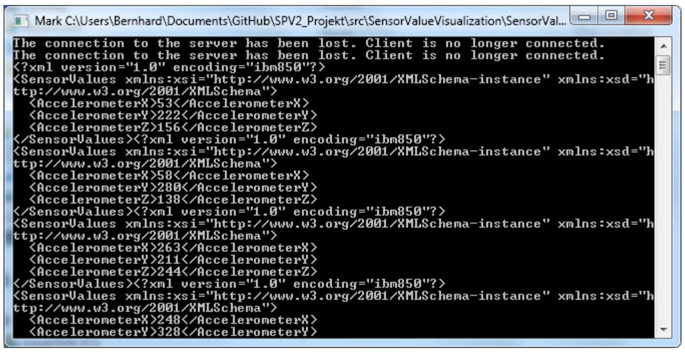
1.4 C# Testclient

Zusätzlich zur Android App wurde für Testzwecke ein C# Testclient entwickelt. Dieser schickt in einem einstellbaren Intervall zufällige Testwerte an den Server.

1.5 Erweiterbarkeit

In der Android App können beliebige Sensorwerte ausgelesen werden. Die Klasse für die Sensordaten muss entsprechend verändert werden. In der WPF Applikation können weitere Tabs hinzugefügt werden in denen dann verschiedene Sensordaten (z.B. Temperatur als Thermometer oder Lichteinfall als Lampe, ...) visualisiert werden können.

2 Test und Screenshots


Mark C:\Users\Bernhard\Documents\GitHub\SPV2_Projekt\src\SensorValueVisualization\SensorVal...

```
The connection to the server has been lost. Client is no longer connected.
The connection to the server has been lost. Client is no longer connected.
<?xml version="1.0" encoding="ibm850"?>
<SensorValues xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="h
ttp://www.w3.org/2001/XMLSchema">
  <AccelerometerX>53</AccelerometerX>
  <AccelerometerY>222</AccelerometerY>
  <AccelerometerZ>156</AccelerometerZ>
</SensorValues><?xml version="1.0" encoding="ibm850"?>
<SensorValues xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="h
ttp://www.w3.org/2001/XMLSchema">
  <AccelerometerX>58</AccelerometerX>
  <AccelerometerY>280</AccelerometerY>
  <AccelerometerZ>138</AccelerometerZ>
</SensorValues><?xml version="1.0" encoding="ibm850"?>
<SensorValues xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="h
ttp://www.w3.org/2001/XMLSchema">
  <AccelerometerX>263</AccelerometerX>
  <AccelerometerY>211</AccelerometerY>
  <AccelerometerZ>244</AccelerometerZ>
</SensorValues><?xml version="1.0" encoding="ibm850"?>
<SensorValues xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="h
ttp://www.w3.org/2001/XMLSchema">
  <AccelerometerX>248</AccelerometerX>
  <AccelerometerY>328</AccelerometerY>
```

Figure 1: C# Testclient

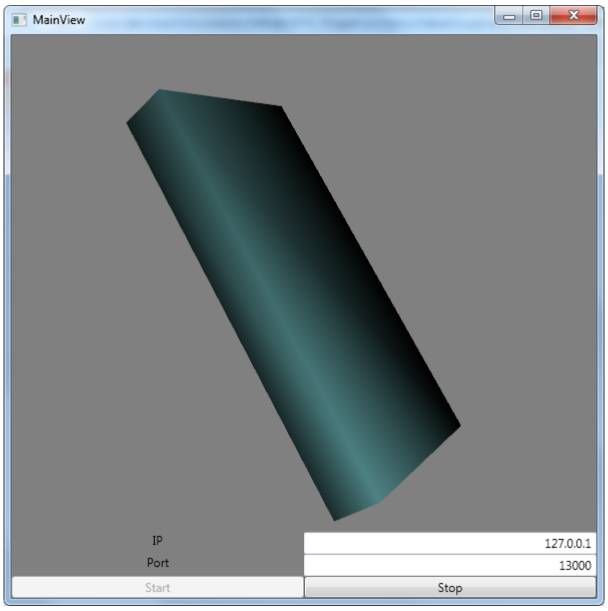


Figure 2: WPF App

3 Source Code

3.1 WPF App

C# Klasse für Sensordaten:

../../src/SensorValueVisualization/SensorValues/SensorValues.cs

```
ï»¿using System;

namespace SensorValuesServer
{
    [Serializable]
    public class SensorValues
    {
        public int AccelerometerX { get; set; }

        public int AccelerometerY { get; set; }

        public int AccelerometerZ { get; set; }

        public override string ToString()
        {
            return String.Format("AccelerometerX: {0}, AccelerometerY: {1},
                AccelerometerZ: {2}", AccelerometerX,
                AccelerometerY, AccelerometerZ);
        }
    }
}
```

WPF App:

../../src/SensorValueVisualization/SensorValueVisualization/View/MainView.xaml

```
ï»¿<Window x:Class="SensorValueVisualization.View.MainView"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainView" Height="600" Width="600"
        DataContext="{Binding Main, Source={StaticResource Locator}}">
    <Grid Background="Gray">
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Viewport3D x:Name="Viewport" Grid.Row="0">
            <Viewport3D.Camera>
                <!--<PerspectiveCamera x:Name="MainCamera" Position="6 5 4"
                    LookDirection="-6 -5 -4" />-->
                <PerspectiveCamera x:Name="ZeroCamera" Position="3 2.125
                    2.5" LookDirection="-1 -1 -1" />
            </Viewport3D.Camera>

            <ModelVisual3D x:Name="TopModelVisual3D">
                <ModelVisual3D.Children>
                    <ModelVisual3D>
                        <ModelVisual3D.Content>
                            <DirectionalLight x:Name="DirLightMain"
                                Direction="-1,-1,-1">
                            </DirectionalLight>
                        </ModelVisual3D.Content>
                    </ModelVisual3D>

                    <ModelVisual3D>
                        <ModelVisual3D.Content>
                            <GeometryModel3D>
                                <GeometryModel3D.Geometry>
                                    <MeshGeometry3D x:Name="MeshMain"
                                        Positions="0 0 0  2 0 0  0 0.25 0  2 0.25 0  0 0 1  2 0 1  0 0.25 1  2 0.25 1"
                                        TriangleIndices="2 3 1  2 1 0  7 1 3  7 5 1  6 5 7  6 4 5  6 2 0  6 0 4  2 7 3  2 6 7  0 1 5  0 5 4 ">
                                    </MeshGeometry3D>
                                </GeometryModel3D.Geometry>
                                <GeometryModel3D.Material>
                                    <DiffuseMaterial>
                                        <DiffuseMaterial.Brush>
                                            <SolidColorBrush Color="CadetBlue"/>
                                        </DiffuseMaterial.Brush>
                                    </DiffuseMaterial>
                                </GeometryModel3D.Material>
                            </GeometryModel3D>
                        </ModelVisual3D.Content>
                        <ModelVisual3D.Transform>
                            <Transform3DGroup>
                                <Transform3DGroup.Children>
                                    <RotateTransform3D CenterX="1" CenterY="0.125" CenterZ="0.5">
                                        <RotateTransform3D.Rotation>
                                            <AxisAngleRotation3D Axis="1 0 0" Angle="{Binding AccelerometerX}" />
                                        </RotateTransform3D.Rotation>
                                    </RotateTransform3D>
                                    <RotateTransform3D CenterX="1" CenterY="0.125" CenterZ="0.5">
                                        <RotateTransform3D.Rotation>
                                            <AxisAngleRotation3D Axis="0 1 0" Angle="{Binding AccelerometerY}" />
                                        </RotateTransform3D.Rotation>
                                    </RotateTransform3D>
                                    <RotateTransform3D CenterX="1" CenterY="0.125" CenterZ="0.5">
                                        <RotateTransform3D.Rotation>
                                            <AxisAngleRotation3D Axis="0 0 1" Angle="{Binding AccelerometerZ}" />
                                        </RotateTransform3D.Rotation>
                                    </RotateTransform3D>
                                </Transform3DGroup.Children>
                            </Transform3DGroup>
                        </ModelVisual3D.Transform>
                    </ModelVisual3D>
                </ModelVisual3D.Children>
            </ModelVisual3D>
        </Viewport3D>

        <Grid Grid.Row="1">
```

```xml
            <Grid.ColumnDefinitions>
                <ColumnDefinition></ColumnDefinition>
                <ColumnDefinition></ColumnDefinition>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition></RowDefinition>
                <RowDefinition></RowDefinition>
                <RowDefinition></RowDefinition>
            </Grid.RowDefinitions>

            <TextBlock Grid.Column="0" Grid.Row="0" HorizontalAlignment="Center">IP</TextBlock>
            <TextBlock Grid.Column="0" Grid.Row="1" HorizontalAlignment="Center">Port</TextBlock>
            <TextBox Grid.Column="1" Grid.Row="0" HorizontalContentAlignment="Right" Text="{Binding IpAdress}"></TextBox>
            <TextBox Grid.Column="1" Grid.Row="1" HorizontalContentAlignment="Right" Text="{Binding Port}"></TextBox>
            <Button Command="{Binding ClickStartCommand}" Grid.Column="0" Grid.Row="2" IsEnabled="{Binding IsDisconnected}">Start</Button>
            <Button Command="{Binding ClickStopCommand}" Grid.Column="1" Grid.Row="2" IsEnabled="{Binding IsConnected}">Stop</Button>
        </Grid>

        <!--<StackPanel Grid.Row="0">
            <Slider x:Name="XAxisSlider" Height="20" HorizontalAlignment="Center" Margin="5" Width="300" Maximum="360" />
            <Slider x:Name="YAxisSlider" Height="20" HorizontalAlignment="Center" Margin="5" Width="300" Maximum="360" />
            <Slider x:Name="ZAxisSlider" Height="20" HorizontalAlignment="Center" Margin="5" Width="300" Maximum="360" />
        </StackPanel>-->

    </Grid>
</Window>
```

Viewmodel:

../../src/SensorValueVisualization/SensorValueVisualization/ViewModel/MainViewModel.cs

```csharp
using System;
using System.ComponentModel;
using System.Diagnostics;
using System.Windows;
using GalaSoft.MvvmLight;
using GalaSoft.MvvmLight.Command;
using SensorValuesServer;

namespace SensorValueVisualization.ViewModel
{
    /// <summary>
    /// This class contains properties that the main View can data bind to.
    /// <para>
    /// Use the <strong>mvvminpc</strong> snippet to add bindable properties to this ViewModel.
    /// </para>
    /// <para>
    /// You can also use Blend to data bind with the tool's support.
    /// </para>
    /// <para>
    /// See http://www.galasoft.ch/mvvm
    /// </para>
    /// </summary>
    public class MainViewModel : ViewModelBase
    {
        private ChatServer _chatServer;
        private BackgroundWorker _chatServerWorker;
        public MainViewModel()
        {
            IpAdress = "127.0.0.1";
            Port = 1234;
            IsConnected = false;
        }

        ~MainViewModel()
        {
            _chatServer.Stop();
            IsConnected = false;
        }

        private string _ipAdress;

        public string IpAdress
        {
            get { return _ipAdress; }
            set
            {
                _ipAdress = value;
                RaisePropertyChanged(() => IpAdress);
            }
        }

        private int _port;

        public int Port
        {
            get { return _port; }
            set
            {
                _port = value;
                RaisePropertyChanged(() => Port);
            }
        }

        private void RunChatServer(object sender, DoWorkEventArgs e)
        {
            _chatServer.Start();
        }

        private void ReadSensorValues(object sender,
           ProgressChangedEventArgs e)
        {
            SensorValues sensorValues = e.UserState as SensorValues;

            if (sensorValues != null)
            {
                AccelerometerX = sensorValues.AccelerometerX;
                AccelerometerY = sensorValues.AccelerometerY;
                AccelerometerZ = sensorValues.AccelerometerZ;
            }
        }

        private int _accelerometerX;
        public int AccelerometerX
        {
            get { return _accelerometerX; }
            set
            {
                _accelerometerX = value;
                RaisePropertyChanged(() => AccelerometerX);
            }
        }

        private int _accelerometerY;
        public int AccelerometerY
        {
            get { return _accelerometerY; }
            set
            {
                _accelerometerY = value;
                RaisePropertyChanged(() => AccelerometerY);
            }
        }

        private int _accelerometerZ;
        public int AccelerometerZ
        {
            get { return _accelerometerZ; }
            set
            {
                _accelerometerZ = value;
                RaisePropertyChanged(() => AccelerometerZ);
            }
        }

        private RelayCommand _clickStartCommand;

        public RelayCommand ClickStartCommand
        {
            get { return _clickStartCommand ?? (_clickStartCommand = new RelayCommand(OnClickStart)); }
        }

        private void OnClickStart()
        {
            _chatServerWorker = new BackgroundWorker
            {
                WorkerReportsProgress = true,
                WorkerSupportsCancellation = true
            };
            _chatServer = new ChatServer(IpAdress, Port, _chatServerWorker);
            try
            {
                _chatServerWorker.DoWork += RunChatServer;
                _chatServerWorker.ProgressChanged += ReadSensorValues;
            }
            catch (Exception e)
            {
                Debug.WriteLine("Unexpected Exception occured.");
                Debug.WriteLine(e.ToString());
            }

            if (!_chatServerWorker.IsBusy)
            {
                _chatServerWorker.RunWorkerAsync();
                IsConnected = true;
            }
        }

        private RelayCommand _clickStopCommand;

        public RelayCommand ClickStopCommand
        {
            get { return _clickStopCommand ?? (_clickStopCommand = new
               RelayCommand(OnClickStop)); }
        }

        private void OnClickStop()
        {
            _chatServer.Stop();
            IsConnected = false;
        }

        private bool _isConnected;

        public bool IsConnected
        {
            get { return _isConnected; }
            set
            {
                IsDisconnected = !value;
                _isConnected = value;
                RaisePropertyChanged(() => IsConnected);
            }
        }

        private bool _isDisconnected;

        public bool IsDisconnected
        {
            get { return _isDisconnected; }
            set
            {
                _isDisconnected = value;
                RaisePropertyChanged(() => IsDisconnected);
            }
        }
    }
}
```

Server:

../../src/SensorValueVisualization/SensorValueVisualization/ViewModel/ChatServer.cs

```
ï»¿using System;
using System.ComponentModel;
using System.Diagnostics;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Runtime.Serialization;
using System.Text;
using System.Threading;
using System.Xml.Serialization;
using SensorValuesServer;

namespace SensorValueVisualization.ViewModel
{
    public class ChatServer
    {
        public string IpAdress { get; private set; }
        public int Port { get; private set; }

        private const int ReadIntervallInMilliseconds = 500;

        private TcpClient _connectedClient;

        private readonly TcpListener _listener;
        //private readonly IFormatter _formatter;
        private readonly XmlSerializer _formatter;
        private bool _isRunning;
        private BackgroundWorker _backgroundWorker;

        public ChatServer(string ipAdress, int port, BackgroundWorker
            backgroundWorker)
        {
            //_formatter = new BinaryFormatter();
            _formatter = new XmlSerializer(typeof(SensorValues));

            IpAdress = ipAdress;
            Port = port;

            IPAddress adress = IPAddress.Parse(ipAdress);
            _listener = new TcpListener(adress, port);

            _backgroundWorker = backgroundWorker;
        }

        public void Start()
        {
            _isRunning = true;

            _listener.Start();
            Console.WriteLine("{0} Server started, now listening for
                clients.", DateTime.Now.ToString("G"));

            while (_isRunning)
            {
                if (!_listener.Pending())
                {
                    Thread.Sleep(500);
                    continue;
                }

                _connectedClient = _listener.AcceptTcpClient();

                try
                {
                    ThreadPool.QueueUserWorkItem(ReadClientMessages, null);
                    Debug.WriteLine("Client has connected properly.");
                }
                catch (InvalidCastException e)
                {
                    Debug.WriteLine("Client has not connected properly.");
                    Debug.WriteLine(e.ToString());
                    _connectedClient.Close();
                }
            }
        }

        public void Stop()
        {
            _isRunning = false;

            if (_connectedClient != null)
            {
                _connectedClient.Close();
            }

            _listener.Stop();
        }

        private void ReadClientMessages(Object obj)
        {
            while (_isRunning && _connectedClient.Connected)
            {
                if (_connectedClient.Connected)
                {
                    NetworkStream stream = _connectedClient.GetStream();
                    {
                        try
                        {
                            //SensorValues sensorValues = (SensorValues)
                               _formatter.Deserialize(stream);

                            StringBuilder receivedXml = new StringBuilder(
                               String.Empty);
                            using (StreamReader reader = new StreamReader(
                               stream))
                            {
                                string receivedLine;

                                while ((receivedLine = reader.ReadLine())
                                   != null)
                                {
                                    receivedXml.AppendLine(receivedLine);
                                }

                                using (Stream xmlStream =
                                   GenerateStreamFromString(receivedXml.
                                   ToString()))
                                {
                                    SensorValues sensorValues = (
                                       SensorValues)_formatter.Deserialize(
                                       xmlStream);
                                    Debug.WriteLine(sensorValues);
                                    _backgroundWorker.ReportProgress(0,
                                       sensorValues);
                                }
                            }
                        }
                        catch (IOException)
                        {
                            //Client closed connection
                        }
                        catch (SerializationException)
                        {
                            //currently no new message
                        }
                        catch (InvalidCastException e)
                        {
                            Debug.WriteLine("Could not cast received
                                message.");
                            Debug.WriteLine(e.ToString());
                        }
                        finally
                        {
                            Thread.Sleep(ReadIntervallInMilliseconds);
                        }
                    }
                }
            }
        }

        private Stream GenerateStreamFromString(string s)
        {
            MemoryStream stream = new MemoryStream();
            StreamWriter writer = new StreamWriter(stream);
            writer.Write(s);
            writer.Flush();
            stream.Position = 0;
            return stream;
        }
    }
}
```

C# Testclient:

../../src/SensorValueVisualization/SensorValuesTestClient/Program.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Xml.Serialization;
using SensorValuesServer;

namespace SensorValuesTestClient
{
    class Program
    {
        private static TcpClient _tcpClient;
        private static XmlSerializer _formatter;
        private static NetworkStream _networkStream;
        private static Random _randomGenerator;

        private const int RandomValueMax = 360;

        static void Main(string[] args)
        {
            _tcpClient = new TcpClient();
            _formatter = new XmlSerializer(typeof(SensorValues));
            _randomGenerator = new Random();

            while (true)
            {
                SendSensorValues(new SensorValues { AccelerometerX =
                    _randomGenerator.Next(0, RandomValueMax), AccelerometerY
                     = _randomGenerator.Next(0, RandomValueMax),
                    AccelerometerZ = _randomGenerator.Next(0, RandomValueMax
                    ) });
                Thread.Sleep(TimeSpan.FromSeconds(1));
            }
        }

        private static void SendSensorValues(SensorValues message)
        {
            IAsyncResult asyncResult = _tcpClient.BeginConnect(IPAddress.
                Parse("127.0.0.1"), 1234, null, null);
            if (!asyncResult.AsyncWaitHandle.WaitOne(TimeSpan.FromSeconds
                (5), false))
            {
                _tcpClient.Close();
                throw new TimeoutException();
            }

            if (_tcpClient.Connected)
            {
                _tcpClient.EndConnect(asyncResult);
                _networkStream = _tcpClient.GetStream();
                _formatter.Serialize(Console.Out, message);
                _formatter.Serialize(_networkStream, message);
                _tcpClient.Client.Shutdown(SocketShutdown.Both);
                _networkStream.Close();
            }
            else
            {
                Console.WriteLine("The connection to the server has been
                    lost. Client is no longer connected.");
            }

            _tcpClient.Close();
            _tcpClient = new TcpClient();
        }
    }
}
```

3.2 Android App

Sensordaten Klasse:

../../src/SensorValuesApp/app/src/main/java/reinhard/sensorvaluesapp/SensorValues.java

```
package reinhard.sensorvaluesapp;

import org.simpleframework.xml.Element;
import org.simpleframework.xml.Root;

```

```java
import java.io.Serializable;

/**
 * Created by Bernhard on 25.05.2015.
 */

@Root
public class SensorValues {

    @Element
    private float AccelerometerX;

    @Element
    private float AccelerometerY;

    @Element
    private float AccelerometerZ;

    public float getAccelerometerX() {
        return AccelerometerX;
    }

    public void setAccelerometerX(float accelerometerX) {
        AccelerometerX = accelerometerX;
    }

    public float getAccelerometerY() {
        return AccelerometerY;
    }

    public void setAccelerometerY(float accelerometerY) {
        AccelerometerY = accelerometerY;
    }

    public float getAccelerometerZ() {
        return AccelerometerZ;
    }

    public void setAccelerometerZ(float accelerometerZ) {
        AccelerometerZ = accelerometerZ;
    }
}
```

TCP Client:

../../src/SensorValuesApp/app/src/main/java/reinhard/sensorvaluesapp/TcpClient.java

```java
package reinhard.sensorvaluesapp;

import android.util.Log;

import org.simpleframework.xml.Attribute;
import org.simpleframework.xml.Element;
import org.simpleframework.xml.Root;
import org.simpleframework.xml.Serializer;
import org.simpleframework.xml.core.Persister;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;

public class TcpClient {

    private String mServerIp;
    private int mServerPort;

    // message to send to the server
    private String mServerMessage;
    // sends message received notifications
    //private OnMessageReceived mMessageListener = null;
    // while this is true, the server will continue running
    private boolean mRun = false;
    // used to send messages
    private PrintWriter mBufferOut;
    // used to read messages from the server
    private BufferedReader mBufferIn;

    /**
     * Constructor of the class. OnMessagedReceived listens for the
        messages received from server
     */
    public TcpClient(String serverIp, int serverPort) {
        //mMessageListener = listener;
        mServerIp = serverIp;
        mServerPort = serverPort;
    }

     // Sends the message entered by client to the server
    public void sendMessage(SensorValues sensorValues) {
        if (mBufferOut != null && !mBufferOut.checkError() && sensorValues
            != null) {
            Serializer serializer = new Persister();
            try {
                serializer.write(sensorValues, mBufferOut);
            }
            catch (Exception e) {
                Log.e("Serialization", "S: Error", e);
            }
        }
    }

    /**
     * Close the connection and release the members
     */
    public void stop() {
        Log.i("Debug", "stop");

        // send mesage that we are closing the connection
        //sendMessage(Constants.CLOSED_CONNECTION + "Kazy");

        mRun = false;

        if (mBufferOut != null) {
            mBufferOut.flush();
            mBufferOut.close();
        }

        //mMessageListener = null;
        mBufferIn = null;
        mBufferOut = null;
        mServerMessage = null;
    }

    public void run() {

        mRun = true;

        try {
            //here you must put your computer's IP address.
            InetAddress serverAddr = InetAddress.getByName(mServerIp);

            Log.i("TCP Client", "C: Connecting...");

            Log.i("Server Ip", serverAddr.getHostAddress());
            Log.i("Server Port", Integer.toString(mServerPort));

            //create a socket to make the connection with the server
            Socket socket = new Socket(serverAddr, mServerPort);

            try {
                Log.i("Debug", "inside try catch");
                //sends the message to the server
                mBufferOut = new PrintWriter(new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())), true);

                //receives the message which the server sends back
                mBufferIn = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                // send login name
                //sendMessage(Constants.LOGIN_NAME + PreferencesManager.getInstance().getUserName());
                //sendMessage("Hi");
                //in this while the client listens for the messages sent by the server
                while (mRun) {
                    mServerMessage = mBufferIn.readLine();
                    //if (mServerMessage != null && mMessageListener != null) {
                    //    //call the method messageReceived from MyActivity class
                    //    mMessageListener.messageReceived(mServerMessage);
                    //}

                }
                Log.e("RESPONSE FROM SERVER", "S: Received Message: '" + mServerMessage + "'");

            } catch (Exception e) {

                Log.e("TCP", "S: Error", e);

            } finally {
                //the socket must be closed. It is not possible to reconnect to this socket
                // after it is closed, which means a new socket instance has to be created.
                socket.close();
            }

        } catch (Exception e) {

            Log.e("TCP", "C: Error", e);

        }

    }

    //Declare the interface. The method messageReceived(String message)
       will must be implemented in the MyActivity
    //class at on asynckTask doInBackground
    public interface OnMessageReceived {
        public void messageReceived(String message);
    }
}
```

Android App:

../../src/SensorValuesApp/app/src/main/java/reinhard/sensorvaluesapp/SensorValuesActivity.java

```java
package reinhard.sensorvaluesapp;

import android.app.Activity;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;


public class SensorValuesActivity extends Activity implements
    SensorEventListener {

    private SensorManager senSensorManager;
    private Sensor senAccelerometer;

    private long lastUpdate = 0;
    private float last_x, last_y, last_z;
    private static final int SHAKE_THRESHOLD = 600;

    private Button connectButton;
    private Button disconnectButton;

    TcpClient tcpClient;

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

        senSensorManager = (SensorManager) getSystemService(Context.
            SENSOR_SERVICE);
        senAccelerometer = senSensorManager.getDefaultSensor(Sensor.
            TYPE_ACCELEROMETER);
        senSensorManager.registerListener(this, senAccelerometer ,
            SensorManager.SENSOR_DELAY_NORMAL);

        connectButton = (Button) findViewById(R.id.btnConnect);
        //connectButton.setEnabled(true);
        disconnectButton = (Button) findViewById(R.id.btnDisconnect);
        //disconnectButton.setEnabled(false);
    }

    protected void onPause() {
        super.onPause();
        senSensorManager.unregisterListener(this);
    }

    protected void onResume() {
        super.onResume();
        senSensorManager.registerListener(this, senAccelerometer,
            SensorManager.SENSOR_DELAY_NORMAL);
    }

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

    @Override
    public void onSensorChanged(SensorEvent event) {
        Sensor mySensor = event.sensor;

        if (mySensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            SensorValues sensorValues = new SensorValues();
            sensorValues.setAccelerometerX(event.values[0]);
            sensorValues.setAccelerometerY(event.values[1]);
            sensorValues.setAccelerometerZ(event.values[2]);

            long curTime = System.currentTimeMillis();

            if ((curTime - lastUpdate) > 100) {
                long diffTime = (curTime - lastUpdate);
                lastUpdate = curTime;

                TextView textView = (TextView) findViewById(R.id.lblAccelerometerValues);
                textView.setText(String.format("X: %s, Y: %s, Z: %s", sensorValues.getAccelerometerX(), sensorValues.getAccelerometerY(), sensorValues.getAccelerometerZ()));

                // check if client is connected
                if (tcpClient != null) {
                    tcpClient.sendMessage(sensorValues);
                }
            }
        }
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {

    }

    public void OnClickConnect(View view) {
        new ConnectTask().execute("");

        // TODO: check if connected

        disconnectButton.setEnabled(true);
        connectButton.setEnabled(false);
    }

    public void OnClickDisconnect(View view) {
        tcpClient.stop();
        connectButton.setEnabled(true);
        disconnectButton.setEnabled(false);
    }

    public class ConnectTask extends AsyncTask<String, String, TcpClient> {

        @Override
        protected TcpClient doInBackground(String... message) {
            EditText serverIp = (EditText) findViewById(R.id.tbServerIp);
            EditText serverPort = (EditText) findViewById(R.id.tbServerPort);
            tcpClient = new TcpClient(serverIp.getText().toString(), Integer.parseInt(serverPort.getText().toString()));
            tcpClient.run();

            return null;
        }
    }
}
```

Android Oberfläche:

../../src/SensorValuesApp/app/src/main/res/layout/activity_sensor_values.xml

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".SensorValuesActivity">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:text="Server Ip:"
            android:id="@+id/lblServerIp" />

        <EditText
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:inputType="phone"
            android:ems="10"
            android:id="@+id/tbServerIp"
            android:text="10.0.3.2"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:text="Server Port:"
            android:id="@+id/lblServerPort" />

        <EditText
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:inputType="phone"
            android:ems="10"
            android:id="@+id/tbServerPort"
            android:text="1234"/>

        <Space
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:minHeight="10dp" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Connect"
            android:id="@+id/btnConnect"
            android:onClick="OnClickConnect"/>

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Disconnect"
            android:id="@+id/btnDisconnect"
            android:onClick="OnClickDisconnect"/>

        <Space
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:minHeight="25dp" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:text="Acceleremoter Values:"
            android:id="@+id/lblAccelerometer" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:text="n/a"
            android:id="@+id/lblAccelerometerValues" />

        </LinearLayout>

</RelativeLayout>
```