Fachschaft Informatik
Caffeine and Code
Timo Cramer, Sebastian Hauer

Übungsblatt 1

Einleitung

Willkommen beim Caffeine and Code, ein von Studenten angefragter Lernabend für DAP1.

Mithilfe dieses Übungszettels sollst du einige Inhalte von DAP1, die etwas mit dem Programmieren in Java zu tun haben, rekapitulieren. Ziel der Aufgaben wird es sein, ein verwendbares Programm zu programmieren, das du für persönliches Üben nach Belieben erweitern kannst.

Theorie

Aufgabe 1.1: Reverse Polish Notation (RPN)

Die auch als **postfix** bekannte Notation von mathematischen Berechnungen stellt die **Operanden** vor ihre **Operatoren** um die Ausführungsreihenfolge von Operatoren eindeutig zu definieren. So kommt diese Notation vor allem ohne Klammern aus.

Die Berechnung **3 + 4** wird in RPN folgendermaßen notiert: **3 4 +**.

Die Berechnung **(4 + 7) * 6** kann folgendermaßen notiert werden: **4 7 + 6 ***.

Nicht-Kommutative Operanden wie **-** und **/** verwenden den linken **Operanden** als ersten: **4 3 -** ergibt 1, **3 4 -** ergibt -1.

Die Semantik dieser Notation basiert auf der einer Maschine, die mit einem **Stapel** arbeitet. Wird ein **Operand** angegeben, wird er dem **Stapel** hinzugefügt. Wird ein **Operator** angegeben, werden entsprechend viele **Operanden** vom Stapel entfernt, der **Operator** mit diesen ausgeführt und das Ergebnis dem **Stapel** wieder hinzugefügt.

Um das Verständnis zu vertiefen, berechne das Ergebnis folgender Ausdrücke:

1 2 3 + + = _____

1 2 3 - - = _____

2 4 * 5 + = _____

2 4 + 5 * = _____

2 3 - 2 * = _____

2 4 2 / + = _____

4 $\sqrt{}$ **3 +** = _____

2 π cos **-** = _____

4 6 8 max **+** = _____

4 2 4 / 2 * / = _____

Hinweise: $\sqrt{}$ und cos sind unäre Operatoren, max ein binärer.

Im Folgenden werden wir nun versuchen ein Programm zu schreiben, das sowohl in der Lage ist, solche Ausdrücke interaktiv, wie ein Taschenrechner, auszuwerten, als auch in der Lage ist, Dateien einzuladen und auszuwerten.

Programmierung

Aufgabe 1.2: Formulieren der Schnittstelle

a) Zuerst sollen die Klassen für den **Stapel**, die **Operanden** und die **Operatoren** definiert werden.

   Schreibt dazu zuerst insgesamt zwei Klassen namens **Stack**, **Operand**, sowie zwei funktionale Schnittstellen **BinaryOperation** und **UnaryOperation**.

   Die Klasse **Stack** soll, anders als in der vorlesungsorientierten Übung, die Funktionalitäten direkt implementieren. Verwaltet also die **Elemente** des Stapels als interne Klasse der Klasse **Stack**. Die Klasse **Stack**, sowie die Schnittstellen für die **Operatoren** sollen einen generischen Typparameter **T** besitzen.

b) Deklariert anschließend die notwendigen Methoden der Klassen, ohne ihre Semantik zu implementieren:

   - **Stack**:
     - `void push(T value){}`
     - `T pop(){}`
     - `T peek(){}`
     - `boolean isEmpty(){}`
     - `getSize(){}`
   - **Element**:
     - `T getValue(){}`
     - `Element getNext(){}`
     - `void setNext(){}`
   - **Operatoren**:
     - `T apply(T a);` bzw. `T apply(T a, T b);`
   - **Operand**:
     - `double getValue(){}`
     - `String toString(){}`

Aufgabe 1.3: Implementieren der Schnittstelle

Nun sollen die Klassen mit Funktionalitäten gefüllt werden.

1. Implementiert den **Stapel**, indem ihr ausschließlich eine Referenz auf das oberste **Element** im **Stapel** verwaltet.
2. Implementiert die **Elemente** als reine Verwaltungsklasse, ohne eigene Algorithmen, das heißt die **Getter** und **Setter** sollen nur den Zugriff auf private Attribute verwalten.
3. Der Konstruktor des **Stapels** soll ohne Parameter einen leeren, neuen **Stapel** erzeugen.
4. Der Konstruktor der **Element**-Klasse soll den zu speichernden Wert entgegen nehmen.
5. Die **Operand**-Klasse soll nichts weiter als ein *Wrapper* für einen **double**-Wert sein.

Aufgabe 1.4: Implementierung der Applikation

Erstelle nun zwei neue Klassen: **Application**, die die **main**-Funktion beinhalten soll und den Taschenrechner starten soll und **Calculator**, die die Klasse für den eigentlichen Taschenrechner darstellt.

1. Implementiere zuerst zwei Methoden **performUnaryOperation** und **performBinaryOperation**, die jeweils einen unären oder binären Operator entgegen nehmen und wie oben beschrieben die benötigten Elemente vom Stack holt, den Operator auf ihnen anwendet und das Ergebnis wieder als Operand auf den Stack legt.
2. Anschließend sollte die Methode implementiert werden, die ein übergebenes Token (ein **String**) analysiert und den Taschenrechner zu einer entsprechenden Handlung bewegt. Wir nennen diese **chooseAction**.

   Hinweis: Eine große Fallunterscheidung über das gegebene Token ist hier eine gute Wahl, um **switch**-Anweisungen zu üben.

   So soll zum Beispiel, wenn das übergebene Token ein + ist, die Methode **performBinaryOperation** ausgeführt werden, für einen Operator, der die beiden obersten Elemente des Stacks miteinander addiert. Verwendet hierfür Lambda-Ausdrücke!

   Hinweis: Wenn kein Operator angegeben ist, sollte ein Operand erwartet werden. Verwendet die Methode **Double.parseDouble(String)**, um aus dem übergebenen String einen Double zu erhalten und diesen in einem Wrapper auf den Stack zu legen.
3. Schreibt nun eine Methode **parse**, die einen übergebenen String – der mathematische Ausdruck in RPN – in seine Einzelteile aufspaltet. Benutzt dafür entweder zum Beispiel den **StringTokenizer** der Standardbibliothek oder schreibt euren eigenen Algorithmus, der einen String anhand seiner Leerzeichen in Teilstrings zerlegt. Letztendlich soll für jedes Token aus dem an **parse** übergebenen String die **chooseAction**-Methode aufgerufen werden.
4. Ihr könnt den **Taschenrechner** nun von der Applikationsklasse aus testen, indem ihr ein neues **Taschenrechner**-Objekt erstellt und die **parse**-Methode mit verschiedenen Ausdrücken aufruft. Zum Ausgeben des Ergebnisses solltet ihr an das Ende der **parse**-Methode das oberste Element des Stacks ausgeben lassen.

Aufgabe 1.5: Erweiterungen

Ihr habt nun einen funktionierenden Taschenrechner, aber es fehlen noch Features wie Interaktivität oder die Möglichkeit, Dateien einzulesen und auswerten zu können.

Ihr könnt hierzu Eure Helfer fragen, oder im Internet selber Nachforschungen anstellen. Eurer Kreativität sind keine Grenzen gesetzt.