JAVA MSP 2015

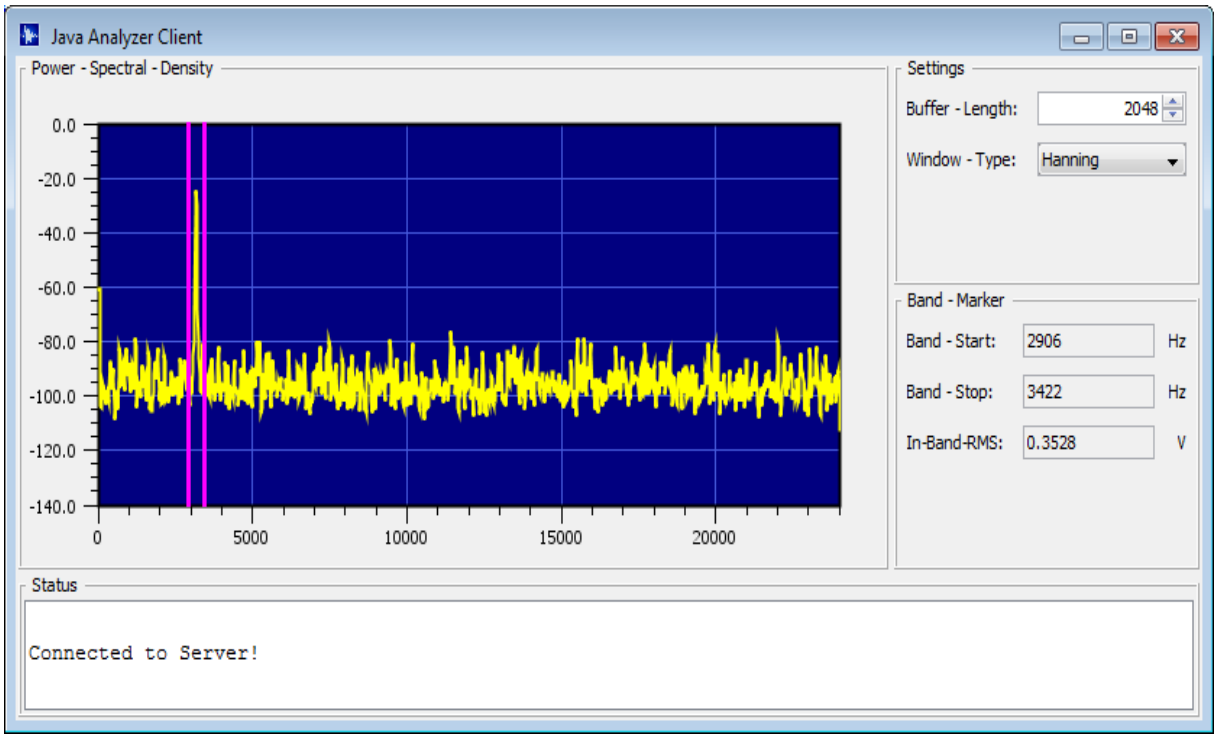
Bedingungen:

- Erlaubte Hilfsmittel: Unterrichtsunterlagen, Java Buch und Übungen.
- **Die Prüfung ist schrittweise, gemäss Aufgabenstellung lokal auf Ihrem Computer zu lösen. Kopieren sie zu diesem Zwecke den gesamten Ordner *MSP_SpectrumAnalyzer_FS2015_Vorlage* auf Ihre lokale Harddisk und importieren sie das Projekt in Eclipse. Am Ende der Prüfung ist der Ordner src umbenannt in *NameVorname* abzugeben.**
- **Setzen sie als erstes Ihren Namen und Vornamen in die Dateien.**
- **Gegenseitiges Abschreiben in irgendeiner Form führt zur Note 1!**
- **Folgend sie bei der Wahl von Variablen den Angaben in der Aufgabenstellung.**
- **Die Beilage muss abgegeben werden!**

Beschreibung:

Ziel dieser Prüfung ist es, eine einfach gehaltene Spektrum-Client-Applikation zu schreiben, die erlaubt, mit einer mittels TCP/IP zugänglichen Server-Hardware zur Datenerfassung in Verbindung zu treten und den Frequenzgehalt des gemessenen Signals zu analysieren. Die Server-Hardware wird zur Entwicklung der Client-Applikation mittels Software in Form einer Analyzer-Server-Applikation gemimt und muss zu diesem Zwecke vor allfälligen Tests aufgestartet werden.

Die Client-Applikation erlaubt grundlegende Einstellungen wie Länge der Fast-Fourier-Transformation (FFT) und Art der Fensterung der Daten (Windowing). Als Messresultat dargestellt wird, neben dem eigentlichen Spektrum, der resultierende Effektivwert des Signals im Spektralbereich zwischen zwei Bandmarkern.



Die Client-Applikation ist im klassischen Model-View-Controller Entwurfsmuster gehalten und verwendet für das Aufdatieren der Benutzerschnittstelle das Observable-Observer Entwurfsmuster. Das zugehörige Klassendiagramm und Layout ist im Anhang zu finden.

Der Aspekt View umfasst die Klassen *View, PNDisplay, PNSettings, PNMarker* und *PNStatus*. Das Panel *PNDisplay* bringt das resultierende Spektrum zur Darstellung und erlaubt mittels Maus die Bandmarker zu verschieben. Das *PNSettings* beheimatet einen *JSpinner* zur Wahl der Puffer-Länge sowie eine *JComboBox* zur Wahl der Fensterung. Das *PNMarker* dient rein der Darstellung der Bandmarker-Positionen sowie des resultierenden Effektivwertes im Band zwischen den Markern. Zu Debugg-Zwecken befindet sich unten in der Applikation ein Status - Fenster. Die Methoden zur Status-Anzeige sind statisch ausgeführt und können von überall her aufgerufen werden.

Der Controller-Aspekt des MVC-Musters in der Klasse *Controller* beschränkt sich darauf, die Aufgaben entsprechend zu delegieren.

Das *Model* ist Observable und erhält die Daten via das *DataListener* Interface von der zugehörigen Klasse *SignalSource*. Die Klasse *SignalSource* beheimatet die Socket-Verbindung zum Server. Da Client- und Server-Applikation auf demselben Rechner laufen, wird als IP - Adresse 127.0.0.1 verwendet. Auf der Socket-Verbindung aufgesetzt sind der *PrintWriter* zum Absetzen von Anfragen und Befehlen, sowie der *BufferedReader* zur Entgegennahme der Antworten. Um den GUI-Thread nicht zu blockieren, besitzt die Klasse *SignalSource* einen eigenen *Thread*. Das Aufdatieren der Benutzerschnittstelle geschieht via Observable - Observer Entwurfsmuster: Liegen neue Berechnungen vor, notifiziert das Model die View mittels der Schnittstelle Observer und leitet die Aufdatierung an die entsprechenden Panel weiter.

Einige Klassen sind bereits vorgegeben. Geschrieben, respektive ergänzt und anschliessend auch bewertet werden nur die Klassen ***View, PNSettings, PNMarker, Controller, Model*** und ***SignalSource.***

Folgen Sie beim Lösen der Prüfung der Aufgabenstellung und den Angaben im Anhang.

Achtung: Starten Sie unbedingt die Server-Applikation bevor Sie mit Tests beginnen.

Die Server-Applikation erlaubt die Steuerung der Blocklänge und die Abfrage des Puffers mit den Abtastwerten des Signals. Mittels der Benutzerschnittstelle lässt sich beim Server weiter der Spitzenwert, die Frequenz, die Form und Auflösung des Signales wählen. Die Applikation ist im Anhang dokumentiert.

Der Server ist so programmiert, dass er nach einem Verbindungsunterbruch eine neue Verbindung erlaubt und muss in aller Regel nicht immer wieder neue gestartet werden. Es ist jedoch nur möglich einen Client zu bedienen.

Aufgabe 1: Klassendiagramm (~3 Pte.)

Machen Sie sich als erstes mit dem Klassendiagramm und den vorhandenen Klassen bekannt. Sie kommen dadurch beim Lösen schneller voran.

a) Überprüfen Sie das Klassendiagramm und ergänzen Sie allenfalls fehlende Elemente im Code. Die Angaben im Klassendiagramm sind verbindlich.

Aufgabe 2: Benutzerschnittstelle (~53 Pte.)

Als nächstes programmieren resp. komplettieren wir das User - Interface der Applikation. Das User-Interface umfasst die Klassen *View, PNSettings* und *PNMarker***.** Das zugehörige Klassendiagramm und das Layout der Applikation sind im Anhang zu finden.

a) Implementieren Sie die Klassen gemäss Beschreibung im Code, soweit möglich und sinnvoll.

b) Starten Sie zur Kontrolle die Applikation.

Aufgabe 3: Klasse *Controller* (~5 gratis Pte.)

Der Controller delegiert die Aufgabe an die anderen Klassen.

a) Implementieren Sie den *Controller* gemäss Dokumentation im Code.

Aufgabe 4: Klasse *SignalSource* (~32 Pte.)

Der Klasse SignalSource kommt die Kommunikation mit dem Server zu. Sie gibt die Abtastwerte ans Model weiter.

a) Implementieren Sie die Klasse gemäss Dokumentation im Code.

Aufgabe 5: Klasse *Model* (~27 Pte.)

Dem Model in der Applikation kommt die Berechnung des Spektrums zu. Der zu gehörige Algorithmus ist im Anhang zu finden. Das Model ist via ***Observable*** - ***Observer*** Pattern mit der *View* verbunden.

a) Implementieren Sie die Klasse gemäss Dokumentation im Code.

b) Sie noch fehlenden Code, so noch Code fehlt …

Anhang:

Algorithmus zur Berechnung des Spektrum

```
public void process(double[] data)
```

Input:

`data`	Array mit den Abtastwerten des Signals

Output:

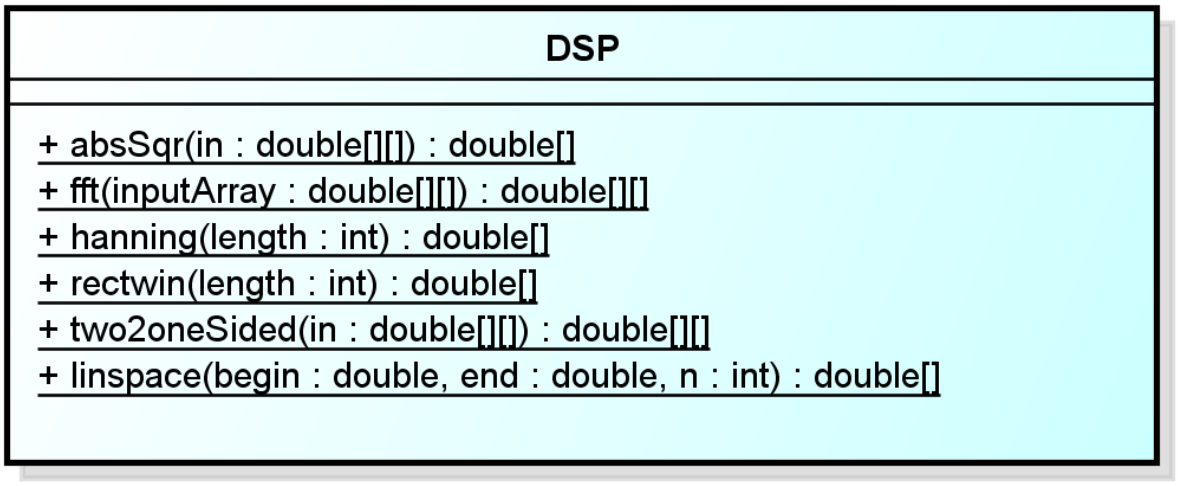
`powerSpectrum`	Das gemittelt Leistungsdichtespektrum via Attribut `powerSpectrum`.

Algorithmus:

Hinweis: Deklarieren Sie, wo notwendig, lokale Variablen.

1. Abtastwerte in `data` elementweise mit `window[i]` und `scale` multiplizieren und Resultat in lokalem Array `x[i][0] ablegen.`
2. Mittels `DSP.fft(x)` ***in-place*** die zugehörige FFT berechnen.
3. Mittels `DSP.two2oneSided(x)` das neue einseitige Spektrum berechnen.
4. Mittels `DSP.absSqr(XoneSided)` das neue, einseitige Leistungsdichtespektrum $S_{xx}[i]$ berechnen.
5. Mittelung:
   Für alle i von Null bis kleiner der Länge des neu berechneten Leistungsdichtespektrum
   - `powerSpectrum` an der Stelle i gleich `alpha` mal `powerSpectrum` an der Stelle i plus (1 minus `alpha`) mal das neue Leistungsdichtespektrum an der Stelle i setzen.

Zur Berechnung steht die Klasse DSP zur Verfügung:



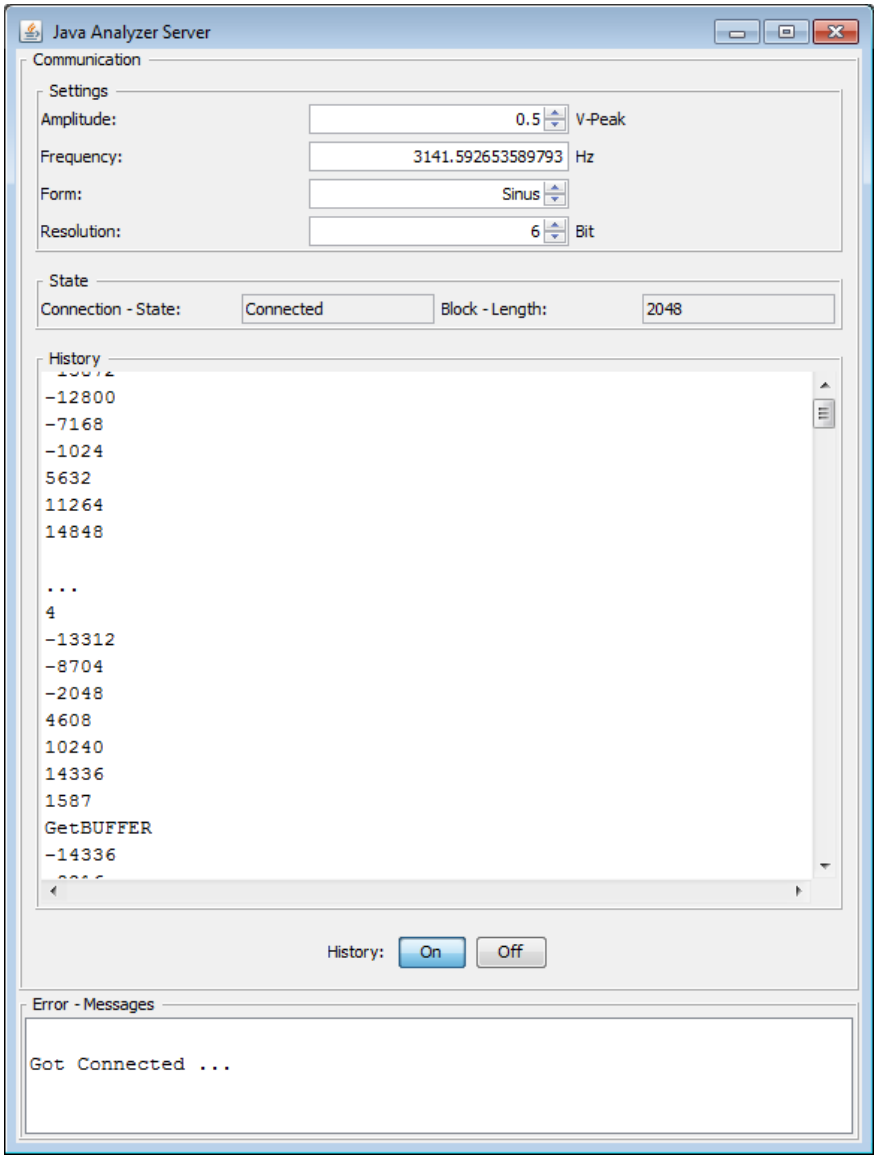
Analyzer-Server-Applikation

Die Server Applikation mimt die Hardware zur Datenaufzeichnung. Sie erlaubt eine TCP/IP - Verbindung via Local-Host (127.0.0.1), Port 11111 und nimmt den nachfolgenden Satz von Anfragen / Befehlen entgegen:

Mögliche Server - Anfragen / Befehle:

Das Kommunikationsprotokoll ist äusserst einfach gehalten und erlaubt folgende Anfragen / Befehle:

Anfrage / Befehl	Antwort / Aktion
`GetBUFFER`	Entsprechend gesetzter Block-Länge, werden Zeilen mit Double - Zahlen geliefert.
`SetBLOCKLENGTH` *Zahl*	Setzt die Block-Länge entsprechend ***Zahl.***



Die Server-Applikation erlaubt weiter die Amplitude, Frequenz, Wellenform und Auflösung der Abtastwerte zu wählen.