Praktikumsaufgaben 1:
Komplexe Zahlen und Schwingkreis

A r b e i t s g r u p p e	
OPP-Gruppe (1,2, oder 3)	OPP-E3b /
Name, Vorname	
Name, Vorname	
Name, Vorname	
Bericht erstellt von	

B e r i c h t s a b n a h m e					
Anmerkungen					
Bewertung	Datum:	Nachbessern		Abgenommen	
Bewertung	Datum:	Nachbessern		Abgenommen	
	Unterschrift:				

Lernziele:

- Erzeugen einer Klasse mit verschiedenen Konstruktoren
- Schreiben von Methoden
- Schreiben von Klassenmethoden
- Kennenlernen des *Math*-Paketes
- Kennenlernen der Methoden *toString* und *equals*
- Automatische Erzeugung von Methoden in *Eclipse*
- Korrekte Dokumentation mittels *JavaDoc*
- Kapselung bei Referenzen als Parameter und Rückgabewert (Aufgabe 1.2)

1 Allgemeine Hinweise

Vorbereitung:

- Bitte bereiten Sie die nachfolgenden Aufgaben soweit wie möglich Zuhause vor.
- Um den größten Lerneffekt zu erzielen, wird dringend empfohlen, dass zunächst jeder von Ihnen alleine versucht, die Aufgaben zu lösen. (Tauschen Sie sich bei Problemen mit Ihrer Gruppe oder mit anderen Kommilitonen aus.)
- Erzeugen Sie vor dem Praktikumstermin ein Java-Projekt, an dem Ihre Gruppe während des Praktikums weiterarbeitet.
- Während des Praktikums sind die Lösungen von Ihrer Dreiergruppe gegebenenfalls zu vervollständigen und zur Abnahme zu präsentieren.

Erfolgreiche Teilnahme / Abnahme:

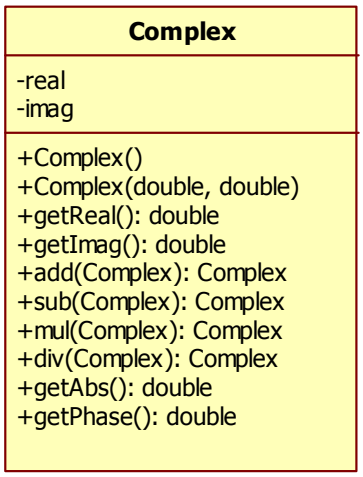
- Die erfolgreiche Abnahme aller Praktikumstermine sowie eines Berichts pro Teilnehmer sind Voraussetzungen zur Teilnahme an der Praktikumsprüfung.
- Die Abnahme erfolgt individuell und nicht pauschal für jede Dreiergruppe.
- Es besteht Anwesenheitspflicht bei allen Praktikumsterminen.
- Der Code muss den in der Vorlesung aufgestellten Programmierrichtlinien entsprechen.
- Mündliche Abnahme:
  - Vorführung der vollständigen Lösungen
  - Befragung zu den Lösungen sowie verwandten theoretischen Themen
- Abgabe des Codes in elektronischer Form in das hierfür vorgesehene Netzlaufwerk.

Beachten: Verwenden Sie ausschließlich die in der Aufgabenstellung angegebenen Bezeichner und Namen (z.B. für Pakete, Klassen, Variablen und Methoden).

2 Komplexe Zahlen (Aufgabe 1.1)

In dieser Aufgabe ist eine Klasse *Complex* zur Repräsentation von komplexen Zahlen zu erstellen. Diese Klasse benötigen Sie unter anderem, um elektrische Kapazitäten und Impedanzen zu beschreiben und Netzwerke aus diesen Bauelementen automatisch zu berechnen.

Die Klasse *Complex* wird grundlegend durch das nachfolgende UML-Symbol beschrieben. Weitere Methoden ergeben sich durch die Aufgabenbeschreibung.



2.1 Benötigte Formeln

Für komplexe Zahlen $z = a + bj$ gilt mit $j^2 = -1$ und dem konjugiert Komplexen $\bar{z} = a - bj$:

- Addition / Subtraktion: $z_1 \pm z_2 = (a_1 \pm a_2) + (b_1 \pm b_2) \cdot j$
- Multiplikation: $z_1 z_2 = (a_1 a_2 - b_1 b_2) + (a_1 b_2 + b_1 a_2) \cdot j$
- Division: $\frac{z_1}{z_2} = \frac{z_1}{z_2} \cdot \frac{\overline{z_2}}{\overline{z_2}} = \frac{z_1 \overline{z_2}}{\|z_2\|^2} = \frac{a_1 a_2 + b_1 b_2}{a_2^2 + b_2^2} + \frac{a_2 b_1 - a_1 b_2}{a_2^2 + b_2^2} \cdot j$
- Betrag: $\|z\| = \sqrt{a^2 + b^2}$

2.2 Eclipse / Projektstruktur

a) Erstellen Sie in Eclipse (i) einen Workspace, hierin (ii) ein Java-Projekt *Praktikum* sowie in diesem Projekt (iii) das Paket *aufgabe1_1*.

2.3 Attribute und Konstruktoren

Requirement 1: Die Klasse besitzt Variablen (Attribute) *real* und *imag* zur Speicherung des Real- und Imaginärteils. Beide sind vom Typ *double* und *private*.

Requirement 2: Es gibt einen Konstruktor ohne Parameter (Standardkonstruktor), der die beiden Attribute mit dem Wert 0.0 vorbelegt.

Requirement 3: Es gibt einen Konstruktor mit zwei *double*-Parametern, der die Attribute auf die Parameterwerte setzt.

Deklarationen:

- `public Complex()`
- `public Complex(double real, double imag)`

2.4 Getter und arithmetische Operatoren

Requirement 4: Es gibt Getter-Methoden für die Attribute *real* und *imag*. Diese geben den aktuellen Wert von *real* bzw. *imag* zurück.

Requirement 5: Es gibt Methoden zur Addition, Subtraktion, Multiplikation und Division zweier komplexer Zahlen. Diesen wird jeweils ein Parameter *z* vom Typ *Complex* übergeben. Der erste Operand der jeweiligen Operation ist das Objekt, für das die Methode aufgerufen wird. Der zweite Operand ist die beim Aufruf übergebene Zahl *z*. Das Ergebnis wird im Objekt, für das die Methode aufgerufen wird, gespeichert. Der Rückgabewert ist eine Referenz auf dieses Objekt.

Deklarationen:

- `public double getReal()`
- `public double getImag()`
- `public Complex add(Complex z)`
- `public Complex sub(Complex z)`
- `public Complex mul(Complex z)`
- `public Complex div(Complex z)`

2.5 Betrag und Phase

Requirement 6: Es gibt eine Methode zur Berechnung des Betrages der komplexen Zahl.

Requirement 7: Es gibt eine Methode zur Berechnung der Phase (Winkel) der komplexen Zahl. Die Phase ist im Bereich $[0, 2\pi)$ zurückzugeben. Für $z = 0 + 0j$ wird die Phase als $\angle z = 0$ definiert.

Hinweise zur Phase:

- Prinzipiell berechnet sich die Phase über $\angle z = \text{atan}(b/a)$, wobei allerdings je nach Quadrant die Zahl π addiert oder subtrahiert werden muss.

- Betrachten Sie die Methoden *Math.atan* und *Math.atan2*. Worin unterscheiden sich die beiden Methoden? Welche ist für diese Phasenberechnung geeigneter?

Deklarationen:

- `public double getAbs()`
- `public double getPhase()`

2.6 Klassenmethoden für arithmetische Operationen

Requirement 8: Es gibt Klassenmethoden (Modifier „static“) zur Addition, Subtraktion, Multiplikation und Division zweier komplexer Zahlen. Diesen werden beide Operanden als Parameter vom Typ *Complex* übergeben. Das Ergebnis der Operation wird in einer neu erzeugten komplexen Zahl gespeichert. Der Rückgabewert ist eine Referenz auf dieses neue Objekt.

Deklarationen:

- `public static Complex add(Complex z1, Complex z2)`
- `public static Complex sub(Complex z1, Complex z2)`
- `public static Complex mul(Complex z1, Complex z2)`
- `public static Complex div(Complex z1, Complex z2)`

2.7 Methoden *toString* und *equals*

Requirement 9: Die Klasse besitzt eine Methode *toString*, die einen String in folgendem Format zurückgibt:
`(<Realteil> + <Imaginärteil>*j)` beziehungsweise
`(<Realteil> – <Imaginärteil>*j)`

Requirement 10: Die Klasse besitzt eine Methode *equals*. Die Methode gibt den Wert *true* zurück, wenn das als Parameter übergebene Objekt eine komplexe Zahl ist, die die gleichen Real- und Imaginärteile hat wie das Objekt, für das *equals* aufgerufen wird.

Hinweis zu *equals*:

- Sie können die Methode in *Eclipse* über das Menü *Source / Generate …* automatisch erzeugen lassen.
- Entfernen Sie anschließend alle unnötigen Kommentarzeilen und mit erzeugten Methoden.

Deklarationen:

- `public String toString()`
- `public boolean equals(Object obj)`

2.8 Beispiel

Requirement 11: Erstellen Sie eine ausführbare Klasse *ExampleComplex*, mit der Sie (in der *main*-Methode) alle Methoden der Klasse *Complex* an einigen Zahlenbeispielen testen.

2.9 Dokumentation

Requirement 12: Dokumentieren Sie die Klasse *Complex* mittels *JavaDoc*. Es müssen mindestens eine Kurzbeschreibung, die Autoren (*@author*) sowie die Version (*@version*) enthalten sein.

Requirement 13: Dokumentieren Sie sämtliche Methoden der Klasse *Complex* mittels *JavaDoc*. Es müssen mindestens eine Kurzbeschreibung sowie – falls vorhanden – die Parameter (*@param*) und die Rückgabe (*@return*) enthalten sein.

Requirement 14: Erzeugen Sie die Dokumentation im HTML-Format. (Diese ist nicht auszudrucken!)

3 Anwendung (Aufgabe 1.2)

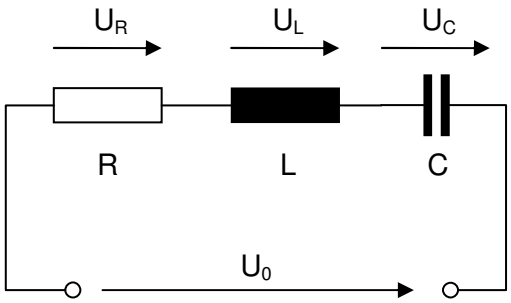
Erstellen Sie unter Verwendung der Klasse *Complex* ein Programm, das für einen Reihenschwingkreis die Spannung am Widerstand berechnet und in einem Frequenzbereich ausgibt.

$$Z_R = R$$

$$Z_L = j\omega L$$

$$Z_C = \frac{1}{j\omega C}$$

$$\underline{Z} = R + j\omega L + \frac{1}{j\omega C}$$



$$\underline{U_R} = R * \frac{U_0}{\underline{Z}}$$

Requirement 1: Erstellen Sie eine ausführbare Klasse *ResonantCircuit*, mit der Sie die Spannung am Widerstand ausgeben. Die Ausgabe erfolgt durch die Klassenmethode *main*.

Requirement 2: Die Kennwerte für den Widerstand R (10 Ω), die Induktivität L (10 mH) und die Kapazität C (10 nF) sollen als Konstante in der Klasse angelegt werden.

Requirement 3: Erstellen Sie eine statische Methode *voltageOverResistor*, die die komplexe Wechselspannung über den Widerstand berechnet. Die Methode erhält die Kreisfrequenz als Parameter. Die Eingangsspannung soll mit 1 V angenommen werden.

Requirement 4: Die Ausgabe der Spannung erfolgt im Bereich von 10 kHz bis 100 kHz in Schritten von 10 kHz.

Requirement 5: Dokumentieren Sie die Klasse mittels *JavaDoc*. Es müssen mindestens eine Kurzbeschreibung, die Autoren (*@author*) sowie die Version (*@version*) enthalten sein.

Requirement 6: Dokumentieren Sie sämtliche Methoden der Klasse *ResonantCircuit* mittels *JavaDoc*. Es müssen mindestens eine Kurzbeschreibung sowie – falls vorhanden – die Parameter (*@param*) und die Rückgabe (*@return*) enthalten sein.

Design-Vorgabe: Es soll zunächst ein Array mit den Frequenzen erstellt werden. Daraus wird dann ein Array mit den Kreisfrequenzen erstellt. Diese Array wird dann abgearbeitet und die zugehörigen Spannungen bestimmt und die Spannungen werden wieder in einem Array abgespeichert. In einem dritten Schritt erfolgt erst die Ausgabe der Frequenz und der Spannung (Prinzip „Trennung der Verantwortlichkeiten“). Fassen Sie ggf. alle Arrays in einem Array zusammen, was damit eine Tabelle mit den einzelnen Arrays als „Spalten“ bildet.

4 Zusatzfragen

Beantworten Sie folgende Fragen in Ihrem Bericht so präzise wie möglich in maximal 5 Sätzen!

1. Was bedeutet „Information Hiding“ im Zusammenhang mit objektorientierter Programmierung und was ist der Vorteil davon?

2. Was bedeutet bzw. welche Auswirkungen hat das Schlüsselwort „static“ für Variablen und Methoden?