McMaster University
Department of Computing and Software
Dr. W. Kahl

SFWR ENG 3E03
Exercise Sheet 2

Design and Selection of Programming Languages

21 September 2005

Exercise 2.1 — Lexing (25% Midterm 3, 2003)

Similar to the *nextToken* procedure in the *TokenStream* class for the textbook language Jay, **implement** the *nextToken* procedure in the *TokenStream* class for a language with the following lexical syntax:

- Comments, extending from the "#" character to the end of the line, are ignored
- Whitespace is ignored.
- Tokens of type *Name* are formed from letters only.
- Each character (occurring outside a comment) that is neither a letter, nor whitespace, is considered as a token of type *Special* by itself.

  Examples are "+", "5", "~", "'", "(", "]".

This should use the same *TokenStream* class context as in the textbook example:

```
class Token {
   public String type;     // token type: either "Name" or "Special"
   public String value;    // token value
}

public class TokenStream {
   private boolean isEof = false;
   private char nextChar = ' ';       // next character in input stream
   private BufferedReader input;

   public TokenStream (String fileName) {  ...  /* same as for Jay */ }
   private char readChar () { ...  /* same as for Jay */ }
```

In addition, you may assume the following functions to be defined appropriately:

```
   private boolean isWhiteSpace (char c) { ... }
   private boolean isEndOfLine (char c) { ... }
   private boolean isLetter (char c) { ... }
```

Exercise 2.2 — Recursive Descent (Textbook Ex. 2.13)

Starting from the skeleton ConcreteSyntax.java that can be found among the student materials on the textbook's web site, complete the implementation recursive descent parser for Jay.

Exercise 2.3 — Expression Manipulation in Java

Substitution $e_1[v \mapsto e_2]$ of an expression e_2 for a variable v in an expression e_1 is defined as follows:

$$
\begin{array}{lcll}
v[v \mapsto e] & = & e & \\
w[v \mapsto e] & = & w & \text{if } v \neq w \\
k[v \mapsto e] & = & k & \text{for } k \in \mathit{Num} \\
(e_1 \oplus e_2)[v \mapsto e] & = & (e_1[v \mapsto e]) \oplus (e_2[v \mapsto e]) & \text{for } \oplus \in \mathit{Op}
\end{array}
$$

This exercise further modifies the expression classes of Exercise 1.3.

(a) Add an instance method *substituteVariable* that takes as arguments a variable, and an expression to be substituted into that variable, and **returns the result of the substitution** into the expression for which the method is called.

(b) Add an instance method *destructivelySubstituteVariable* that takes as arguments a variable, and an expression to be substituted into that variable, and **modifies the expression object for which the method is called** by performing the substitution.

(c) Discuss the difference between these two methods!