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

Universität Osnabrück, 18.04.2017
http://www-lehre.inf.uos.de/~binf
Testat bis 26.04.2017, 14:00 Uhr

Übungen zu Informatik B

Sommersemester 2017

Blatt 2

Aufgabe 2.1: Singleton, Garbage Collector (25 Punkte)

In dieser Aufgabe sollen Sie eine Art altertümlichen Börsenticker simulieren, der alle Aktienkurse nacheinander in einer langen Zeichenkette auf dem Bildschirm ausgibt.

Implementieren Sie dazu die Klasse `Ticker` nach dem Singleton - Entwurfsmuster. Ein Objekt der Klasse `Ticker` enthält die Methode `print(String text)`, die den übergebenen `String` auf der Standard-Konsole ausgibt. Dabei werden alle Zeilenumbrüche entfernt und jeder übergebene `String` mit `+++` von den vorherigen und nachfolgenden `String`-Objekten getrennt.

Implementieren Sie zusätzlich die Klasse `Company`, die eine beliebige Firma repräsentieren soll. Jede `Company` enthält einen Namen und die Methode `changeStockPrice(double d)`, die mit dem übergebenen `double` und einer geeigneten Meldung über den auf `d` geänderten Aktienkurs die Methode `print` von `Ticker` aufruft. Außerdem soll die Klasse `Company` um einen Destruktor, der bei Aufruf einen `String` mit einer Nachricht über die Insolvenz der Firma an den `Ticker` schickt, erweitert werden.

Beispielsweise könnte die Ausgabe auf der Kommandozeile folgendermaßen aussehen:

```
+++Weyland Yutani 528.0+++Umbrella 491.0+++Dharma is insolvent
```

Nachdem Sie die oben beschriebene Funktionalität umgesetzt haben, bearbeiten Sie folgende Teilaufgaben:

1. Erklären Sie Ihrem Tutor anhand der implementierten Klassen die Vor- und Nachteile des Singleton - Entwurfsmusters
2. Implementieren Sie zusätzlich einen Testklasse, die durch die Erzeugung von Objekten des Typs `Company` und expliziten Anstoß des *Garbage Collectors* dessen Arbeitsweise im Zusammenspiel mit *Konstruktoren* und *Destruktoren* verdeutlicht. Erklären Sie Ihrem Tutor wie Sie vorgegangen sind und was man aus den Ergebnissen Ihrer Testklasse schließen kann.

Aufgabe 2.2: Copy Constructor (25 Punkte)

Betrachten Sie die Klassen `StringStack` und `StringStackEntry` mit denen eine Datenstruktur vom Typ *Keller* (engl. *Stack*) implementiert wurde, die auf Verweisen beruht. Ergänzen Sie den Quellcode um einen *Copy-Constructor*, mit dem es ermöglicht wird, eine Kopie des übergebenen

`StringStack` zu erzeugen. Beurteilen Sie, ob Ihr Konstruktor eine *deep* oder *shallow-copy* erzeugt und testen Sie sein Verhalten in einer separaten Testklasse auf Korrektheit. Für die Testklasse empfiehlt es sich, eine separate Klasse mit Testmethoden anzulegen, die sie auch in noch folgenden Aufgaben wieder benutzen können.

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 +, −, ⋆ 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.

Hinweis: Das Symbol ⋆ hat auf der Konsole eine besondere Bedeutung, deswegen geben Sie dieses beim Testen immer in " an. (Beispiel: `java Calculator 1/2 "*" -1/2`).