Institut für Systemsoftware **Übungen zu Übersetzerbau WS 2004/2005**

Prof. Dr. H. Mössenböck **Übung 2**

Zuname _____ Matr. Nr. _____

Übungsgruppe: Punkte _____ korr. _____

- ❑ 1 (Kotzmann) Do 10^{15} - 11^{45}
- ❑ 2 (Wimmer) Do 12^{00} - 13^{45}
- ❑ 3 (Wöß) Do 10^{15} - 11^{45}

Letzter Abgabetermin:
Donnerstag, 28.10.2004, 8^{15} Uhr

Lexikalischer Analysator (*Scanner*)

1) Implementierung (22 Punkte)

Beginnen Sie die Arbeit an Ihrem *MicroJava*-Compiler, indem Sie einen lexikalischen Analysator (*Scanner*) für die Sprache *MicroJava* schreiben. Lesen Sie dazu die Sprachdefinition (VO-Skript Seiten 8-13) genau durch. Welche Terminalsymbole und Terminalklassen werden in der Grammatik gebraucht? Wie sehen die Kommentare aus, wie die Zeichenkonstanten? Welche Schlüsselwörter und vordeklarierten Namen gibt es?

Folgende Schnittstellen sind gegeben:
(Sie finden die Dateien auf der Homepage (`UB-UE2-Angabe.zip`))

Datei *Token.java*:

```
package ssw.mj;
public class Token {
  // ----- token codes
  public static final int
    none  =  0,
    ident =  1, number =  2, charConst =  3, …,
    if_   = 34, // append '_', because 'if' is a Java keyword
    new_  = 35, …,
    eof   = 42;

    int kind;    // token kind
    int line;    // token line
    int col;     // token column
    int val;     // token value for number & charConst
    String str;  // string representation of token for ident & number
}
```

Datei *Scanner.java*:

```
package ssw.mj;
public class Scanner {
  //----- static variables
  private static Reader in;        // input stream (source file)
  private static PrintWriter out;  // output stream
  private static char ch;          // lookahead character
  private static int line;         // current line
  private static int col;          // current column

  //----- prints error message
  private static void error (String msg) { … }
  //----- initializes scanner
  public static void init (Reader r) { … }
  //----- returns next token
  public static Token next () { … }
}
```

Beachten Sie:

- Bei jedem Aufruf von *next()* muss das nächste erkannte Terminalsymbol (= *Token*) zurückgegeben werden. Wenn das Ende des Eingabestroms erreicht ist, wird das Terminalsymbol *eof* zurückgegeben.
- Schlüsselwörter, Bezeichner, (positive ganze) Zahlen, Zeichenkonstanten, sowie Operatoren müssen erkannt werden.
- Leerzeichen, Tabulatoren und Zeilenumbrüche müssen überlesen werden.
- Kommentare (von "/*" bis "*/" auch geschachtelt) müssen überlesen werden.
- Folgende Ereignisse haben einen lexikalischen Fehler zur Folge:
  - Das Auftreten ungültiger Zeichen (`INVALID_CHAR, INVALID_CHARCONST, UNDEF_ESC`).
  - Fehlende schließende Anführungszeichen bei Zeichenkonstanten (`MISS_SQUOTE`).
  - Leere Zeichenkonstanten (`EMPTY_CHARCONST`).
  - Zu große Zahlenkonstanten (`BIG_NUM`): der Wertebereich von *int* ist wie in *Java* von -2147483648 bis 2147483647 definiert.
    Achtung: der lexikalische Analysator muss nur positive ganze Zahlen liefern (also von 0 bis 2147483647). Negative Zahlen werden erkannt, indem ein Minuszeichen und eine positive ganze Zahl als zwei separate Tokens bei wiederholtem Aufruf von *next()* geliefert werden. (*Anm.*: -2147483648 kann daher nicht als Konstante im Programm direkt angegeben werden. Will man diese Zahl tatsächlich verwenden, müsste man -2147483647-1 schreiben).
- Zusätzlich zu den im VO-Skript definierten Schlüsselwörtern werden wir in den Übungen auch noch ***do*** für die do-while-Schleife verwenden.

Lexikalische Fehler sollen Fehlermeldungen (siehe Datei `messages.properties`) nach sich ziehen. Geben Sie diese unter Angabe von Zeilen- und Spaltenposition der Fehlerstelle aus.

Beispiel: `-- line 10, col 8: invalid character 'Œ'!`

b) Test

Implementieren Sie alle Teile Ihres Compilers so, dass er mit den vorgegebenen JUnit-Testfällen getestet werden kann. Auch die Tutoren verwenden diese (und ev. weitere) Testfälle und die Übungen werden nur beurteilt, wenn die Testfälle ausführbar sind.

Sie können die gegebenen JUnit-Testfälle auch als Vorlage für Ihre eigenen Tests verwenden, um so Ihren Compiler noch ausführlicher zu testen.

Testen Sie auf jeden Fall alle in der Spezifikation angegebenen Eigenschaften.
Testen Sie auch die Reaktionen bei fehlerhaften Eingaben!
Richtlinien und Anleitungen zum Testen finden Sie auf der UE-Homepage.

2) Abgabe (2 Punkte)

Die Abgabe der Übungen 2-6 soll auf *Papier* **UND** *elektronisch* erfolgen.
Um das elektronische Abgabesystem zu testen, gibt es für diese Aufgabe alleine schon Punkte! Probieren Sie die elektronische Abgabe aus, indem Sie eine Datei **`test.zip`**

bis spätestens **Mittwoch, den 20. Oktober, 2004, 12:00 (mittags)**

"abgeben" (diese muss nichts Sinnvolles enthalten und möglichst unter 1KB groß sein).

Richtlinien und Anleitungen für die Abgabe finden Sie auf der UE-Homepage.