CSE505 – Fall 2014

Assignment 2 – Object Oriented Parsing

(may be done by a team of two students)

Assigned Weds, Oct 1 (rev Oct 2)

Due, Weds, Oct 15 (11:59 pm, online submission)

Consider the following grammar for a simple programming language, TinyPL:

```
program ->  decls stmts end
decls   ->  int idlist ;
idlist  ->  id { , id }
stmts   ->  stmt [ stmts ]
cmpdstmt->  '{' stmts '}'
stmt    ->  assign | loop | cond
assign  ->  id = expr ;
loop    ->  while '(' rexp ')' cmpdstmt
cond    ->  if '(' rexp ')' cmpdstmt [ else cmpdstmt ]
rexp    ->  expr (< | > | = | !=) expr
expr    ->  term   [ (+ | -) expr ]
term    ->  factor [ (* | /) term ]
factor  ->  int_lit | id | '(' expr ')'
```

Write an object-oriented top-down parser in Java that translates every TinyPL program into an equivalent sequence of byte codes for a Java Virtual Machine. It would be helpful if you develop your program in three stages, as follows, but you only need to submit the result of Stage 3:

Stage 1: Assume that stmt is of the form: `stmt -> assign`
Stage 2: Assume that stmt is of the form: `stmt -> assign | loop`
Stage 3: Assume that stmt is of the form: `stmt -> assign | loop | cond`

Assumptions:

1. All input test cases will be syntactically correct; syntax error-checking is not necessary.
2. An `id` is a single character, and an `int_lit` is an unsigned integer.
3. Follow Java bytecode naming convention for opcodes as well as for int literals. Generate iconst, bipush, or sipush depending upon the numeric value of the int literal.

Program Structure:

1. There should be one Java class definition for each nonterminal of the grammar. Place the code for the top-down procedure in the class constructor itself.
2. There should be a top-level driver class, called Parser, and another class, called Code, which has methods for code generation.
3. The code for the lexical analyzer will be given to you.

Output:

1. For each test case, show the byte code generated, as well as the object and sequence diagrams produced by JIVE at the end of execution:
   a. In generating the object diagram, choose the "Stacked" (i.e., without tables) option while saving the object diagram.
   b. In generating the sequence diagram, make sure that Buffer, Lexer, and Token are added to the "Exclusion Filter" so that they do **not** appear in the diagram.
2. Sample test cases and their outputs will be posted on Piazza. File naming convention for submission will also be posted as well as further clarifications, as needed.

End of Assignment 2