Karlsruher Institut für Technologie

Prof. Dr. Ralf H. Reussner
Software Design and Quality (SDQ)
https://sdqweb.ipd.kit.edu/wiki/Programmieren
programmieren-vorlesung@ipd.kit.edu

Programmieren – Sommersemester 2019

Abschlussaufgabe 1
Version 1.3

20 Punkte

Ausgabe:	15.07.2019, ca. 13:00 Uhr
Praktomat:	29.07.2019, 13:00 Uhr
Abgabefrist:	13.08.2019, 06:00 Uhr

Abgabehinweise

Bitte beachten Sie, dass das erfolgreiche Bestehen der öffentlichen Tests für eine erfolgreiche Abgabe dieses Blattes notwendig ist. Der Praktomat wird Ihre Abgabe zurückweisen, falls eine der nachfolgenden Regeln verletzt ist. Eine zurückgewiesene Abgabe wird automatisch mit null Punkten bewertet. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

- Achten Sie auf fehlerfrei kompilierenden Programmcode.
- Verwenden Sie keine Klassen der Java-Bibliotheken, ausgenommen Klassen der Pakete `java.lang`, `java util`, `java.util.regex` und `java.util.function`.
- Achten Sie darauf, nicht zu lange Zeilen, Methoden und Dateien zu erstellen. Sie müssen bei Ihren Lösungen eine maximale Zeilenbreite von 120 Zeichen einhalten.
- Halten Sie alle Whitespace-Regeln ein.
- Halten Sie alle Regeln zu Variablen-, Methoden- und Paketbenennung ein.
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klassen, Methoden und Attribute.
- Nutzen Sie nicht das `default`-Package.
- `System.exit()` und `Runtime.exit()` dürfen nicht verwendet werden.
- Halten Sie die Regeln zur Javadoc-Dokumentation ein.
- Halten sie auch alle anderen Checkstyle-Regeln an.

Checkstyle

Der Praktomat überprüft Ihre Quelltexte während der Abgabe automatisiert auf die Einhaltung obenstehender Regeln. Falls eine Regel entsprechend markiert ist, wird der Praktomat die Abgabe zurückweisen, wenn die Regel nicht eingehalten wird. Andere Regelverletzungen können zu Punktabzug führen. Sie können und sollten Ihre Quelltexte bereits während der Entwicklung auf die Regeleinhaltung überprüfen. Das Programmieren-Wiki (ILIAS) beschreibt, wie das funktioniert.

Terminal-Klasse

Laden Sie für diese Aufgabe die Terminal-Klasse herunter und platzieren Sie diese unbedingt im Paket `edu.kit.informatik`. Die Methode `Terminal.readLine()` liest eine Benutzereingabe von der Konsole und ersetzt `System.in`. Die Methode `Terminal.printLine()` schreibt eine Ausgabe auf die Konsole und ersetzt `System.out`. Verwenden Sie für jegliche Konsoleneingabe oder Konsolenausgabe die Terminal-Klasse. Verwenden Sie in keinem Fall `System.in` oder `System.out`. Fehlermeldungen werden ausschließlich über die Terminal-Klasse ausgegeben und müssen aus technischen Gründen unbedingt mit `Error,` beginnen. Laden Sie die Terminal-Klasse **niemals** zusammen mit Ihrer Abgabe hoch.

Plagiarismus

Wichtig: Das Einreichen fremder Lösungen, seien es auch teilweise Lösungen von Dritten, aus Büchern, dem Internet oder anderen Quellen wie beispielsweise der Lehrveranstaltung oder dem Übungsbetrieb selbst, ist ein Täuschungsversuch und führt zur Bewertung nicht bestanden. Dies beinhaltet unter anderem auch die Lösungsvorschläge des Übungsbetriebes. Es werden nur rein selbstständig erarbeitete Lösungen akzeptiert. Für weitere Ausführungen sei auf die Einverständniserklärung (Disclaimer) verwiesen

Bearbeitungshinweise

Diese Bearbeitungshinweise sind relevant für die Bewertung Ihrer Abgabe, jedoch wird der Praktomat Ihre Abgabe **nicht** zurückweisen, falls eine der nachfolgenden Regeln verletzt ist.

- Achten Sie darauf, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung als auch Funktionalität bewertet werden. Halten Sie die Hinweise zur Modellierung im ILIAS-Wiki ein.
- Programmcode muss in englischer Sprache verfasst sein.

- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich.
- Die Kommentare sollen einheitlich in englischer oder deutscher Sprache verfasst werden.
- Wählen Sie aussagekräftige Namen für alle Ihre Bezeichner.

Abgabemodalitäten

Die Praktomat-Abgabe wird am **Montag, den 29.07.2019 um 13:00 Uhr**, freigeschaltet. Achten Sie unbedingt darauf, Ihre Dateien im Praktomaten bei der richtigen Aufgabe vor Ablauf der Abgabefrist hochzuladen.

- Geben Sie Ihre Klassen zu Aufgabe A als `*.java`-Dateien ab.

A. Kommandozeilenbasiertes Rechenhilfsmittel für komplexe Zahlen (20 Punkte)

In dieser Aufgabe soll ein kommandozeilenbasiertes Rechenhilfsmittel für die vier Grundrechenarten Addition, Subtraktion, Multiplikation und Division für komplexe Zahlen implementiert werden. Zusätzlich soll das Rechenhilfsmittel zur Laufzeit das Speichern und Rechnen von komplexen Zahlen in Variablen unterstützen.

Die komplexen Zahlen $\mathbb{C}$ erweitern den Zahlenbereich der reellen Zahlen $\mathbb{R}$ derart, dass die Gleichung $x^2 + 1 = 0$ lösbar wird. Dies gelingt durch Einführung einer neuen imaginären Zahl i mit der Eigenschaft $\mathrm{i}^2 = -1$. Diese Zahl i wird als imaginäre Einheit bezeichnet. Komplexe Zahlen können in der Form $(a + b\,\mathrm{i})$ dargestellt werden, wobei a und b in diesem Rechenhilfsmittel ganze Zahlen sind und i die imaginäre Einheit ist. Zur Vereinfachung wird $a, b \in \mathbb{Z}_{INT}$ mit $\mathbb{Z}_{INT} := \left\{x \in \mathbb{Z} \mid x \geq -2^{31} \wedge x \leq 2^{31} - 1\right\}$ angenommen. D.h. Sie müssen keine Unter- und Überläufe behandeln.

Auf die so dargestellten komplexen Zahlen lassen sich die üblichen Rechenregeln für ganze Zahlen anwenden, wobei i^2 stets durch -1 ersetzt werden kann und umgekehrt. Zur Vereinfachung müssen auch hier keine Unter- und Überläufe geprüft oder behandelt werden.

A.1. Grundrechenregeln

A.1.1. Addition +

Für die Addition zweier komplexer Zahlen $z_1 = a + b\,\mathrm{i}$ mit $a, b \in \mathbb{Z}_{INT}$ und $z_2 = c + d\,\mathrm{i}$ mit $c, d \in \mathbb{Z}_{INT}$ gilt:

$$z_1 + z_2 = (a + c) + (b + d)\,\mathrm{i}$$

A.1.2. Subtraktion -

Für die Subtraktion zweier komplexer Zahlen $z_1 = a + b\,\mathrm{i}$ mit $a, b \in \mathbb{Z}_{INT}$ und $z_2 = c + d\,\mathrm{i}$ mit $c, d \in \mathbb{Z}_{INT}$ gilt:

$$z_1 - z_2 = (a - c) + (b - d)\,\mathrm{i}$$

A.1.3. Multiplikation *

Für die Multiplikation zweier komplexer Zahlen $z_1 = a + b\,\mathrm{i}$ mit $a, b \in \mathbb{Z}_{INT}$ und $z_2 = c + d\,\mathrm{i}$ mit $c, d \in \mathbb{Z}_{INT}$ gilt:

$$z_1 * z_2 = (ac - bd) + (ad + bc)\,\mathrm{i}$$

A.1.4. Division /

Für die Division zweier komplexer Zahlen $z_1 = a + b\,\mathrm{i}$ mit $a, b \in \mathbb{Z}_{INT}$ und $z_2 = c + d\,\mathrm{i}$ mit $z_2 \neq 0$ und $c, d \in \mathbb{Z}_{INT}$ gilt:

$$z_1 / z_2 = \frac{ac + bd}{c^2 + d^2} + \frac{bc - ad}{c^2 + d^2}\,\mathrm{i}$$

Die Division in $\mathbb{Z}_{INT}$ wird ~~auf~~ gegen 0 gerundet. D.h. der Quotient für die Operanden n und d, mit $n, d \in \mathbb{Z}_{INT}$, ist ein ganzzahliger Wert $q \in \mathbb{Z}_{INT}$, dessen Größe so groß wie möglich ist, während er $|d * q| \leq |n|$ erfüllt. Außerdem ist q positiv, wenn $|n| \geq |d|$ und n und d das gleiche Vorzeichen haben, aber q ist negativ, wenn $|n| \geq |d|$ und n und d entgegengesetzte Vorzeichen haben.

B. Zuweisung =

Mit einer Variable lässt sich eine komplexe Zahl speichern, die vom Rechenhilfsmittel gelesen und geschrieben werden kann. Um eine Variable zu nutzen, muss ihr eine komplexe Zahl zugewiesen werden. Der Zuweisungsoperator = ist ein binärer Operator, bei dem auf der linken Seite die zu belegende Variable steht und auf der rechten Seite ein Ausdruck. Erst nach dem Auswerten des Ausdrucks kopiert der Zuweisungsoperator das Ergebnis in die Variable. Gibt es Laufzeitfehler, etwa durch eine Division durch null, gibt es keinen Schreibzugriff auf die Variable. Wie in Java, können Variablen beliebig oft neue komplexe Zahlen zugewiesen werden.

B.1. Operatorrangfolge

Die Operatorrangfolge entspricht den gebräuchlichen mathematischen Konventionen: Klammern zuerst, Punkt vor Strich und von links nach rechts.

Für die Operatorrangfolge der vier Grundrechenarten gilt die Konvention Punktrechnung vor Strichrechnung. D.h. in einem Ausdruck haben, sofern keine Klammern gesetzt sind, Multiplikationen und Divisionen immer Vorrang vor Additionen und Subtraktionen.

Eine Klammerung bietet die Möglichkeit ein Teilstück einer Kette von notierten Operationen bevorzugt auszuwerten. Der eingeklammerte, also von einem eckigen Klammerpaar [...] eingeschlossene Bereich ist rechnerisch zuerst auszuführen und durch das entsprechende Teilergebnis zu ersetzen, was die Klammerung obsolet macht, da sie nun keine Operatoren mehr enthält. Die Klammerung muss die Operatoren samt ihren nötigen Operanden enthalten. Entgegen der verbreiteten Konvention werden Ausdrücke nicht mit runden Klammern, sondern mit eckigen Klammern geklammert, um diese deutlich von der Repräsentation der komplexen Zahlen abzugrenzen.

Für die Operatorrangfolge der vier Grundrechenarten gilt die Konvention links nach rechts. D.h. wenn Operatoren gleicher Priorität im gleichen Ausdruck erscheinen, werden diese von links nach rechts rechnerisch zuerst ausgeführt. Die Zuweisungsoperatoren werden jedoch von rechts nach links ausgewertet.

Die Operatoren in Table 1 sind in der Reihenfolge ihrer Priorität aufgeführt. Je näher am oberen Rand der Tabelle ein Operator erscheint, desto höher ist seine Priorität. Operatoren mit höherer Priorität werden vor Operatoren mit niedrigerer Priorität ausgewertet. Operatoren auf der gleichen Ebene haben die gleiche Priorität.

Priorität	Operator	Beschreibung	Assoziativität
3	`[ ]`	Klammern	links nach rechts
2	`* /`	Multiplikation Division (Punktrechnung)	links nach rechts
1	`+ -`	Addition Subtraktion (Strichrechnung)	links nach rechts
0	`=`	Zuweisung	rechts nach links

Tabelle 1: Operatorrangfolge

B.2. Grammatik

Gültige Ausdrücke zur Eingabe in das Rechenhilfsmittel werden im Folgendem als kontextfreie Grammatik beschrieben. Eine kontextfreie Grammatik ist eine formale Grammatik, die nur solche einfachen Produktionsregeln enthält, bei welchen immer genau ein Nichtterminalsymbol auf eine beliebig lange Folge von Nichtterminal- und/oder Terminalsymbolen abgeleitet wird. Die Produktionsregeln haben folglich immer die Form $V \rightarrow w$, wobei V ein Nichtterminalsymbol ist und w eine Folge mit Alternativen bestehend aus Nichtterminal- und/oder Terminalsymbolen. Zeichenfolgen, die zwischen den spitzen Klammern < > eingeschlossen sind, stellen die Nichtterminalsymbole dar. Das Zeichen → (Pfeil) wird zur Definition verwendet und das Zeichen | (vertikaler Strich) hat die Bedeutung einer logischen Disjunktion (∨) zur Definition von Alternativen.

In den Produktionsregeln stellen die Zeichenfolgen, die zwischen den einfachen Hochkommata `''` eingeschlossen sind, Terminalsymbole dar. Dazu werden für die kontextfreie Grammatik noch die beiden Terminalsymbole ***Integer*** und ***Variable*** definiert. Das Terminalsymbole ***Integer*** entspricht hierbei der Zeichenfolge, die von der `parseInt`-Methode fehlerfrei in den primitiven Datentyp `int` übersetzt werden kann.

Das Terminalsymbole ***Variable*** entspricht hierbei einer Zeichenfolge, die ein gültiger Java-Variablenname ist. Mit Ausnahme der Zeichenkette `quit`.

Listing 1: Grammatik zum Programmverhalten

```
1 <Zuweisung> → <Variable> '=' <Ausdruck>
2 <Ausdruck> → <Literal> | <Addition> | <Subtraktion> | <
      Multiplikation> | <Division> | <Klammer>
3 <Literal> → <Variable> | <Komplex>
4 <Komplex> → '('<Integer> '+' <Integer> 'i' ')'
5 <Addition> → <Ausdruck> '+' <Ausdruck>
6 <Subtraktion> → <Ausdruck> '-' <Ausdruck>
7 <Multiplikation> → <Ausdruck> '*' <Ausdruck>
8 <Division> → <Ausdruck> '/' <Ausdruck>
9 <Klammer> → '[' <Ausdruck> ']'
```

C. Interaktive Benutzerschnittstelle

Nach dem Start nimmt Ihr Programm über die Konsole mittels `Terminal.readLine()` Eingaben entgegen, die im Folgenden näher spezifiziert werden. Nach Abarbeitung einer

Eingabe wartet Ihr Programm auf weitere Eingaben, bis das Programm irgendwann durch die Eingabe der Zeichenfolge `quit` beendet wird.

Achten Sie darauf, dass durch die Ausführung der folgenden Befehle die zuvor definierte Grammatik (siehe Listing 1) nicht verletzt wird und geben Sie in diesen Fällen immer eine aussagekräftige Fehlermeldung beginnend mit `Error ,␣` aus. Entspricht eine Eingabe nicht dem vorgegebenen Format, dann ist immer eine Fehlermeldung auszugeben. Danach soll das Programm auf die nächste Eingabe warten. Bei Fehlermeldungen dürfen Sie den Text frei wählen, er sollte jedoch sinnvoll sein. Jede Fehlermeldung muss aber mit `Error ,␣` beginnen und darf keine Zeilenumbrüche enthalten.

Da wir automatische Tests Ihrer interaktiven Benutzerschnittstelle durchführen, müssen die Ausgaben exakt den Vorgaben entsprechen. Insbesondere sollen sowohl Klein- und Großbuchstaben als auch die Leerzeichen und Zeilenumbrüche genau übereinstimmen. Geben Sie auch keine zusätzlichen Informationen aus. Beginnen Sie frühzeitig mit dem Einreichen, um Ihre Lösung dahingehend zu testen, und verwenden Sie das Forum, um eventuelle Unklarheiten zu klären.

Beachten Sie, dass bei der Beschreibung der Eingabe- und Ausgabeformate die Wörter zwischen spitzen Klammen (< und >) für Platzhalter stehen, die bei der konkreten Ein- und Ausgabe durch Werte ersetzt werden. Diese eigentlichen Werte enthalten bei der Ein- und Ausgabe keine spitzen Klammern. Vergleichen Sie hierzu auch den Beispielablauf. Leerzeichen werden durch ␣ dargestellt.

C.1. Der `Zuweisungs`-Befehl

Für die Eingabe eines Zuweisungsbefehls gilt die unter 1 spezifizierte Grammatik. Für die Ausgabe wird die in der Zuweisung deklarierte Variable verwendet.

Ausgabeformat `<Variablenname>␣=␣(<Wert-der-Variable>)`
Im Erfolgsfall wird der Wert der Zuweisung im oben angegeben Format ausgegeben. Im Fehlerfall, d.h. bei ungültigen Eingaben wird eine aussagekräftige Fehlermeldung beginnend mit `Error ,␣` ausgegeben.

C.2. Der `<Variable>`-Befehl

Falls der Name einer Variable eingegeben wird, wird der aktuelle Wert der Variable auf der Kommandozeile ausgegeben.

Eingabeformat `<Variablenname>`

Ausgabeformat `<Variablenname>␣=␣(<Wert-der-Variable>)`
Im Erfolgsfall wird der Wert der Variable im oben angegeben Format ausgegeben. Im Fehlerfall, z.B. Variable wurde noch nicht deklariert, wird eine aussagekräftige Fehlermeldung beginnend mit `Error ,␣` ausgegeben.

C.3. Der `quit`-Befehl

Der parameterlose Befehl ermöglicht es, Ihr Rechenhilfsmittel vollständig zu beenden. Beachten Sie, dass hierfür keine Methoden wie `System.exit()` oder `Runtime.exit()` verwendet werden dürfen.

Eingabeformat `quit`

Ausgabeformat Im Erfolgsfall findet keine Ausgabe statt. Im Fehlerfall (z.B. bei einem falsch spezifizierten Eingabeformat) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,␣` ausgegeben.

C.4. Beispiel eines Programmablaufs

Beachten Sie auch, dass bei dem folgenden Beispielablauf die Eingabezeilen mit dem >-Zeichen gefolgt von einem Leerzeichen eingeleitet werden. Diese beiden Zeichen sind ausdrücklich kein Bestandteil des eingegebenen Befehls, sondern dienen nur der Unterscheidung zwischen Ein- und Ausgabezeilen.

Beispielablauf

```
> valueLength
Error, no previous results available
> valueLength = (10 + 30i)
valueLength = (10 + 30i)
> valueWidth = (90 + 20i)
valueWidth = (90 + 20i)
> valueComposition = valueLength + valueWidth
valueComposition = (100 + 50i)
> valueComposition = valueLength - valueWidth
valueComposition = (-80 + 10i)
> valueComposition = valueLength * valueWidth
valueComposition = (300 + 2900i)
> valueComposition = (1 + 2i) / (10 + 5i)
valueComposition = (0 -+ 0i)
> valueComposition = (5 + 20i) / (10 + 5i)
valueComposition = (1 + -1i)
> valueComposition = [valueLength + valueWidth] - valueLength
valueComposition = (-90 + -20i)

> valueLength
valueLength = (10 + 30i)
> valueComposition = valueComposition
valueComposition = (-90 + -20i)

> valueComposition = valueLength + valueLength * valueWidth
valueComposition = (310 + 2930i)
> valueComposition = valueLength + valueLength
valueComposition = (20 + 60i)
> valueLength = valueLength + valueWidth
valueLength = (100 + 50i)
> quit
```

Hier nochmal der Beispielablauf ohne Änderungshistorie:

Beispielablauf

```
> valueLength
Error, no previous results available
> valueLength = (10 + 30i)
valueLength = (10 + 30i)
> valueWidth = (90 + 20i)
valueWidth = (90 + 20i)
> valueComposition = valueLength + valueWidth
valueComposition = (100 + 50i)
> valueComposition = valueLength - valueWidth
valueComposition = (-80 + 10i)
> valueComposition = valueLength * valueWidth
valueComposition = (300 + 2900i)
> valueComposition = (1 + 2i) / (10 + 5i)
valueComposition = (0 + 0i)
> valueComposition = (5 + 20i) / (10 + 5i)
valueComposition = (1 + 1i)
> valueComposition = [valueLength + valueWidth] - valueLength
valueComposition = (90 + 20i)
> valueLength
valueLength = (10 + 30i)
> valueComposition = valueComposition
valueComposition = (90 + 20i)
> valueComposition = valueLength + valueLength * valueWidth
valueComposition = (310 + 2930i)
> valueComposition = valueLength + valueLength
valueComposition = (20 + 60i)
> valueLength = valueLength + valueWidth
valueLength = (100 + 50i)
> quit
```