Aufgabe 2: Interpreter für boolesche Ausdrücke

In Übungsaufgabe 7 von Kapitel 4 haben wir eine Skriptsprache für boolesche Ausdrücke spezifiziert und ihre Ausführung durch eine attributierte Grammatik beschrieben. Implementieren Sie nun mittels Coco/R einen Interpreter, der Programme dieser Skriptsprache verarbeitet und ausführt. Es soll dabei kein Code erzeugt werden, sondern die Anweisungen sollen direkt nach ihrer Erkennung ausgeführt werden.

Als Parserbeschreibung können Sie die attributierte Grammatik in Coco/R-Notation verwenden. Als Scannerbeschreibung brauchen wir lediglich eine Spezifikation des Terminalsymbols ident. Verwalten Sie die Symbolliste in einer Hashtabelle, die in den globalen semantischen Deklarationen der Compilerbeschreibung deklariert wird. Schreiben Sie ein Hauptprogramm, das einen Scanner und einen Parser erzeugt und die Syntaxanalyse (und damit den Interpreter) startet. Das Hauptprogramm soll den Namen der Datei mit dem Skript-Programm als Kommandozeilenparameter entgegennehmen.

Erzeugen Sie mittels Coco/R aus der Compilerbeschreibung einen Scanner und einen Parser, und übersetzen Sie die beiden zusammen mit dem Hauptprogramm, um den lauffähigen Interpreter zu erhalten.

Lösung

```
import java.util.HashMap;

COMPILER BoolScript

//---------------------- global declarations ----------------------------
   HashMap<String, Boolean> values = new HashMap<String, Boolean>();  // symbol table

//---------------------- scanner specification ----------------------------
CHARACTERS
   letter = 'A'..'Z' + 'a'..'z'.

TOKENS
   ident = letter {letter}.

IGNORE '\r' + '\n' + '\t'

//---------------------- parser specification ----------------------------
PRODUCTIONS

BoolScript = { Statement }.

Statement                       (. boolean val; .)
=  ident                        (. String name = t.val; .)
   "="
   Expr <out val>               (. values.put(name, val); .)
  ";"
|  "print"
   Expr <out val>               (. System.out.println(val); .)
   ";".

Expr <out boolean val>          (. boolean val2; .)
=  Term <out val>
   {  "||" Term <out val2>      (. val = val || val2; .)
   }.

Term <out boolean val>          (. boolean val2; .)
=  Factor<out val>
   {  "&&" Factor <out val2>    (. val = val && val2; .)
   }.
```

```
Factor <out boolean val>
=                                (. val = false; .)
( "true"                         (. val = true; .)
| "false"                        (. val = false; .)
| ident                          (. Boolean c = values.get(t.val);
                                   if (c != null) val = c; else { SemErr("undeclared name"); val = false; } .)
| "(" Expr <out val> ")"
| "!" Factor <out val>           (. val = ! val; .)
).

END BoolScript.
```

Die in der Scriptsprache verwendeten Namen und ihre booleschen Werte werden in einer Hashtabelle values gespeichert, die als *Symbolliste* dient und in den globalen semantischen Deklarationen deklariert wird.

Die Scannerbeschreibung spezifiziert das Terminalsymbol ident als Folge von Buchstaben. Außerdem legt sie fest, dass neben Leerzeichen auch Zeilenenden und Tabulatoren ignoriert werden sollen.

Die Parserbeschreibung ist eine attributierte Grammatik. Expr, Term und Factor führen die Berechnung eines booleschen Ausdrucks durch. Der Wert von Namen wird aus der Symbolliste entnommen. Eine Zuweisung trägt den berechneten booleschen Wert unter dem Namen auf der linken Seite der Zuweisung in die Symbolliste ein. Die print-Anweisung gibt den Wert eines booleschen Ausdrucks auf der Konsole aus.

Wir brauchen noch ein Hauptprogramm, das den Namen der Script-Datei als Kommandozeilenparameter liest, einen Scanner und einen Parser erzeugt und die Syntaxanalyse (und somit die Interpretation) startet:

```
class BoolScript {
   public static void main (String[] arg) {
      Scanner scanner = new Scanner(arg[0]);
      Parser parser = new Parser(scanner);
      parser.Parse();
      System.out.println(parser.errors.count + " errors detected");
   }
}
```

Wir verarbeiten nun die Compilerbeschreibung mit Coco/R:

```
java -jar Coco.jar BoolScript.atg
```

wodurch ein Scanner und ein Parser erzeugt wird. Anschließend übersetzen wir alle Java-Dateien

```
javac Scanner.java Parser.java BoolScript.java
```

und führen dann den Interpreter wie folgt aus:

```
java BoolScript script.txt
```

Die Datei script.txt enthält das Scriptprogramm, das wie folgt aussehen könnte:

```
big = true;
small = ! big;
ready = false;
print (big || small) && ready || big && ! ready;
```

Die Ausgabe ist in diesem Fall

```
true
```