hochschule mannheim

PR2 – Programmierung 2 FAKULTÄT FÜR INFORMATIK
Cyber Security Bachelor (CSB), Medizinische Informatik Bachelor (IMB)
Sommersemester 2020 [24.03.2020]

PR2 Aufgabenblatt für Testate Nr. 1

Allgemeine Hinweise

Ausgabe der Übung:	Di, 24.03.2020, 16:50 Uhr
Abgabe:	Mo, 20.04.2020, 16:00 Uhr
Testat am:	Di, 21.04.2020 (bei Präsenzbetrieb)

- Die Abnahme der Übungen gilt als **Prüfungsleistung**. Bei einer Verhinderung durch Krankheit ist eine ärztliche Bescheinigung der Arbeitsunfähigkeit vorzulegen.
- Laden Sie Ihre Lösungen bis zur Deadline in Moodle bei der entsprechenden Unteraufgabe hoch. Quellcodes müssen einheitlich und sinnvoll formatiert sein (vorzugsweise mithilfe Ihrer IDE wie Eclipse oder IntelliJ). **Ausarbeitungen in einem anderen Format werden nicht berücksichtigt.**
- Während der Abnahme sind die Ergebnisse am Rechner live zu demonstrieren.
- Bei der Abnahme der Übung ist der Studentenausweis vorzulegen.

Lernziele

- Objektorientierte Programmierung in Java anwenden können
- Ausnahmebehandlung in Java umsetzen können und auf Problemstellungen anwenden können
- Innere Klassen benutzen zu können

Aufgabe 1
Komplexe Zahlen als abstrakter Datentyp

Entwickeln Sie eine Klasse namens `KomplexeZahl` zur Repräsentation komplexer Zahlen. Komplexe Zahlen bestehen aus zwei Anteilen: einem Realteil und einem Imaginäranteil:

$$C = R * R = \{(x, y) | x, y \in R\}$$

Die Klasse soll das Prinzip *infomration hiding* umsetzen. Die Selektoren („Getter“) sollen `re()` und `im()` heißen. Die Klasse soll keine Mutatoren („Setter“) haben. Die Klasse soll genau einen Konstruktor haben. Die Klasse soll im Package `pr2.pu1` liegen.

Aufgabe 2

Textuelle Repräsentation komplexer Zahlen (`toString()`)

Implementieren Sie die Methode `toString()` für `KomplexeZahl`. Sie soll als Ergebnis einen `String` liefern, der Objekte dieser Klasse in der algebraischen Form darstellt (s. z. B. `https://de.wikibooks.org/wiki/Komplexe_Zahlen/_Darstellungsformen`).

Aufgabe 3

Addition zweier komplexer Zahlen

Schreiben Sie eine Methode namens `add` für die Addition zweier komplexer Zahlen. Die Methode soll an einem Objekt vom Typ `KomplexeZahl` aufgerufen werden können. Dabei soll ein zweites Objekt vom Typ `KomplexeZahl` als Parameter übergeben werden. Das Ergebnis soll eine neue Instanz von `KomplexeZahl` sein, bei der gilt:

$$(x1, y1) + (x2, y2) = (x1 + x2, y1 + y2)$$

Aufgabe 4

Multiplikation zweier komplexer Zahlen

Schreiben Sie eine Methode namens `prod` für die Multiplikation zweier komplexer Zahlen. Die Methode soll an einem Objekt vom Typ `KomplexeZahl` aufgerufen werden können. Dabei soll ein zweites Objekt vom Typ `KomplexeZahl` als Parameter übergeben werden. Das Ergebnis soll eine neue Instanz von `KomplexeZahl` sein, bei der gilt:

$$(x1, y1) * (x2, y2) = (x1 * x2 - y1 * y2, x1 * y2 + y1 * x2)$$

Aufgabe 5

`main`-Methode

Schreiben Sie eine `main`-Methode für die Klasse `KomplexeZahl`, so dass die Klasse als Programm ablaufen kann. Zuerst soll eine Instanz von `KomplexeZahl` erzeugt und der lokalen Variable `x` zugewiesen werden, die als reellen Anteil 2,5 beinhaltet und als imaginären Anteil 4,5. Diese komplexe Zahl soll auf der Konsole ausgegeben werden in der Form

```
x = 2.5 + 4.5i
```

Aufgabe 6

Statische Methoden

Schreiben Sie zusätzlich zu `add` und `prod` zwei statische Methoden („Klassenmethoden“), die dieselbe Funktionalität anbieten.

Aufgabe 7
Reelle Zahlen als Unterklasse von `KomplexeZahl`

Verwenden Sie `KomplexeZahl` und lassen Sie die neue Klasse `ReelleZahl` davon erben, die dafür verwendet werden soll, reelle Zahlen zu repräsentieren. Die neue Klasse soll auch im Package `pr2.pu1` sein.

Es soll also alles so sein wie bei `KomplexeZahl`, außer dass es keinen imaginären Anteil geben soll (ist immer auf 0.0 gesetzt).

Die Klasse darf nur einen Konstruktor haben. Er soll den Konstruktor der Oberklasse benutzen. Es sollen keine weiteren Mutatoren („Setter“) oder Selektoren („Getter“) implementiert werden.

Die textuelle Repräsentation (`toString()`) soll ausschließlich den reellen Anteil zurückliefern.

Aufgabe 8
Ausnahmebehandlung

Für einen speziellen Anwendungszweck sollen komplexe Zahlen ausgechlossen werden, bei denen der imaginäre Anteil größer als 10 ist. Entwickeln Sie die neue Klasse `KomplexeZahlKlein1` im Package `pr2.pu1`.

Sie soll von `KomplexeZahl` erben und im Konstruktor prüfen, ob der imaginäre Anteil größer als 10 ist. Wenn das der Fall ist, soll eine `Exception` erzeugt und geworfen werden. Sie soll die Nachricht beinhalten, dass die Zahl für den imaginären Anteil zu groß ist und welchen Wert der reelle Anteil hat.

Aufgabe 9
Eigene Ausnahme als innere Klasse

Die neu zu entwickelnde Klasse `KomplexeZahlKlein2` im Package `pr2.pu1` soll dieselbe Funktion haben wie `KomplexeZahlKlein1`, aber statt mit `Exception` mit einer selbst definierten Ausnahme namens `KZKException` in Form einer nicht statischen inneren Member-Klasse arbeiten.

Bei dieser Klasse muss die Methode `getMessage()`, die einen `String` liefert, überschrieben werden, so dass im Fehlerfall dieselbe Fehlermeldung wie bei `KomplexeZahlKlein1` erscheint.

Aufgabe 10
Live-Testat

Wird erst beim Testieren bekannt gegeben. Es wird um JUnit-Tests gehen.