Programmieren in Java

http://proglang.informatik.uni-freiburg.de/teaching/java/2015/

Java-Übung Blatt 11 (Interpreter)

2015-07-15

Hinweise

- Ändern Sie nicht die Schreibweise von Bezeichnen, die auf dem Übungsblatt vorgegeben sind. Dies betrifft sämtliche global sichtbaren Namen von Eclipse-Projekten, Paketen, Klassen, etc.
- Bezeichner und Kommentare bitte auf *Englisch*!
- Schreiben Sie *sinnvolle* Kommentare.
- Laden Sie Ihre Lösungen mit Subversion (SVN) ins Übungssystem hoch. Den entsprechenden Pfad finden Sie online.
- Sollte das Übungssystem Ihre Einreichung nicht übersetzen können, dann wird sie nicht korrigiert und Sie erhalten keine Punkte.
- Die Korrektur Ihrer Abgabe wird unter dem Namen `Feedback-<login>-ex<NN>_<X>.txt` in das jeweilige Projektverzeichnis eingecheckt. Dabei ist `<login>` Ihr *myAccount*-Name und `<NN>_<X>` der Projektname.
- Sie können Ihre Gesamtpunktzahl im Übungsportal einsehen.
- Dokumentieren Sie Ihren Zeitaufwand, den gefühlten Schwierigkeitsgrad, sowie Probleme beim Lösen der Aufgabe oder auch Anregungen und Vorschläge zur Verbesserung in der Datei `<Projektverzeichnis>/erfahrungen.txt`

Hinweis: Dieses PDF stammt aus dem Archiv `ex011.zip`. Dieses Archiv enthält weiterhin noch Skelett-Projekte, die für die Bearbeitung der Aufgaben hilfreich sind.

Exercise 1 (Read–eval–print loop (REPL), 20 Punkte)
Projekt: `ex11_1`. Package: `repl`.

Ziel dieser Aufgabe ist es eine Kommandozeilenapplikation zu implementieren, die es erlaubt While-Programme Statement-weise einzugeben und auszuwerten. Das Programm soll sich wie eine sogenannte „Read-eval-print loop (REPL)" verhalten: Der Benutzer hat die Möglichkeit in einer Kommandozeile ein „Kommando" einzugeben (read). Dann wird das Kommando ausgeführt (eval) und schließlich ein Resultat angezeigt (print). Das Ganze beginnt dann von vorne (loop).

Die Kommandos für unsere REPL können die folgenden sein:

1. Ein WHILE-Statement
2. Ein arithmetischer Ausdruck
3. Ein *watch*-Kommando der Form `:watch <variable>...` (hier bedeutet `<variable>...` eine Liste von Variablennamen, die durch Leerzeichen getrennt sind)
4. Ein *unwatch*-Kommando der Form `:unwatch <variable>...`
5. Ein *quit*-Kommando der Form `:quit`, welches die REPL beendet.

Wenn ein Statement oder Ausdruck eingegeben wird, soll dieser in einer *globalen* Variablenumgebung ausgewertet werden. Das heißt, die Variablen, die in einem Statement einer Eingabe definiert werden, stehen auch für die Auswertung nachfolgender Eingaben (Statements oder Ausdrücke) zur Verfügung (siehe auch Beispiel unten).

Wenn ein watch- oder unwatch-Kommando eingebegen wird, sollen die aufgelisteten Variablen zu einer Menge *beobachteter Variablen* hinzugefügt bzw. entfernt werden.

Nach der Auswertung eines Kommandos sollen die Werte der beobachteten Variablen ausgegeben werden. Wenn ein Ausdruck eingegeben wurde, dann soll zusätzlich noch das Ergebnis ausgegeben werden.

Danach beginnt die Loop von vorne.

Hier ist eine Bespiel-Session mit dem Programm. Die Benutzereingaben sind in den Zeilen die mit `>>>` beginnen. Das `>>>`, ein sogenannter „Prompt", wird noch vom Progamm ausgegeben. Es soll dem Benutzer signalisieren, dass eine Eingabe erwartet wird.

```
Welcome to WHILE-REPL!
>>> :watch x y z
x: <undefined>
y: <undefined>
z: <undefined>
>>> x = 5
x: 5
```

```
y: <undefined>
z: <undefined>
>>> y = x + 2
x: 5
y: 7
z: <undefined>
>>> z = 10
x: 5
y: 7
z: 10
>>> while (z + (-1) * x) x = x + 1
x: 10
y: 7
z: 10
>>> z = a + x
Not found: value a
x: 10
y: 7
z: 10
>>> --a^8
Invalid expression
x: 10
y: 7
z: 10
>>> :unwatch z
x: 10
y: 7
>>> :unwatch x y
>>> 2 * (x + y) + z
Result: 44
>>>
```

1. Implementieren Sie die wesentliche Funktionalität wie oben beschrieben. Sie können dabei wesentliche Teile aus der Vorlesung übernehmen. Versuchen Sie nicht alles neu zu implementieren!

   Das mitgelieferte Skelett demonstriert eine Möglichkeit wie man Eingaben zeilenweise in Java einlesen kann. Wenn Sie möchten, können Sie Ihre REPL auf diesem Skelett aufbauen.

   Hinweis: Zum Testen kann es hilfreich sein `toString`-Methoden für Statements analog zu den arithmetischen Ausdrücken zu implementieren.

   Hinweis: Hier noch einmal, der Vollständigkeit halber, die Grammatik für WHILE Programme, die der Parser aus der Vorlesung unterstützt:

   ```
   STMT ::= if ( <EXPR> ) <STMT> else <STMT>
          | while ( <EXPR> ) <STMT>
          | <VAR> = <EXPR>
          | { <REST>
   REST ::= } | <STMT> ; <REST>
   ```

   (für `<Var>` und `<EXPR>`, siehe Vorlesung)

2. Stellen Sie sicher, dass Ihre REPL die Benutzereingaben komplett parst und nicht bespielsweise den Suffix einer Eingabe nach dem ersten erfolgreich geparsten Statement ignoriert: Hier ist sind Beispiele des gewünschten Verhaltens:

   ```
   >>> :watch x
   x : <undefined>
   >>> x=42 x=42
   Invalid expression
   x : <undefined>
   >>> x ^^^ 5
   Invalid expression
   x : <undefined>
   >>> x=42
   x : 42
   ```

```
>>> if (x) y=0 else y=1 bar
Invalid expression
x : 42
>>> x foo
Invalid expression
x : 42
```

Hier ist ein Beispiel von unerwünschtem Verhalten, bei dem der ungültige Rest einer Zeile ignoriert wird:

```
>>> :watch x
x : <undefined>
>>> x=42 x=42
x : 42
>>> x ^^^ 5
x : 42
```

Hinweis: Man kann mit dem regulärem Ausdruck `$` überprüfen ob das Zeilenende in einem String erreicht ist.

3. Stellen Sie sicher, dass die Schlüsselworte `while` und `if` niemals als Variablennamen angesehen werden. Im Statement-Parser ist das bereits durch Fallunterscheidung auf den Tokens sichergestellt. Der Expression-Parser behandelt `while` und `if` jedoch noch als Variablennamen. Beheben Sie dieses Problem. Im Ergebnis weist die REPL folgendes Verhalten auf.

```
>>> while = 42
Invalid expression
>>> if = 42
Invalid expression
>>> if
Invalid expression
>>> while
Invalid expression
>>> :watch while
Invalid expression
>>> :unwatch if
Invalid expression
```

4. Unterstützen Sie mehrzeilige Eingaben. Der Benutzer soll eine mehrzeilige Eingabe mit einem Backslash am Ende jeder Zeile kennzeichnen:

```
>>> 3 + \
5
Result: 8
>>> while (x) \
      x = x + (-1)
>>>
```

5. Ändern Sie Parser und Auswertung so ab, dass Zuweisungen Ausdrücke sind (anstatt wie bisher Statements). Eine Zuweisung gibt dann den zugewiesenen Wert zurück (und setzt gleichzeitig ihre Zielvariable). Beispiel:

```
Welcome to WHILE-REPL!
>>> :watch x y
x: <undefined>
y: <undefined>
>>> 42 + (x = 5)
x: 5
y: <undefined>
Result: 47
>>> x = (y = 42)
x : 42
y : 42
Result: 42
```

Zuweisungen können weiterhin als Statements verwendet werden, d.h. `while (x) x=0` soll weiterhin funktionieren. Halten Sie die üblichen Präzedenz- und Assoziativitätsregeln für Operatoren ein.[1]

Ein letzter Hinweis: Wenn Sie bei einem der Aufgabenteile nicht weiterkommen, versuchen Sie bei einem anderen weiterzumachen. Informieren Sie aber dann Ihren Tutor in der Feedbackdatei darüber, welche Aufgabenteile Sie vollständig bearbeitet haben, und welche nur teilweise.

[1] `http://www.cs.bilkent.edu.tr/~guvenir/courses/CS101/op_precedence.html`