Übung Nr.: 1

HTBLA Kaindorf
Höhere Lehranstalt für Mechatronik,
Schwerpunkt Robotik

Jahrgang: 5BHME (23/24)
Gruppe: 2B
Lehrer: SN

htblakaindorf
Leistung mit Menschlichkeit

Übungsdatum: 12.09.2024 – 30.09.2024

Abgabe: Mo, 07.10.2024

Note: .....

Technischer Bericht

Thema: Automatisierte Frequenzlinienmessung

Übungsdatum: 12. September 2024 – 30. September 2024
Übungszeit: 10:45 – 13:15
Übungsort: Kaindorf, PRR Labor

Teilgruppe: 2B – Gollien Florian, Peterlin Fabian, Pronegg Christoph

Aufgabenstellung

Automatisierte Frequenzlinienmessung über serielle Schnittstelle mit WG810 Funktionsgenerator und Fluke 45 Multimeter.

Resümee

Werte können über die GUI eingestellt und angezeigt werde. Swing Worker unvollständig.

Unterschriften:

Inhaltsverzeichnis

Zeitplan

12.09.2024:
Besprechung der Aufgabenstellung
Kommunikation zwischen Fluke45, WG810 und PC herstellen

19.09.2024:
GUI programmieren

23.09.2024:
Gesamte Teilgruppe krank

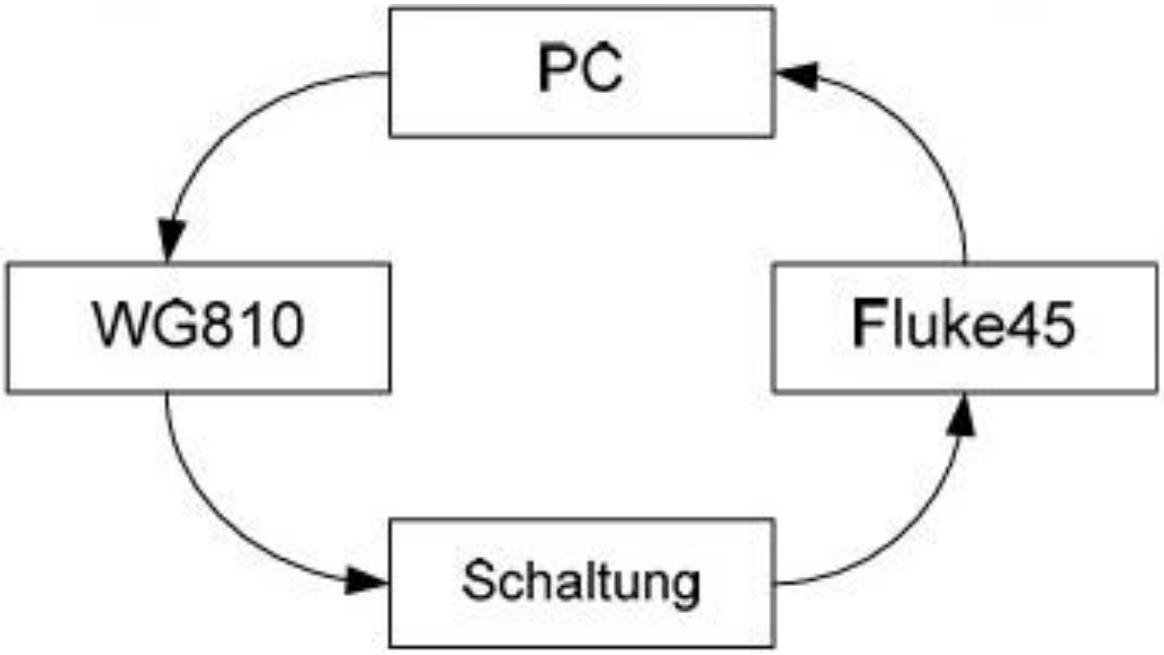
30.09.2024
Werte vom Multimeter (Fluke45) einlesen und in der GUI ausgeben
Programmierung Swingworker (nicht fertig)

Aufgabenstellung

Schreiben Sie ein Programm am PC unter JAVA, das mit Hilfe eines Frequenzgenerators und eines
Messgerätes den Amplitudengang eines beliebigen Vierpols aufnehmen kann.
gegeben:

- Klasse SerialInterface
- Klasse WG810
- Klasse Fluke45 (rudimentär)

a) Lesen Sie über eine einfache Methode Daten vom Fluke45 Multimeter über die serielle Schnittstelle.
b) Schreiben Sie ein vollständiges Programm, das den Amplitudengang eines Vierpols aufnehmen kann. (WG810 einstellen, Daten in Liste speichern, Liste in Datei speichern)
c) Erweitern Sie das Programm für das automatische Aufnehmen des Amplitudengangs eines Vierpols. (Anfangsfrequenz, Endfrequenz, Anzahl der Punkte, Amplitude, Schrittzeit, …)
d) Erweitern Sie die Klasse Fluke45 um ‚sinnvolle' Funktionen.



GUI

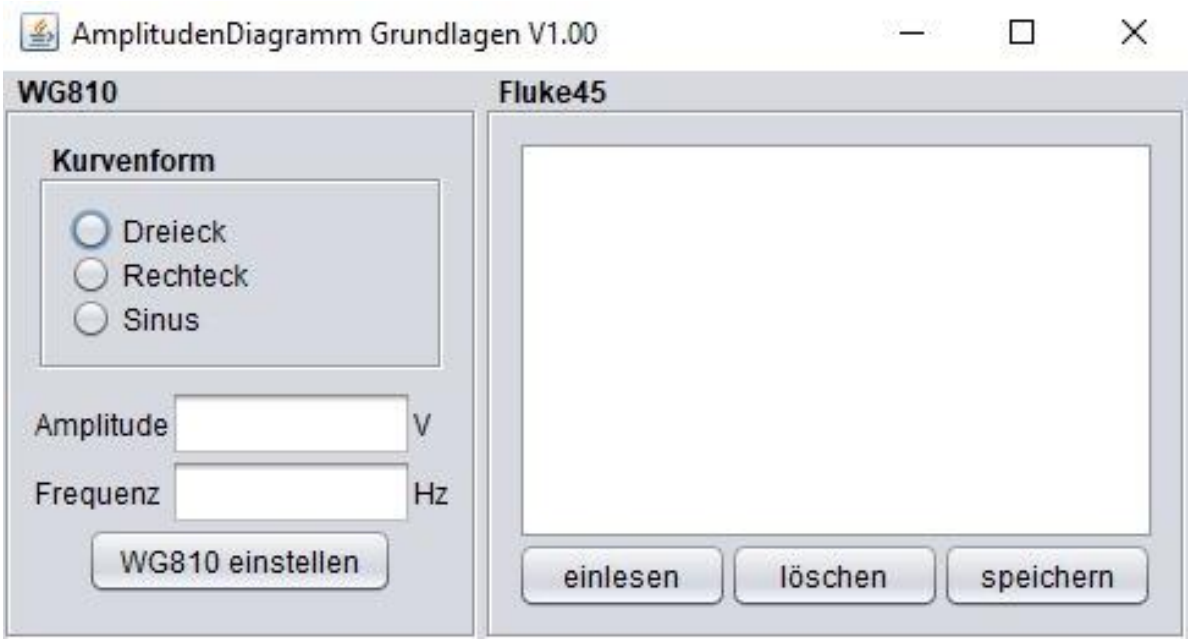


Abbildung 1: GUI am 19.09. erstellt

Aufbau

Source Code GUI

```
package serial.gui;

import java.util.*;

import javax.swing.*;
import serial.Fluke45V2019;
import serial.WG810V2019;
import serial.data.SerialOutput;
import serial.gui.SerialListModel;

/**
 *
 * @author fgoll
 */
public class SerialGUI extends javax.swing.JFrame
{
  private final List<SerialOutput> serialOutputs = new ArrayList<>();

  /**
   * Creates new form SerialGUI
   */
  public SerialGUI()
  {
    initComponents();
  }

  /**
   * This method is called from within the constructor to initialize the form.
   * WARNING: Do NOT modify this code. The content of this method is always
   * regenerated by the Form Editor.
   */
  @SuppressWarnings("unchecked")
  // <editor-fold defaultstate="collapsed" desc="Generated Code">
  private void initComponents()
  {
    java.awt.GridBagConstraints gridBagConstraints;

    btGroupKurvenform = new javax.swing.ButtonGroup();
    pMain = new javax.swing.JPanel();
    pCenter = new javax.swing.JPanel();
    pList = new javax.swing.JPanel();
    jScrollPane1 = new javax.swing.JScrollPane();
    listenfeld = new javax.swing.JList<>();
    pButton = new javax.swing.JPanel();
    btEinlesen = new javax.swing.JButton();
    btLoeschen = new javax.swing.JButton();
    btSpeichern = new javax.swing.JButton();
    pWest = new javax.swing.JPanel();
    pKurvenform = new javax.swing.JPanel();
    rbDreieck = new javax.swing.JRadioButton();
    rbRechteck = new javax.swing.JRadioButton();
    rbSinus = new javax.swing.JRadioButton();
    pEinstellung = new javax.swing.JPanel();
    tfAmplitude = new javax.swing.JFormattedTextField();
    tfFrequenz = new javax.swing.JFormattedTextField();
```

Importieren der gegebenen Klassen

Eine Liste wird erstellt.

```
    btWG810 = new javax.swing.JButton();
    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    jLabel4 = new javax.swing.JLabel();

    setDefaultCloseOperation(javax.swing.WindowConstants.DO_NOTHING_ON_CLOSE);
    setTitle("AmplitudenDiagrammV1.0");
    addWindowListener(new java.awt.event.WindowAdapter()
    {
      public void windowClosing(java.awt.event.WindowEvent evt)
      {
        onWindowClosing(evt);
      }
    });

    pMain.setLayout(new java.awt.BorderLayout());


pCenter.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.cre
ateTitledBorder(javax.swing.BorderFactory.createEtchedBorder(), "Fluke45",
javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.TOP),
javax.swing.BorderFactory.createEmptyBorder(1, 1, 1, 1)));
    pCenter.setLayout(new java.awt.BorderLayout());

    pList.setLayout(new java.awt.BorderLayout(8, 8));

    jScrollPane1.setViewportView(listenfeld);

    pList.add(jScrollPane1, java.awt.BorderLayout.CENTER);

    pCenter.add(pList, java.awt.BorderLayout.CENTER);

    pButton.setLayout(new java.awt.GridLayout(1, 3, 8, 8));

    btEinlesen.setText("einlesen");
    btEinlesen.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(java.awt.event.ActionEvent evt)
      {
        onEinlesen(evt);
      }
    });
    pButton.add(btEinlesen);

    btLoeschen.setText("löschen");
    btLoeschen.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(java.awt.event.ActionEvent evt)
      {
        onLoeschen(evt);
      }
    });
    pButton.add(btLoeschen);

    btSpeichern.setText("speichern");
    btSpeichern.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(java.awt.event.ActionEvent evt)
      {
```

```
        onSpeichern(evt);
      }
    });
    pButton.add(btSpeichern);

    pCenter.add(pButton, java.awt.BorderLayout.SOUTH);

    pMain.add(pCenter, java.awt.BorderLayout.CENTER);


pWest.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.creat
eTitledBorder(javax.swing.BorderFactory.createEtchedBorder(), "WG810",
javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.TOP),
javax.swing.BorderFactory.createEtchedBorder()));
    pWest.setLayout(new java.awt.GridLayout(2, 1));


pKurvenform.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory
.createTitledBorder(javax.swing.BorderFactory.createEtchedBorder(), "Kurvenform",
javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.TOP),
javax.swing.BorderFactory.createEtchedBorder()));
    pKurvenform.setLayout(new java.awt.GridLayout(3, 0));

    btGroupKurvenform.add(rbDreieck);
    rbDreieck.setText("Dreieck");
    pKurvenform.add(rbDreieck);

    btGroupKurvenform.add(rbRechteck);
    rbRechteck.setText("Rechteck");
    pKurvenform.add(rbRechteck);

    btGroupKurvenform.add(rbSinus);
    rbSinus.setSelected(true);
    rbSinus.setText("Sinus");
    pKurvenform.add(rbSinus);

    pWest.add(pKurvenform);

    java.awt.GridBagLayout pEinstellungLayout = new java.awt.GridBagLayout();
    pEinstellungLayout.columnWidths = new int[] {0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0};
    pEinstellungLayout.rowHeights = new int[] {0, 8, 0, 8, 0};
    pEinstellung.setLayout(pEinstellungLayout);

    tfAmplitude.setColumns(10);
    tfAmplitude.setValue(0.0);
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 2;
    gridBagConstraints.gridy = 0;
    pEinstellung.add(tfAmplitude, gridBagConstraints);

    tfFrequenz.setColumns(10);
    tfFrequenz.setValue(0.0);
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 2;
    gridBagConstraints.gridy = 2;
    pEinstellung.add(tfFrequenz, gridBagConstraints);

    btWG810.setText("WG810 einstellen");
    btWG810.addActionListener(new java.awt.event.ActionListener()
    {
```

```
    public void actionPerformed(java.awt.event.ActionEvent evt)
    {
      onWG810Einstellen(evt);
    }
  });
  gridBagConstraints = new java.awt.GridBagConstraints();
  gridBagConstraints.gridx = 0;
  gridBagConstraints.gridy = 4;
  gridBagConstraints.gridwidth = 5;
  pEinstellung.add(btWG810, gridBagConstraints);

  jLabel1.setText("Amplitude");
  gridBagConstraints = new java.awt.GridBagConstraints();
  gridBagConstraints.gridx = 0;
  gridBagConstraints.gridy = 0;
  gridBagConstraints.anchor = java.awt.GridBagConstraints.LINE_START;
  pEinstellung.add(jLabel1, gridBagConstraints);

  jLabel2.setText("Frequenz");
  gridBagConstraints = new java.awt.GridBagConstraints();
  gridBagConstraints.gridx = 0;
  gridBagConstraints.gridy = 2;
  gridBagConstraints.anchor = java.awt.GridBagConstraints.LINE_START;
  pEinstellung.add(jLabel2, gridBagConstraints);

  jLabel3.setText("V");
  gridBagConstraints = new java.awt.GridBagConstraints();
  gridBagConstraints.gridx = 4;
  gridBagConstraints.gridy = 0;
  gridBagConstraints.anchor = java.awt.GridBagConstraints.LINE_START;
  pEinstellung.add(jLabel3, gridBagConstraints);

  jLabel4.setText("Hz");
  gridBagConstraints = new java.awt.GridBagConstraints();
  gridBagConstraints.gridx = 4;
  gridBagConstraints.gridy = 2;
  gridBagConstraints.anchor = java.awt.GridBagConstraints.LINE_START;
  pEinstellung.add(jLabel4, gridBagConstraints);

  pWest.add(pEinstellung);

  pMain.add(pWest, java.awt.BorderLayout.WEST);

  getContentPane().add(pMain, java.awt.BorderLayout.CENTER);

  pack();
}// </editor-fold>

private void onWindowClosing(java.awt.event.WindowEvent evt)
{
  onX();
}
```

Bei Event onWindowClosing wird die Methode onX aufgerufen

```
  private void onEinlesen(java.awt.event.ActionEvent evt)
  {
    double messwert;
    try (Fluke45V2019 mg1 = new Fluke45V2019("COM15"))
    {
      messwert = mg1.getValue();

      final double frequenz
        = // Zahl aus dem formatierten Eingabefeld holen
        ((Number) tfFrequenz.getValue()).doubleValue();

      final double amplitude
        = // Zahl aus dem formatierten Eingabefeld holen
        ((Number) tfAmplitude.getValue()).doubleValue();

      String kurvenform = "SIN";

      if (rbDreieck.isSelected())
      {
        kurvenform = "TRI";

      }
      else if (rbRechteck.isSelected())
      {
        kurvenform = "REC";
      }

      SerialOutput serialoutput = new SerialOutput(kurvenform, amplitude, frequenz, messwert);

      serialOutputs.add(serialoutput);

      listenfeld.setModel(new SerialListModel(serialOutputs));


    }
    catch (Exception ex)
    {
      System.out.println(ex.getMessage());
    }


  }

  private void onLoeschen(java.awt.event.ActionEvent evt)
  {
//////////*    // TODO add your handling code here:
  }
```

COM Port einstellen, neues Objekt erstellen

Werte aus den Textfeldern und Radiobuttons holen

Neues Objekt der Klasse SerialOutput erstellen, Variablen werden als Parameter an den Konstruktor übergeben

serialoutput wird zur Liste serialOutputs hinzugefügt

Listenfeld wird auf ein neues Objekt vom Typ serialListModel gesetzt

```
  private void onSpeichern(java.awt.event.ActionEvent evt)
  {
    // TODO add your handling code here:
  }

  private void onWG810Einstellen(java.awt.event.ActionEvent evt)
  {

    final double frequenz
      = // Zahl aus dem formatierten Eingabefeld holen
      ((Number) tfFrequenz.getValue()).doubleValue();

    final double amplitude
      = // Zahl aus dem formatierten Eingabefeld holen
      ((Number) tfAmplitude.getValue()).doubleValue();

    try (WG810V2019 wg810 = new WG810V2019("com13"))
    {
      wg810.setRemote(true);
      wg810.setAmplitude(amplitude, true);
      wg810.setFrequenz(frequenz, true);
      if (rbSinus.isSelected())
      {
        wg810.setKurvenform(WG810V2019.FORMSIN, 0, true);
      }
      else if (rbDreieck.isSelected())
      {
        wg810.setKurvenform(WG810V2019.FORMTRI, 0, true);

      }
      else if (rbRechteck.isSelected())
      {
        wg810.setKurvenform(WG810V2019.FORMRECT, 0, true);
      }
      wg810.setRemote(false);
    }
    catch (Exception ex)
    {
      JOptionPane.showMessageDialog(this, ex.getMessage(), "Fehler aufgtreten",
JOptionPane.ERROR_MESSAGE);
    }
  }

  private void onX()
  {
    if (JOptionPane.showConfirmDialog(this, "Wirklich beenden?", "Sicherheitsabfrage",
JOptionPane.YES_NO_OPTION) == JOptionPane.YES_NO_OPTION)
      dispose();
  }

  /**
   * @param args the command line arguments
   */
  public static void main(String args[])
  {
    /* Set the Nimbus look and feel */
    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
```

onSpeichern und onLoeschen wurde nicht ausprogrammiert da keine Priorität dafür bestand

onWG810Einstellen liest die Werte aus den Textfeldern aus und schreibt sie mit wg810.set…. in den Funktionsgenerator
Die Kurvenform wird mit if Clauses abgefragt und eingestellt

Funktion onX ruft einen Dialogfenster auf, der den Benutzer fragt, ob er wirklich beenden möchte. Wenn die Bedingung erfüllt ist (Ja) wird das Fenster geschlossen

```
    /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and
feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
     */
    try
    {
      for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels())
      {
        if ("Nimbus".equals(info.getName()))
        {
          javax.swing.UIManager.setLookAndFeel(info.getClassName());
          break;
        }
      }
    }
    catch (ClassNotFoundException ex)
    {

java.util.logging.Logger.getLogger(SerialGUI.class.getName()).log(java.util.logging.Level.SEVE
RE, null, ex);
    }
    catch (InstantiationException ex)
    {

java.util.logging.Logger.getLogger(SerialGUI.class.getName()).log(java.util.logging.Level.SEVE
RE, null, ex);
    }
    catch (IllegalAccessException ex)
    {

java.util.logging.Logger.getLogger(SerialGUI.class.getName()).log(java.util.logging.Level.SEVE
RE, null, ex);
    }
    catch (javax.swing.UnsupportedLookAndFeelException ex)
    {

java.util.logging.Logger.getLogger(SerialGUI.class.getName()).log(java.util.logging.Level.SEVE
RE, null, ex);
    }
    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable()
    {
      public void run()
      {
        new SerialGUI().setVisible(true);
      }
    });
  }

  // Variables declaration - do not modify
  private javax.swing.JButton btEinlesen;
  private javax.swing.ButtonGroup btGroupKurvenform;
  private javax.swing.JButton btLoeschen;
  private javax.swing.JButton btSpeichern;
  private javax.swing.JButton btWG810;
  private javax.swing.JLabel jLabel1;
  private javax.swing.JLabel jLabel2;
```

```
  private javax.swing.JLabel jLabel3;
  private javax.swing.JLabel jLabel4;
  private javax.swing.JScrollPane jScrollPane1;
  private javax.swing.JList<SerialOutput> listenfeld;
  private javax.swing.JPanel pButton;
  private javax.swing.JPanel pCenter;
  private javax.swing.JPanel pEinstellung;
  private javax.swing.JPanel pKurvenform;
  private javax.swing.JPanel pList;
  private javax.swing.JPanel pMain;
  private javax.swing.JPanel pWest;
  private javax.swing.JRadioButton rbDreieck;
  private javax.swing.JRadioButton rbRechteck;
  private javax.swing.JRadioButton rbSinus;
  private javax.swing.JFormattedTextField tfAmplitude;
  private javax.swing.JFormattedTextField tfFrequenz;
  // End of variables declaration
```

List Model

```
package serial.gui;

import java.util.*;
import javax.swing.*;
import serial.data.SerialOutput;

/**
 *
 * @author fgoll
 */
public class SerialListModel extends AbstractListModel<SerialOutput>
```

Liste verwaltet Objekte vom Typ SerialOutput

```
{
  private final List<SerialOutput> serialOutputs;

  public SerialListModel(List<SerialOutput> serialOutputs)
  {
    this.serialOutputs = serialOutputs;
  }

  @Override
  public int getSize()
  {
    return serialOutputs.size();
  }

  @Override
  public SerialOutput getElementAt(int index)
  {
    return serialOutputs.get(index);
  }
}
```

Konstruktor

Gibt Anzahl der Elemente in der Liste zurück

Gibt das SerialOutput-Objekt an der gegebenen Indexposition in der Liste zurück

Datenhaltungsklasse

package serial.data;

```
/**
 *
 * @author fgoll
 */
public record SerialOutput(String kurvenform, double frequenz, double amplitude, double
messwert )
```

SerialOutput ist ein Record, für die Felder wird der Konstruktor automatisch generiert

```
{
  @Override
  public String toString()
  {
    return String.format("%s:%.3fHZ:%.3fV:%.3fV",kurvenform,frequenz,amplitude,messwert);
  }
}
```

Override Annotation formatiert die Ausgabe in eine Zeichenkette

12.09.: Kommunikation zwischen Fluke45, WG810 und PC herstellen

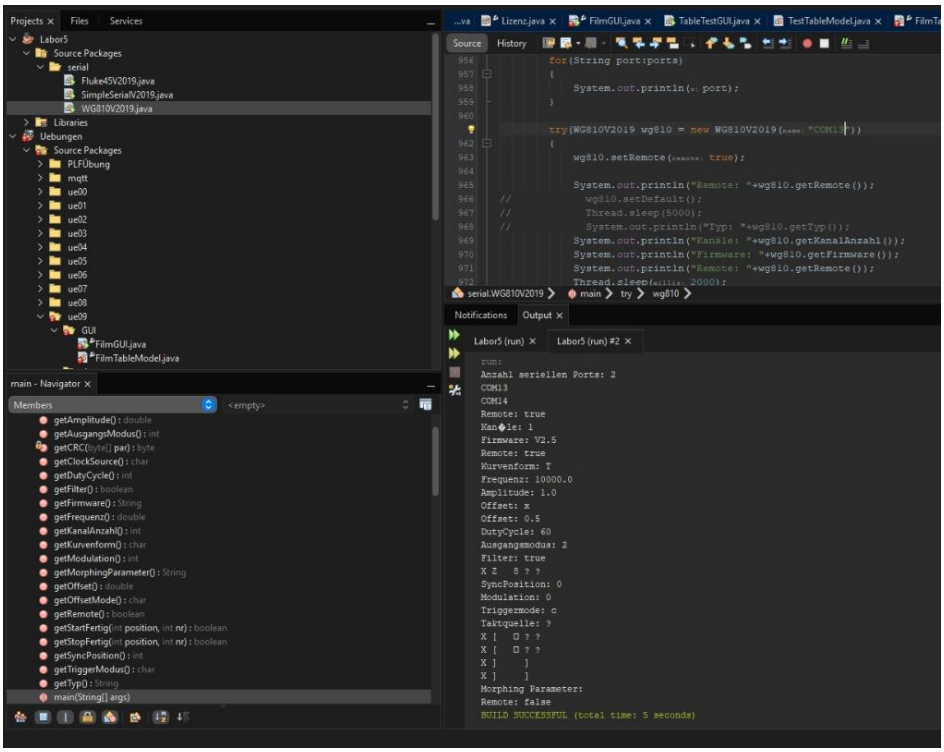


Abbildung 3: WG810 auf vorgewählte Werte einsellen

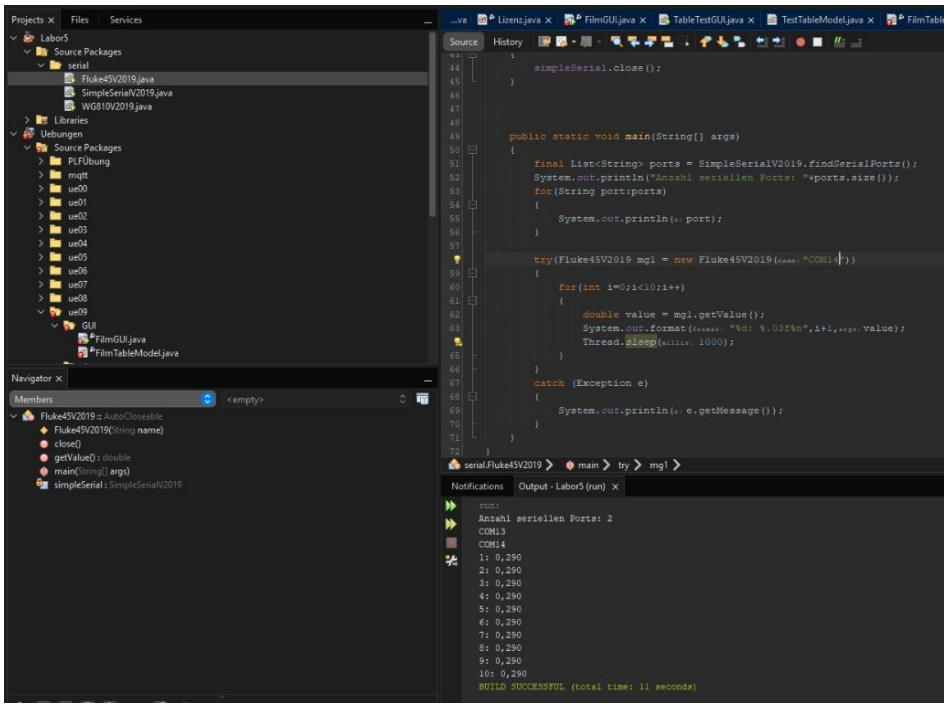


Abbildung 2: Spannung von Fluke45 auslesen

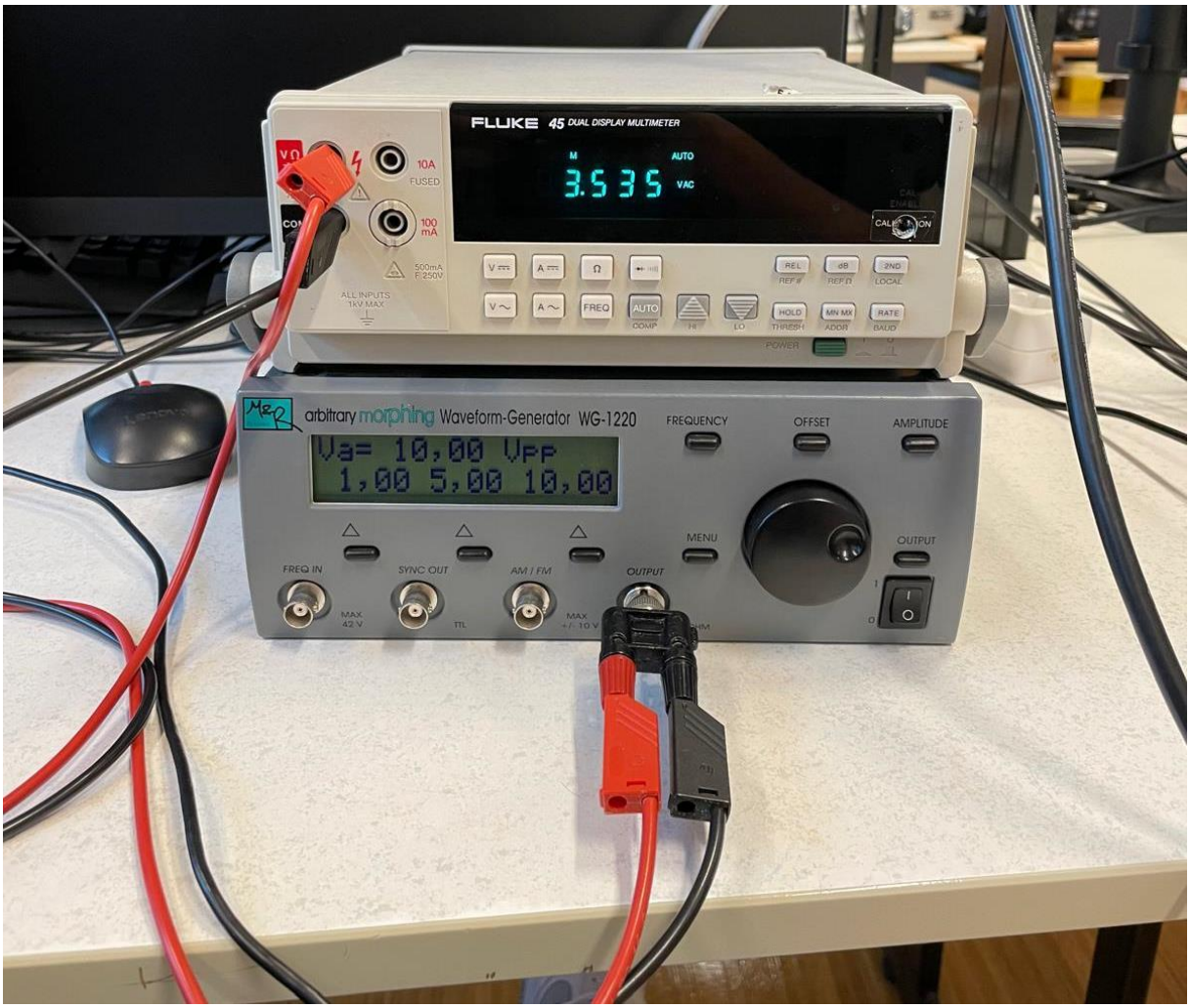


Abbildung 4: Aufbau

In Abbildung 4 wurde der Funktionsgenerator auf 10 V Spitze - Spitze eingestellt. Der Multimeter misst jedoch den Effektivwert dieses Sinussignals, welcher mit $U_{eff} = \frac{U_S}{\sqrt{2}}$ berechnet wird.

In Abbildung 2 und 3 werden die Werte in den Funktionsgenerator geschrieben und danach wird der vom Fluke 45 gemessene Spannungswert wieder ausgegeben.

30.09.: Werte über GUI einstellen und auslesen

Abbildung 5: Eingabe und Ausgabe über GUI

Mit dem Button „WG810 einstellen“ können die Werte für Amplitude, Frequenz und Kurvenform dem Funktionsgenerator übergeben werden und mit dem Button „einlesen“ kann der vom Fluke45 gemessene Wert wieder in der GUI ausgegeben werden.

Erweiterung Swing Worker(nicht fertiggestellt, nicht getestet)

```
private void onEinlesen(java.awt.event.ActionEvent evt)
  {
    new SwingWorker<Void, Object>()
    {
      @Override
      protected Void doInBackground()
        throws Exception
      {
        double messwert;
        try (Fluke45V2019 mg1 = new Fluke45V2019("COM15"))
        {
          messwert = mg1.getValue();

          final double frequenz = ((Number) tfFrequenz.getValue()).doubleValue();
          final double amplitude = ((Number) tfAmplitude.getValue()).doubleValue();

          String kurvenform = "SIN";

          if (rbDreieck.isSelected())
          {
            kurvenform = "TRI";
          }
          else if (rbRechteck.isSelected())
          {
            kurvenform = "REC";
```

```
          }
          SerialOutput serialoutput = new SerialOutput(kurvenform,
               amplitude, frequenz, messwert);
          serialOutputs.add(serialoutput);
        }
        catch (Exception e)
        {
          System.out.println(e.getMessage());
        }
        Catch (InterruptedException e)
        {
        }
        return null;
      }

      @Override
      protected void done()
      {
        listenfeld.setModel(new SerialListModel(serialOutputs));
      }
    }.execute();
  }
```

Damit die GUI während dem Einlesen benutzbar bleibt, soll die Methode onEinlesen mit einem SwingWorker als innere Klasse erweitert werden. Dazu werden die Einlese Vorgänge in die Methode doInBackground verlagert.

Resümee

Es wurde erfolgreich eine GUI (mit zugehörigem List Model und Datenhaltungsklasse) programmiert, um Daten vom Fluke45-Multimeter über die serielle Schnittstelle einzulesen und in der GUI auszugeben. Die Werte für den Funktionsgenerator konnten ebenfalls erfolgreich in der GUI eingestellt werden. Die zusätzlichen Funktionen der GUI, wie „Löschen“ und „Speichern“, wurden nicht implementiert, da diese nicht unbedingt erforderlich waren. Die nächste Aufgabe bestand darin unseren Code mit einem Swing Worker zu erweitern, jedoch konnte dies aufgrund von Zeitmangel nicht vollständig umgesetzt werden.

Außerdem fehlten uns immer wieder die nötigen Fähigkeiten im Java Programmieren, welche uns diese Aufgabenstellung schwer fallen lies.