Das Programm Slang (kurz für Stack Language) erstellt einen leeren Stack von Values auf dem dann unterschiedliche Operationen ausgeführt werden können. Die Operationen werden von System.in mittels AtomScanner gelesen. Operationen können den Inhalt des Stacks verändern und Nebeneffekte (zB Ausgaben) erzeugen. Details zu den einzelnen Operationen folgen weiter unten.

Bereitgestellte Dateien

AtomScanner.java: Scanner für "Strings mit Leerzeichen" (siehe unten)

Lexer.java, LexerException.java, SourcePosition.java, Token.java: Klassen, die für den AtomScanner benötigt werden

SlangException.java: Basisklasse für alle zu definierenden Exception-Klassen für Slang

Value.java: gemeinsames Interface aller Slang-Werte (siehe unten)

Slang.java: ausführbare Klasse und Startgerüst für das Programm Slang Diese Datei sollen sie bearbeiten und gegebenenfalls weitere Klassen erstellen.

progs/: Beispielprogramme zum Testen ihrer Implementierung

AtomScanner

Im Gegensatz zum java.util.Scanner berücksichtigt der AtomScanner auch öffnende und schließende Anführungszeichen sowie Kommentare (// bis Zeilenende und /* bis */).

Beispieleingabe: "Hello World" print
java.util.Scanner: ("Hello) (World") (print)
AtomScanner: ("Hello World") (print)

Grammatik:

Token ::= String | Atom | Integer

String ... Zeichenkette zwischen doppelten Anführungszeichen "" oder alternativ dazu umschlossen von geschwungenen Klammern {} (zB wenn der String Anführungszeichen enthalten soll).
zB "Foo"
zB {String mit "Anführungszeichen"}

Atom ... Zeichenfolge ohne Whitespace
zB print
zB .s

Integer ... ganze Zahl
zB 42

Die folgenden Methoden des AtomScanner können verwendet werden:

```
boolean hasNext() -- true, wenn weitere Eingabe verfügbar ist

boolean hasNextInteger() -- true, wenn die nächste Eingabe eine Zahl ist
boolean hasNextString() -- true, wenn die nächste Eingabe ein String ist
boolean hasNextAtom() -- true, wenn die nächste Eingabe ein Atom ist

int nextInteger() -- die nächste Eingabe als int
String nextAtom() -- die nächste Eingabe als String
String nextString() -- die nächste Eingabe als String
```

Value

Slang speichert ganze Zahlen und Strings auf einem Stack. Als gemeinsames Interface der Werte in Slang-Programmen dient das Interface Value, das die Methoden toS() und toI() vorschreibt.

String toS() -- liefert eine textuelle Repräsentation des Wertes.

int toI() -- liefert die Zahl als Integer bzw 0 für nicht numerische Werte.

Fehlerbehandlung

Fehler beim Einlesen und Verarbeiten von Slang-Programmen werden per Exception-Handling verarbeitet und führen zu Fehlermeldungen gefolgt von einem Abbruch des Programms. Details zu den Fehlermeldungen entnehmen Sie bitte den mitgelieferten Beispielprogrammen.

Testen der Implementierung

Eine Sammlung an Beispielprogrammen finden Sie zusammen mit der Angabe. Jedes der Beispielprogramme (*.slang) erzeugt eine Ausgabe, die sie mit der gleichnamigen .output Datei vergleichen können. Wenn ihre Implementierung von Slang für ein beliebiges Slang-Programm FOO.slang eine andere Ausgabe erzeugt als in FOO.output enthalten, ist ihre Implementierung nicht korrekt.

Operationen

Es folgt die Beschreibung der insgesamt 19 Operationen in Slang.

Die Angabe in der Klammer gibt an, wieviele Elemente durch einen Befehl vom Stack entfernt und wieviele auf den Stack gelegt werden.

Beispiel: (a b c -- z) dieser Befehl entfernt die drei obersten Elemente vom Stack und legt anschließend einen Wert zurück auf den Stack.

Beispiel: (--) dieser Befehl verändert den Inhalt des Stacks nicht.

<String> (-- z)
Ein String wird auf den Stack gelegt.

<Integer> (-- z)
Ein Integer wird auf den Stack gelegt.

.s (--)
Gibt den aktuellen Inhalt des Stacks aus.

Beispiel: 1 2 3 .s
```
-- Top of Stack:
3
2
1
-- Bottom
```

prin (x --)
Entfernt das oberste Element vom Stack und gibt es aus.

print (x --)
Wie prin aber gefolgt von einem Zeilenwechsel.

sub (x y -- z)
Entfernt die obersten beiden Elemente vom Stack und legt die Differenz der numerischen Werte auf den Stack. z := x - y

add (x y -- z)
Wie sub aber z := x + y, Addition

mul (x y -- z)
Wie sub aber z := x * y, Multiplikation

div (x y -- z)
Wie sub aber z := x / y, ganzzahlige Division

mod (x y -- z)
Wie sub aber z := x modulo y, Divisionsrest

read (src -- string)
Entfernt das oberste Element vom Stack. Wenn es ein String ist, der mit "http://" oder "https://" beginnt, wird der Inhalt der angegebenen Webseite als String auf den Stack gelegt. Andernfalls wird die angegebene Datei gelesen und ihr Inhalt als String auf den Stack gelegt.

```
write ( dest string -- )
  Überschreibt die durch den String <dest> angegebene Datei mit dem
  Inhalt von <string>.

ask ( prompt -- string )
  Zeigt ein Dialogfeld (JOptionPane.showInputDialog) an, das zur
  Eingabe einer Zeichenkette auffordert. <prompt> wird dabei im
  Dialogfeld angezeigt. Die zur Laufzeit eingegebene Zeichenkette wird
  auf den Stack gelegt.

askn ( prompt -- int )
  Wie ask aber die Eingabe wird als Zahl auf den Stack gelegt.

append ( s1 s2 -- string )
  Entfernt zwei Elemente vom Stack und hängt sie als Zeichenketten
  aneinander. Das Ergebnis wird auf den Stack gelegt.
  Beispiel: "foo" "bar" append -- "foobar"

skip-to ( s1 pattern -- s2 )
  Entfernt zwei Elemente vom Stack. Wenn der String <pattern> im
  String <s1> enthalten ist, wird <s1> ab der Stelle des ersten
  Vorkommens von <pattern> auf den Stack gelegt, andernfalls wird der
  leere String auf den Stack gelegt.

  Beispiel: "Das ist ein String" "ein" skip-to -- "ein String"
  Beispiel: "Das ist ein String" "kein" skip-to -- ""

skip-n ( s1 n -- s2 )
  Entfernt zwei Elemente vom Stack. Legt den String <s1> beginnend mit
  dem <n>-ten Zeichen auf den Stack.

  Beispiel: "Beispiel" 0 skip-n -- "Beispiel"
  Beispiel: "Beispiel" 3 skip-n -- "spiel"

trunc ( s1 n -- s2 )
  Entfernt zwei Elemente vom Stack. Legt den String <s1> bis zum
  <n>-ten Zeichen inklusive auf den Stack.

  Beispiel: "Beispiel" 3 trunc -- "Bei"
  Beispiel: "Beispiel" 12 trunc -- "Beispiel"
  Beispiel: "Beispiel" 0 trunc -- ""

copy-to ( s1 pattern -- s2 )
  Entfernt zwei Elemente vom Stack. Wenn der String <pattern> im
  String <s1> enthalten ist, wird <s1> bis zur Stelle des ersten
  Vorkommens von <pattern> aber ohne <pattern> selbst auf den Stack
  gelegt, andernfalls wird der leere String auf den Stack gelegt.

  Beispiel: "Das ist ein String" "ein" copy-to -- "Das ist "
  Beispiel: "Das ist ein String" "kein" copy-to -- ""

dup ( x -- x x )
  Legt das oberste Element des Stacks noch einmal auf den Stack.
  Beispiel: 1 2 3 dup -- 1 2 3 3

pop ( x -- )
  Entfernt das oberste Element des Stacks.
  Beispiel: 1 2 3 pop -- 1 2
```