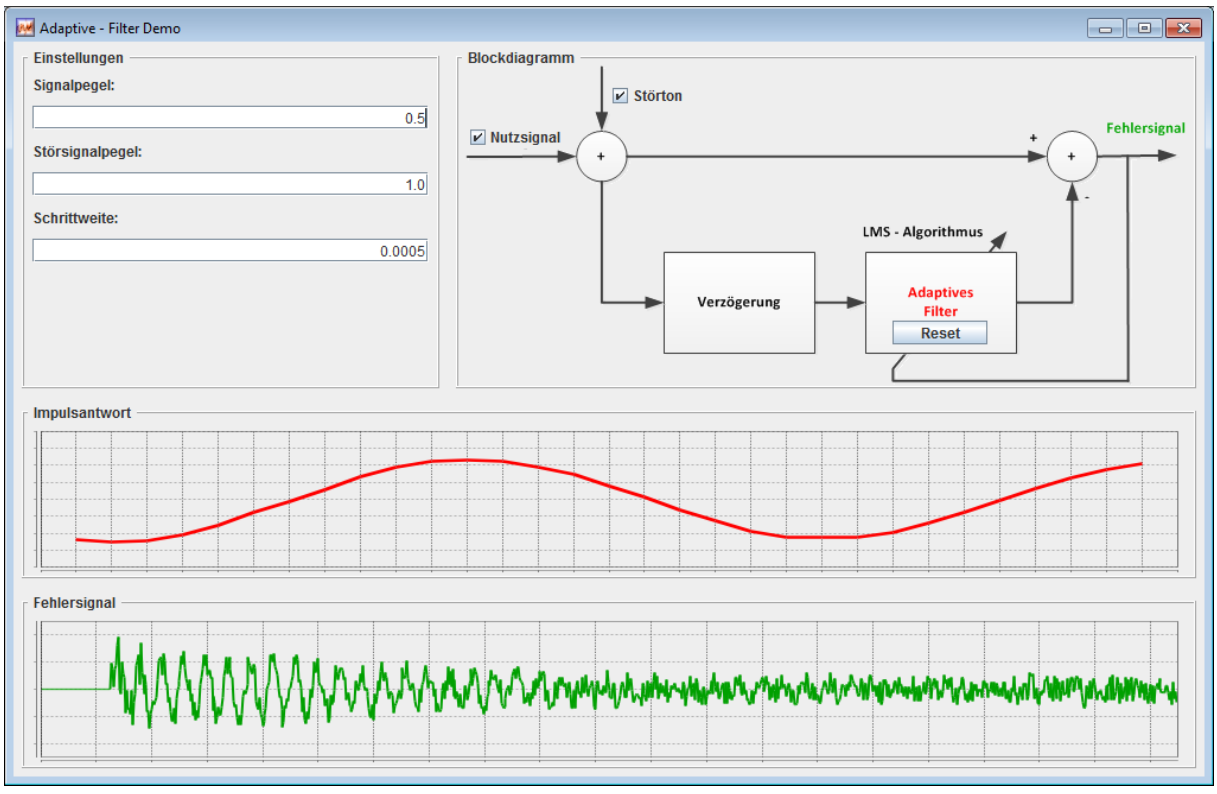
JAVA PRÜFUNG I

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 *P1_AdaptiveFilterDemo_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 exakt den Angaben in der Aufgabenstellung.**
- **Die Beilage mit dem Layout muss unterschrieben zurückgegeben werden!**

Beschreibung:

Ziel dieser Prüfung ist es, ein Demo-Programm zur Visualisierung adaptiver Filter zu programmieren. Die Demo zeigt, wie mit einem adaptiven Filter ein Störton entfernt werden kann. Adaptive Filter spielen in verscheiden Bereichen der Signalverarbeitung eine wichtige Rolle und kommen insbesondere auch in Modems vor.



Die Applikation ist im klassischen Model - View/Controller Entwurfsmuster programmiert. Die *View* ist in vier Panel gegliedert: Das *BildPanel* zeigt das zur Identifikation zugehörige Blockdiagramm und erlaubt die Signale ein- und auszuschalten sowie das adaptive Filter zurückzusetzten. Das *Parameter-Panel* beinhaltete drei Textfelder zur Definition der Parameter. *ImpulsPlotPanel* und *FehlerPlotPanel* dienen der Darstellung der Impulsantwort des adaptiven Filters und des Fehlersignals. Das *Model* hat eine *SignalQuelle*, eine Verzögerung und ein *LMSFilter*. Die *SignalQuelle* erzeugt das mit einem Störton gestörte Nutzsignal und erbt zu diesem Zwecke von der Klasse *Thread*. Sie ruft in regelmässigen Abständen via *SignalListener* die Methode *processSignal(signal : double[])* mit einem Block von digi-

talen Signalwerten auf. Die Signalwerte werden dann, zur Estimation des Störtons durch die Verzögerung und durch das *LMSFilter* geschickt und letzteres mittels Fehlersignal adaptiert. Der *Controller* delegiert im Wesentlichen die Aufgaben ans Model, welches wiederum via *Observer - Observable* Entwurfsmuster zum Aufdatieren der *View* führt.

Die Aufgabenstellung führt Sie schrittweise zum Ziel. Folgen Sie daher beim Lösen der Prüfung der Aufgabenstellung. Sämtliche Bilder sind im Ordner src/bilder zu finden und selbstsprechend bezeichnet.

Achten Sie darauf, dass Ihr Code kompilierbar bleibt!

Aufgabe 1: Aspekt View: Klassen des User - Interfaces (~ 52 Pte.)

View:

Die *View* ist als *GridBagLayout* organisiert und enthält die vier erwähnten Panels.

Das *BildPanel* hat den *Controller* und ist im Null-Layout organisiert. Es enthält das Blockdiagramm als Bild und zeichnet es in seiner natürlichen Grösse. Die bevorzugte Grösse des *BildPanel* wird mittels *setPreferredSize*() gleich der Grösse des Bildes gesetzt. Das *BildPanel* hat weiter zwei *JCheckboxen* und einen *JButton*. Jedes GUI-Element hat den *ActionListener* des *BildPanel*. Bei den entsprechenden Ereignissen werden die zugehörigen Methoden im *Controller* aufgerufen.

Das *ParameterPanel* hat den *Controller* und ist als *GridBagLayout* organisiert. Es enthält die drei Label und die zugehörigen Textfelder. Es verfügt über ein leeres *JLabel* um den überflüssigen Raum aufzunehmen. Jedes Textfeld hat den *ActionListener* des *ParameterPanel*. Bei den entsprechenden Ereignissen werden die zugehörigen Methoden im *Controller* aufgerufen. Die beiden Panels zur Darstellung der Impulsantworten und des Fehlersignals sind bereits implementiert und müssen nur entsprechend eingebettet werden.

a) Implementieren Sie die Konstruktoren der Klassen *View*, *BildPanel* und *ParameterPanel* gemäss Dokumentation und Beilage.

b) Komplettieren Sie die drei Klassen gemäss Dokumentation und Beilage.

Aufgabe 2: Aspekte Controller: *Klasse zu Steuerung* (~ 10 Pte.)

Controller: Der Controller delegiert die entsprechenden Aufgaben an das Model und umfasst neben dem Konstruktor vier Methoden, wobei deren drei den Aufruf lediglich weiterleiten. Einzig die Methode *setParameter()* liest erst die entsprechende Information aus den Textfeldern aus und ruft dann die zugehörige Methode des Models auf.

a) Implementieren Sie die Klasse *Controller* gemäss Dokumentation im Code.

Aufgabe 3: Aspekt Model: Klassen zur Berechnung (~ 58 Pte.)

Model: Das Model hat wie eingangs erläutert eine *SignalQuelle*, einen *Delay* und ein *LMSFilter* sowie einige Attribute und Methoden. Der Methode *processSignal(signal : double[])* kommt dabei die zentrale Bedeutung zu. Sie realisiert die im Blockdiagramm gezeigte Struktur.

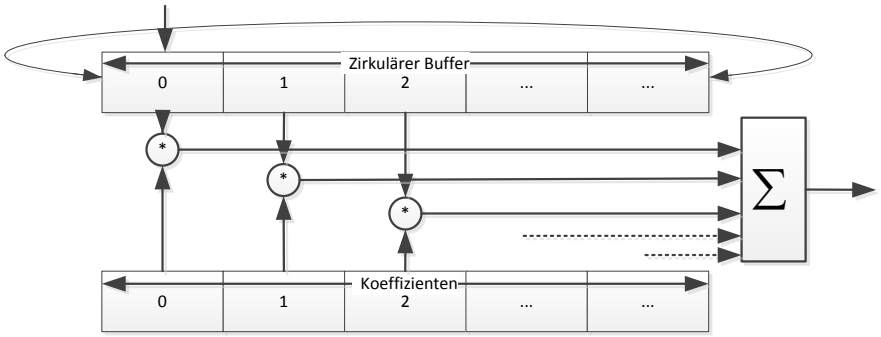
Die *SignalQuelle* hat einen Frame - Buffer, der jeweils via *SignalListener* an die Methode *processSignal(signal : double[])* weiter gegeben wird. Die entsprechenden Werte werden in der Methode *run()* in einer Endlos - Schleife generiert wobei der zugehörige *Thread* danach jeweils wieder mittels *sleep*() schlafen gelegt wird.

a) Implementieren Sie die Klasse *Signalquelle* gemäss Dokumentation im Code.

Das *FIRFilter* kapselt den Zustand und die Koeffizienten *coeffs* des Filters. Der Zustand ist als Zirkulärer-Buffer organisiert. Er enthält den momentanen und die vergangenen Signalwerte. Jeder neue Signalwert wird beim Aufruf der Methode *filter*() des *FIRFilters* in den Zirkulären-Buffer geschrieben und danach die Faltung berechnet. Die Faltung wird allgemein geschrieben als:

$$y[n] = \sum_{k=0}^{N} x[n-k] \cdot h[k]$$

wobei y[n] das Ausgangssignal zur Zeit n, x[n-k] den Eingangswert zur Zeit (n-k) und h[k] die Impulsantwort des Filters beschreibt. Die Berechnung der Faltung in der Methode *filter*() kann wie folgt skizziert werden:



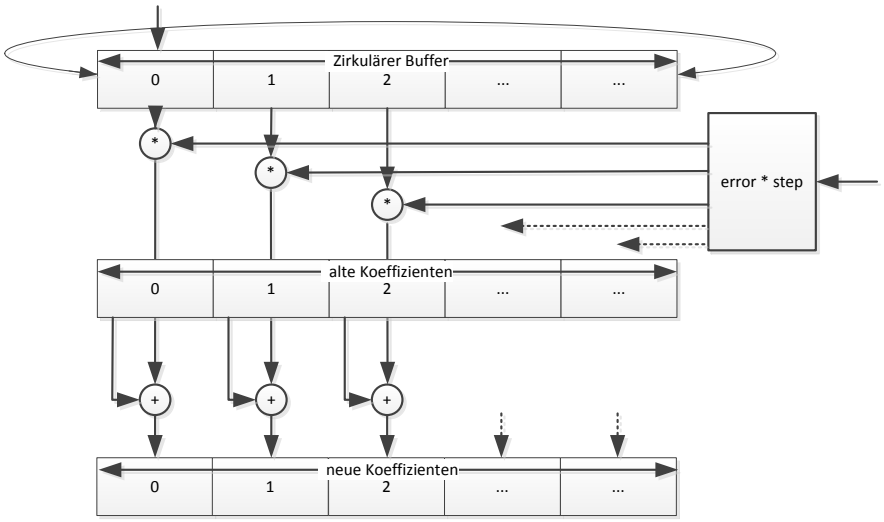
Die Berechnung für jeden Signalwert x[n] erfolgt dann im Model. Das heisst die Methode *filter*() berechnet nur die Summe über die Produkte zwischen Zirkulärem-Buffer und Koeffizienten.

b) Implementieren Sie die Klasse *FIRFilter* gemäss obiger Beschreibung und der Dokumentation im Code.

Das *LMSFilter* erweitert das *FIRFilter* um die Fähigkeit sich zu adaptieren. Dazu wird für jeden der Koeffizienten die Update - Beziehung, die vektoriell geschrieben wird, berechnet:

$$\vec{h}[n] = \vec{h}[n-1] + e[n] * \mu * \vec{x}[n]$$

Die Berechnung des Updates in der Methode *lms*() kann wie folgt skizziert werden:



c) Implementieren Sie die Klasse *LMSFilter* gemäss obiger Beschreibung und der Dokumentation im Code.

d) Komplettieren Sie die Klasse *Model* gemäss Dokumentation im Code.

Aufgabe 4: Clean Up (Bonus Pte.)

Gelegentlich geht etwas vergessen:

a) Ergänzen Sie den Code um noch allfällig fehlenden Code.

Korrigenda

Es müssen nur nachfolgende 8 Klassen modifiziert werden:

- **View**
- **BildPanel**
- **ParameterPanel**
- **Model**
- **SignalQuelle**
- **FIRFilter**
- **LMSFilter**
- **Controller**

BildPanel:

- Belassen Sie die Elemente an der Stelle wie in der Aufgabenstellung gegeben!

ParameterPanel:

- Default - Werte (wie in Aufgabenstellung) in die Textfelder schreiben.

SignalQuelle:

- stoerer mit stoerPegel multiplizieren: