JAVA PRÜFUNG I

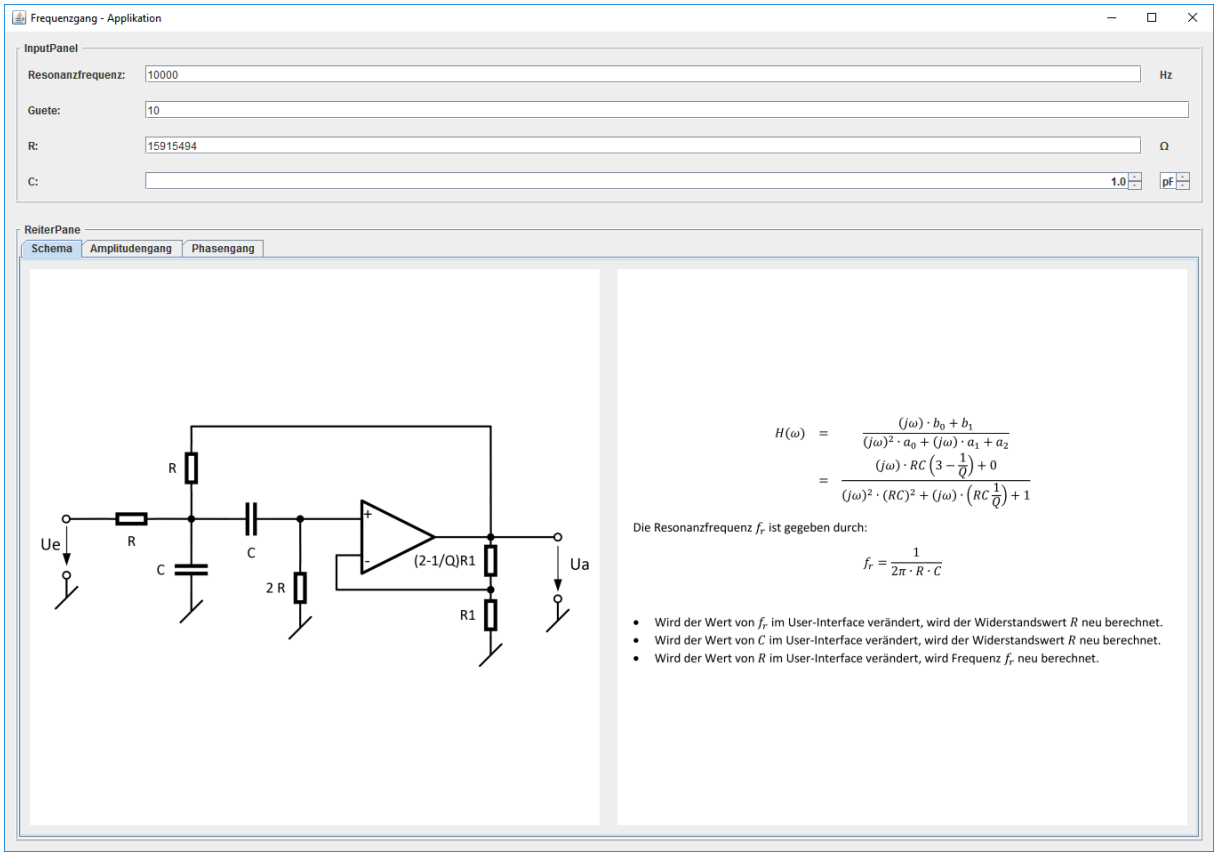
Bedingungen:

- Erlaubte Hilfsmittel: Unterrichtsunterlagen, Java Buch, alte Übungen und Prüfungen sowie Beispiele aus dem Internet.
- **Die Prüfung ist schrittweise, gemäss Aufgabenstellung, lokal auf Ihrem Computer zu lösen. Kopieren sie zu diesem Zwecke den gesamten Ordner *P1_AktivRCBandpass_FS_2019_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 und den Modulanlass den Sie besuchen, 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!**

Top-Down-Beschreibung:

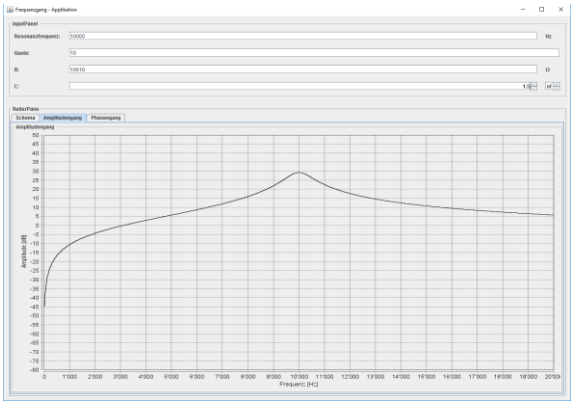
Ziel dieser Prüfung ist es, eine Applikation zur Dimensionierung eines Aktiv-RC-Bandpasses zu programmieren. Aktiv-RC-Filter erlauben, mit Hilfe eines Operationsverstärkers, Filter mit konjugiert-komplexen Polen zu realisieren. Ein Bandpass-Filter lässt das erwünschte Band passieren und sperrt unerwünschte Frequenzkomponenten.

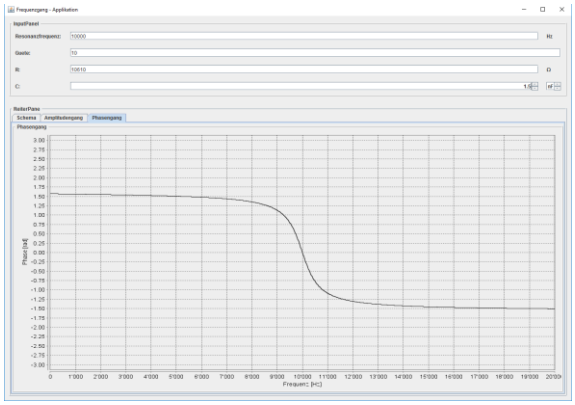
Die Applikation erlaubt die Komponenten R und C zu berechnen und den zugehörigen Amplituden- und Phasengang zu visualisieren:



Die gesamte Applikation ist im *Model – View/Controller* und *Observer-Observable* Entwurfsmuster gehalten. Die *View* beheimatet ein *InputPanel* und ein *ReiterPane*. Das *InputPanel* erlaubt die entsprechenden Werte zu spezifizieren. Die Resonanzfrequenz, die Güte und der Widerstandswert lassen sich frei festlegen. Die Kapazität wird mittels Spinner aus der E12-Reihe gewählt.

Das *ReiterPane* hat drei Reiter: Der Reiter *Schema* zeigt das Schema des Aktiv-RC-Bandpasses sowie die zugehörigen Gleichungen, je auf einem *BildPanel*. Die Reiter *Amplitudengang* und *Phasengang* zeigen den entsprechenden Plot.





Das *InputPanel* steuert den *Controller* und der *Controller* löst via Methode *aktion*() die entsprechende Berechnung im *Model* aus. Das *Model* beheimatet die Berechnungen und verwendet zur Berechnung allenfalls die Klasse *PicoMatlab*. Das Model ist *Observable* und notifiziert nach der Berechnung via die Schnittstelle *Observer* die *View*. Die *View* wiederum leitet das *Update* an die entsprechenden Panels weiter. In den *update*()-Methode werden allenfalls die Daten aus dem Model geholt.

Die Aufgabenstellung führt Sie Schrittweise zum Ziel!

Achten Sie darauf, dass Ihr Code kompilierbar bleibt!

Aufgabe 1: Abfolge beim Aufstarten des Codes (~7 Pte.)

Beim Aufstarten der Applikation werden die benötigten Objekte erzeugt und allenfalls mit einander verbunden. Die Applikation soll beim Aufstarten nachfolgende Spur hinterlassen.

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

✓	**Start via AktivRCBandpassApplikation.main(String args[])**
✓	**Attribute von AktivRCBandpassApplikation@1757115189 werden initialisiert ...**
✓	**Attribute von Model@700108102 werden initialisiert ...**
✓	**Konstruktor Model():Model@700108102 wird ausgeführt ...**
✓	**Attribute von Controller@724389817 werden initialisiert ...**
✓	**Konstruktor Controller():Controller@724389817 wird ausgeführt ...**
✓	**Attribute von View@559583620 werden initialisiert ...**
	Attribute von ReiterPane@1229278551 werden initialisiert ...
	Attribute von AmplitudenPlot@1144857746 werden initialisiert ...
	Konstruktor AmplitudenPlot():AmplitudenPlot@1144857746 wird ausgeführt ...
	Attribute von PhasenPlot@1592919222 werden initialisiert ...
	Konstruktor PhasenPlot():PhasenPlot@1592919222 wird ausgeführt ...
	Attribute von SchemaPanel@42471095 werden initialisiert ...
	Attribute von BildPanel@1142908149 werden initialisiert ...
	Konstruktor BildPanel():BildPanel@1142908149 wird ausgeführt ...
	Attribute von BildPanel@255647422 werden initialisiert ...
	Konstruktor BildPanel():BildPanel@255647422 wird ausgeführt ...
	Konstruktor SchemaPanel():SchemaPanel@42471095 wird ausgeführt ...

	Konstruktor ReiterPane():ReiterPane@1229278551 wird ausgeführt ...
✓	**Konstruktor View():View@559583620 wird ausgeführt ...**
	Attribute von InputPanel@1284771892 werden initialisiert ...
	Konstruktor InputPanel():InputPanel@1284771892 wird ausgeführt ...
✓	**Konstruktor AktivRCBandpassApplikation():AktivRCBandpassApplikation@1757115189 wird ausgeführt..**
✓	**Methode Controller@724389817.setView() wird ausgeführt ...**
✓	**Objekt View@559583620 wird als Observer von Objekt Model@700108102 registriert!**

✓ Diese Zeilen erscheinen bereits zu Beginn in der Spur!

Aufgabe 2: Klasse *View* (~5 Pte.)

Als nächstes widmen wir uns der Klasse *View*. Sie ist der Top-Level-Container und beinhaltet das *ReiterPane* und das *InputPanel*. Die *View* ist *Observer* des *Models* und gibt einen Aufruf der Methode *update()* geeignet weiter.

a) Implementieren Sie die Klasse View gemäss Dokumentation im Code.

Aufgabe 3: Klasse *InputPanel* (~33 Pte.)

Das *InputPanel* hat die Vielzahl der User-Interface-Elemente und erhält ein *update()*, so die Werte im Model sich geändert haben.

a) Implementieren Sie den Konstruktor entsprechend den Angaben in der Aufgabenstellung.

b) Programmieren Sie die Methode *update()* entsprechend den Angaben im Code.

c) Programmieren Sie die Methode *expandReihe()* gemäss Pseudo-Code.

Aufgabe 4: Klasse *ReiterPane* (~6 Pte.)

Das *ReiterPane* beheimatet auf seinen Reitern das *SchemaPanel* sowie den Amplituden- und den Phasenplot.

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

Aufgabe 5: Klasse *SchemaPanel* (~2 Pte.)

Das *SchemaPanel* umfasst lediglich ein *BildPanel* mit dem Schema und ein *BildPanel* mit der Formel.

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

Aufgabe 6: Klasse *Controller* (~14 Pte.)

Der Controller hat im Wesentlichen die Methode *aktion()*. Sie wird aufgerufen, wenn neue Werte im Model gesetzt und eine neue Berechnung ausgeführt werden soll. In diesem Teil ist Ihr Können gefragt! Die Beschreibung ist bewusst als Prosa-Text gehalten.

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

Aufgabe 7: Klasse *Model* (~11 Pte.)

Dem Model kommen die Berechnungen, unter Zuhilfenahme der Klasse *PicoMatlab,* zu. Die entsprechenden Beziehungen sind hier der Bequemlichkeit halber nochmals wiedergegeben:

$$H(\omega) = \frac{(j\omega)\cdot b_0 + b_1}{(j\omega)^2\cdot a_0 + (j\omega)\cdot a_1 + a_2}$$

$$= \frac{(j\omega)\cdot RC\left(3-\frac{1}{Q}\right)+0}{(j\omega)^2\cdot (RC)^2 + (j\omega)\cdot\left(RC\frac{1}{Q}\right)+1}$$

Die Resonanzfrequenz f_r ist gegeben durch:

$$f_r = \frac{1}{2\pi \cdot R \cdot C}$$

- Wird der Wert von f_r im User-Interface verändert, wird der Widerstandswert R neu berechnet.
- Wird der Wert von C im User-Interface verändert, wird der Widerstandswert R neu berechnet.
- Wird der Wert von R im User-Interface verändert, wird Frequenz f_r neu berechnet.

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

Aufgabe 8: Super-League (~10 Pte.)

Zeigen Sie ihr Können!

a) Ergänzen Sie den Value-Spinner um einen anonymen *MouseWheelListener*, so dass man durch die Werte Scrollen kann. Hinweis: *getNextValue*(), *getPreviousValue*(), *setValue*().

b) Ergänzen Sie den Code um noch allfällig fehlende Zeilen.