Institut für Informatik
Prof. Dr.-Ing. Elke Pulvermüller
Dipl.-Systemwiss. Mathias Menninghaus

Universität Osnabrück, 17.04.2018
Testat bis 25.04.2018, 14:00 Uhr

Übungen zu Informatik B

Sommersemester 2018

Blatt 2

Allgemein

Kommentieren Sie all Ihre Programme fortan mit javadoc-Kommemntaren. In den Übungen werden javadoc-Kommentare detailliert vorgestellt.

Aufgabe 2.1: Fibonacci (25 Punkte)

Die Fibonacci Zahl $f(n)$ zu einer ganzen Zahl $n \geq 2$ sei definiert durch

$$f_n = f_{n-1} + f_{n-2} \text{ für } n \geq 2 \text{ mit } f_0 = 0 \text{ und } f_1 = 1$$

Implementieren Sie die Klasse `Fibonacci`. Ein Objekt der Klasse `Fibonacci` besitzt die Instanzmethode `next()`, die, beginnend mit $f(2)$, immer die nächste Zahl der Fibonacci-Folge zurück gibt. Beim ersten Aufruf von `next` wird also `1`, beim zweiten `2`, beim dritten `3`, dann `5`, `8` usw. zurück gegeben.

Implementieren Sie anschließend die Klasse `FibonacciPrint`. Wenn die Klasse über die Kommandozeile aufgerufen wird, soll ihr als Parameter genau eine ganze Zahl *n* übergeben werden. Diese wird in Java automatisch in das `String[]` der `main` Methode übergeben. Von `FibonacciPrint` sollen alle Fibonacci-Zahlen von $f(0)$ bis $f(n)$ wie folgt formatiert auf der Kommandozeile ausgegeben werden:

```
| n |  f(n)  |
+---+--------+
|  0|       0|
```

Benutzen Sie für die Standard Ein- und Ausgabe Java-Bordmittel und nicht die `AlgoTools.jar` aus Informatik A. Einen `String s` aus dem der `main`-Methode übergebenen `String`-Array können Sie mit `Integer.parseInt()` in ein `int` umwandeln. Ignorieren Sie dabei Exceptions vom Typ `NumberFormatException`. Achten Sie auf die Behandlung von weiteren Fehlern.

Aufgabe 2.2: Testen (25 Punkte)

Bei den jährlich stattfindenden *Hungerspielen* treten 24 *Tribute* aus den 12 Distrikten des fiktiven Landes Panem in einer künstlichen Arena gegeneinander an. Bei der 75. Durchführung der Hungerspiele ist die kreisrunde Arena analog zu einer Uhr in zwölf gleich große Bereiche unterteilt. Die Mitte der

Arena habe die Position $(0,0)$ und die Arena einen Durchmesser von 3 Meilen. Je nach Uhrzeit wird ein Bereich aktiv und in ihm werden tödliche Fallen freigeschaltet. Beispielsweise ist von 12 bis 1 Uhr der Bereich im Winkel von $0°$ bis $30°$ aktiv, wobei $0°$ die Linie von der Mitte bis zur Position $(0,1.5)$ anzeigt. Ein Tribut wird fast sicher umkommen, wenn er sich in einem aktiven Bereich aufhält oder die Arena verlässt.

Schreiben sie eine Java-Klasse `Arena`, die die Methode `int getArea(double x, double y)` enthält und zurück gibt, in welchem der zwölf Bereiche sich ein Tribut mit der Position (x,y) befindet. Wenn sich der Tribut außerhalb der Arena befindet, soll die Methode -1 zurück geben. Die Methoden in der Klasse `java.lang.Math` können dafür nützlich sein. Testen Sie Ihre Implementation ausführlich mit einem automatisierten Testprogramm. Wie viele Testfälle müssen Sie mindestens erzeugen, um alle möglichen Eingaben hinsichtlich einer korrekten Ausgabe geprüft zu haben? Für die Testklasse empfiehlt es sich, eine separate Klasse mit Testmethoden anzulegen, die Sie auch in noch folgenden Aufgaben wieder benutzen können. Wenn Sie separate Hilfs-Methoden implementieren, sollten Sie diese ebenfalls testen.

Aufgabe 2.3: EBNF und Syntaxdiagramm (25 Punkte)

Definieren Sie eine Grammatik in EBNF-Syntax um Rechenoperationen mit den Grundrechenarten auf Brüchen darstellen zu können. Berücksichtigen Sie dabei folgendes: Es können beliebig viele Brüche durch Operatoren miteinander verknüpft werden. Eine jede solche Operation und jeder Bruch wiederum kann beliebig tief mit Klammern geschachtelt werden. Ein Bruch besteht immer aus Zähler, Bruchstrich und Nenner und darf keinen Nullteiler haben, der Zähler darf aber sehr wohl Null sein. Als Operatoren sind `+`, `-`, `*` und `/` erlaubt. Brüche, Klammern und Operatoren sollten immer durch ein Leerzeichen voneinander getrennt sein. Richten Sie sich auch nach folgenden Beispielwörtern der Grammatik:

```
4/3 * 1/2 + -2/2 * ( 3/1 + -3/2 )
( -1/2 ) + 3/4 * 2/1
```

Lösen Sie diese Aufgabe **schriftlich** und zeichnen Sie passend zu Ihrer EBNF-Syntax das zugehörige Syntaxdiagramm.

Aufgabe 2.4: Bruchrechner (25 Punkte)

Erweitern Sie die Klasse `Fraction` von Blatt 1 um die Methoden `add(Fraction addend)` und `substract(Fraction subtrahend)`, die die übergebene `Fraction` addieren bzw. subtrahieren und das Ergebnis als neue Fraction zurückgeben.

Implementieren Sie zusätzlich die *Klassenmethode* `parseFraction`, die eine `Fraction` wie von der `toString`-Methode ausgegeben übergeben bekommt und die passende Instanz vom Typ `Fraction` zurückliefert. Um zu überprüfen, ob der übergebene `String` einen korrekten Bruch darstellt, sollen Sie die Methode `matches(String regex)` der Klasse `String` benutzen und für `regex` einen passenden *regulären Ausdruck* einsetzen. Erklären Sie Ihrem Tutor, welche Funktion die einzelnen Komponenten Ihres regulären Ausdrucks haben. In der Dokumentation der Klasse

`Pattern` aus der Java-API finden sie alles Wissenswerte über die Generierung eines regulären Ausdrucks in Java. Nutzen Sie zur Verarbeitung des `String` seine Methode `split` und die Methode `Integer.parseInt` vom letzten Aufgabenblatt.

Verwenden Sie die erweiterte `Fraction` anschließend für ein einfaches Rechenprogramm, das über die Kommandozeile zwei Brüche und einen Operator erhält, die so definierte Rechnung ausführt und das Ergebnis auf der Standard-Konsole ausgibt. Als Operatoren sind +, −, $\star$ und / zulässig. Achten Sie auf Fehlerbehandlung und eine geeignete Ausgabe von Fehlermeldungen auf `System.err`. Geben Sie bei Fehleingaben auch immer eine Anleitung zur Bedienung des Programms auf der Standard-Konsole aus.

Hinweise:

- Das Symbol $\star$ hat auf der Konsole eine besondere Bedeutung, deswegen geben Sie dieses beim Testen immer in `"` an. (Beispiel: `java Calculator 1/2 "*" -1/2`).
- Sie können die Musterlösung des letzten Aufgabenblattes verwenden. Diese steht in StudIP zum Download zur Verfügung.