OOP2

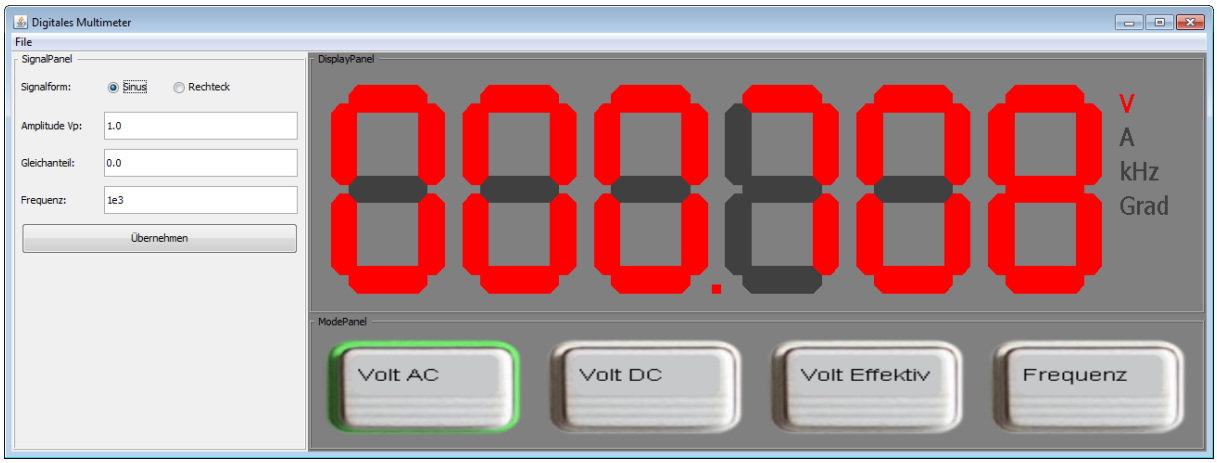
PRÜFUNG II

Bedingungen:

- Erlaubte Hilfsmittel: Eclipse, Unterrichtsunterlagen, Übungen und Java Buch.
- Die Prüfung ist schrittweise, gemäss Aufgabenstellung lokal auf Ihrem Computer zu lösen. Kopieren Sie zu diesem Zwecke den gesamten Ordner auf Ihre lokale Harddisk und importieren Sie das Projekt in Eclipse. Am Ende der Prüfung ist der Ordner src, umbenannt in **NameVorname**, auf dem Memory-Stick 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, ein funktionsfähiges Multimeter zu erstellen, mit dem die Spannung (AC/DC/Eff.wert) sowie die Frequenz eines Signals bestimmt werden können. Zur Entwicklung des Multimeters wird als Signalquelle eine sogenannte Mock-Klasse verwendet, die später durch ein entsprechendes Messsystem ersetzt würde.



Das Multimeter zeigt als User-Interface das klassische Design mit Tasten und digitaler Anzeige. Auf der linken Seite ist ein separates Panel zur Einstellung der Signalquelle vorhanden. Die *View* ist dazu in die drei Panel *SignalPanel*, *DisplayPanel* und *ModePanel* gegliedert. Alle drei Panels basieren auf einem *JPanel* und sind entsprechend der Beilage organisiert.
Die Klasse *View* ist als *GridBagLayout* organisiert und beheimatet die drei erwähnten Panel. Das *DisplayPanel* hat zur Anzeigte der Amplituden und der Frequenz ein *DigitDisplay*, bei dem sich mittels rechter Maustaste die Farbe wählen lässt. Das *ModePanel* hat vier Tasten zur Wahl des Messmodus. Das *SignalPanel* erlaubt die Wahl der Signalform, der Spitzenwertamplitude, des Gleichanteils und der Frequenz. Die Werte werden mittels Taste *Übernehmen* gesetzt. Die Applikation verfügt über ein Menu *File* mit den Item *Logging* und *Exit* und erlaubt die angezeigten Messwerte in eine Log-Datei zu schreiben resp. das Programm zu beenden.
Wie aus dem UML – Klassendiagramm in der Beilage ersichtlich, ist die Applikation nach dem klassischen MVC – Muster programmiert: Entsprechend dem ausgelösten Ereignis, wird seitens des GUI die zugehörige Methode des Controllers aufgerufen. Der Controller enthält die Logik der Applikation und bewirkt entsprechend die Änderung im Model. Das Model macht die Berechnungen und löst via Observer - Interface der *View* das Update des User–Interfaces aus. Die Klasse *Model* implementiert einen

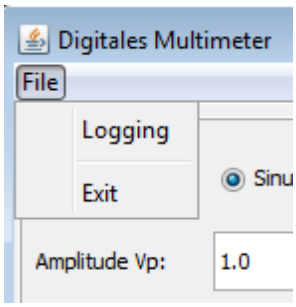
ChannelListener und wird mittels der Klasse *SignalQuelle* in regelmässigen Abständen mit Signalwerten bei einer Abtastrate von 48 kHz versorgt.
Die nachfolgende Aufgabenstellung führt sie schrittweise zum Ziel. Halten sie sich an die Vorgaben bezüglich Methoden- und Variabelnamen. Im Lauf der Prüfung sind nur die Klassen *View*, *DisplayPanel*, *SignalPanel*, *ModePanel*, *Controller*, *Model* und *MenuBar* auszuprogrammieren. Änderungen in den anderen Klassen werden bei der Korrektur nicht berücksichtigt.

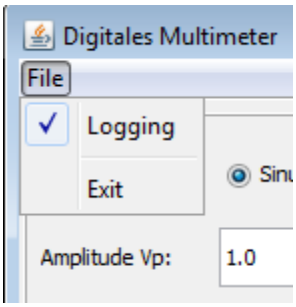
Aufgabe 1: Aspekt View: Klassen des User - Interfaces (~ 67 Pte.)

Die View ist wie beschrieben aufgebaut und erlaubt mittels *SignalPanel* die Steuerung der Signalquelle und mittels *ModePanel* die Wahl der Messart. Das *DisplayPanel* bringt den Messwert mittels Siebensegmentanzeigen zur Darstellung.

a) Implementieren Sie die Klassen View, *SignalPanel*, *DisplayPanel* und *ModePanel* gemäss Dokumentation und Beilage. Die dazu benötigten Klassen *DigitPanel* und *BasicRadioButton* sind bereits implementiert.

b) Die *MenuBar* erlaubt einen Data-Logger zu aktivieren und die Applikation mittels *Exit* zu beenden. Implementieren Sie die Klassen gemäss Dokumentation und Pseudo-Code **OHNE** den Teil ***Challenge***.





Aufgabe 2: Aspekt Controller: *Klasse zu Steuerung* (~ 9 Pte.)

Der *Controller* leitet allfällige Aufgaben an das *Model* weiter. Die allenfalls benötigten Informationen werden vom *Controller* via *View* ausgelesen. Um das Update alle 333ms auszulösen, verfügt der Controller weiter über einen Swing-Timer.

a) Implementieren Sie die Klasse *Controller* gemäss Dokumentation und Pseudo-Code **OHNE** den Teil ***Challenge***.

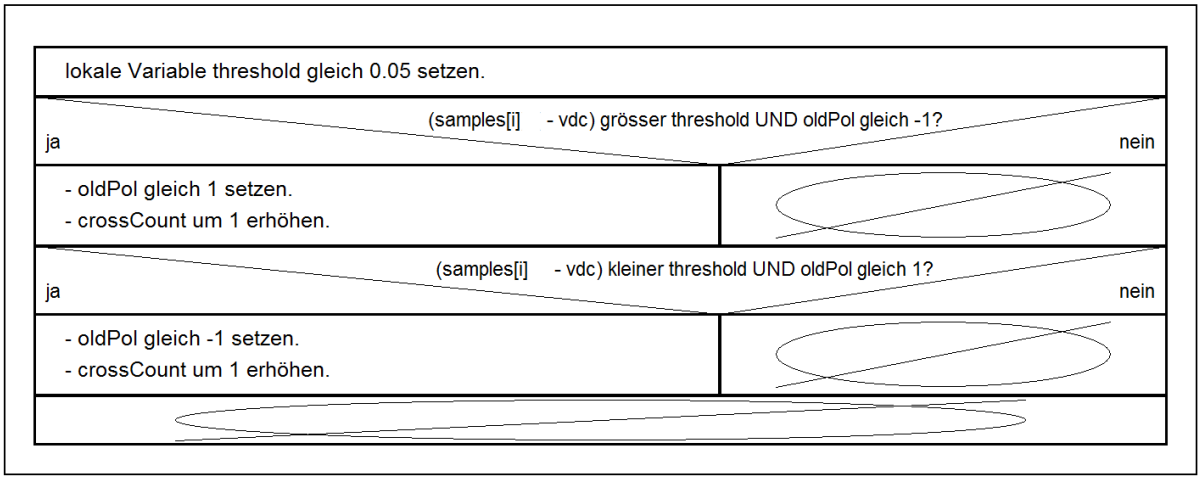
Aufgabe 3: Aspekt Model: Klassen zur Berechnung (~ 27 Pte.)

Das *Model* hat, wie eingangs erwähnt, die *SignalQuelle*. Die Berechnung der Messgrössen geschieht vollumfänglich in der Methode process() des Models, die in regelmässigen Abständen je 2048 Signalwerte bekommt. Die nachfolgend angegebenen Berechnungen müssen in einer for – Schleife für jeden der 2048 Signalwerte ausgeführt werden:

$$V_{dc} = \alpha \cdot V_{dc} + (1-\alpha) \cdot s[i]$$
$$P_V = \alpha \cdot P_V + (1-\alpha) \cdot s^2[i]$$

wobei $s[i]$ gleich samples[i] ist.

Zur Berechnung der Frequenz muss innerhalb der for – Schleife die Anzahl Nulldurchgänge des Signals gezählt werden:



Die restlichen Werte lassen sich nun ausserhalb der for – Schleife berechnen:

$$f_{req} = \alpha_{freq} \cdot f_{req} + (1-\alpha_{freq}) \cdot 0.5 \cdot \frac{c_{rossCount} \cdot f_{sampling}}{2048}$$
$$V_{ac} = \sqrt{P_V - V_{dc}^2}$$
$$V_{eff} = \sqrt{P_V}$$

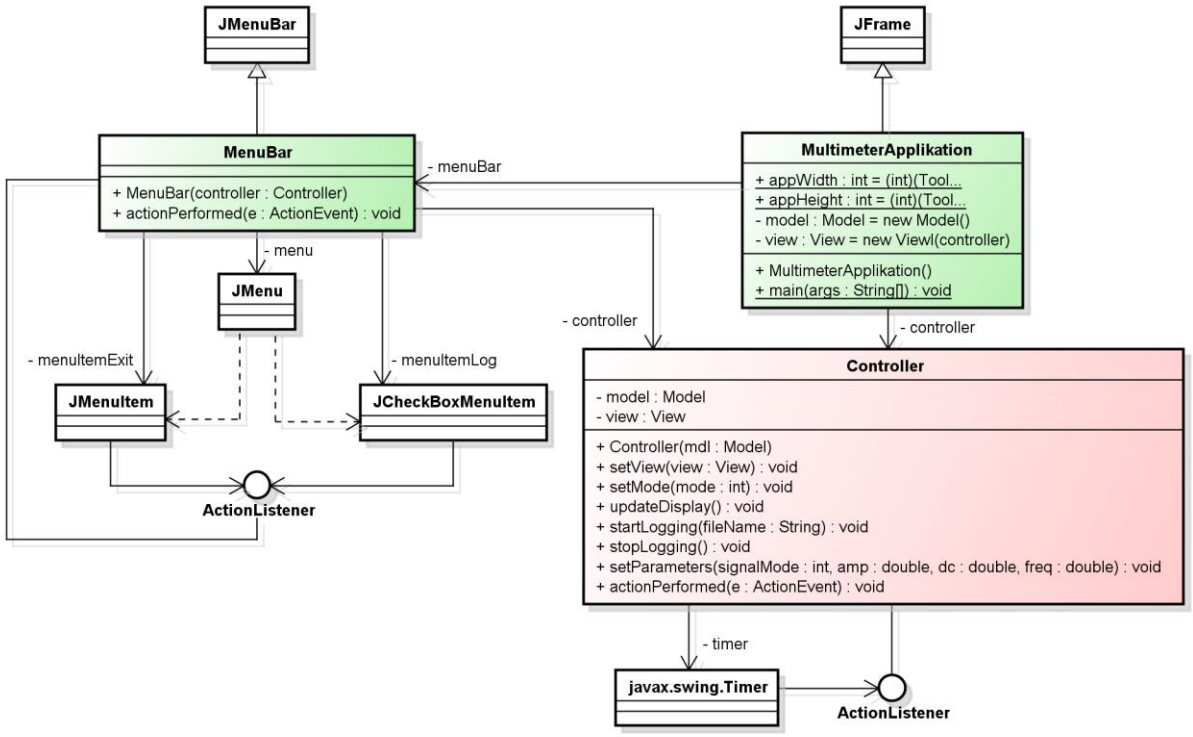
d.h. einmal pro Methodenaufruf durch die Signalquelle.

a) Implementieren Sie die Klasse gemäss Dokumentation im Code.
b) Testen Sie ihr Programm gegen Ihr AET-Wissen. Welche Werte müssen sich einstellen?

Aufgabe 4: Challenge League: (~ 21 Pte)

Die *MenuBar* erlaubt unter dem Menu *File* einerseits via *Exit* das Programm zu schliessen, andererseits mittels *JCheckBoxMenuItem* einen *JFileChooser* zum Festlegen einer Datei zu öffnen und damit das Protokollieren der Anzeigewerte zu starten. Wird das *JCheckBoxMenuItem* wieder deaktiviert, gilt es die Protokollierung wieder zu stoppen.

a) Implementieren Sie die Klasse innerhalb der vorgegebenen Struktur.
b) Testen Sie ihr Programm.



Aufgabe 5: Clean Up (~ ? Pte)

Gelegentlich geht etwas vergessen …

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