Assignment 2

Due: 29 September at 11:59pm

Implement a recursive descent parser for the following context-free grammar:

Program ::= IDENTIFIER (Declaration SEMI | Statement SEMI)*
Declaration :: = VariableDeclaration | ImageDeclaration | SourceSinkDeclaration
VariableDeclaration ::= VarType IDENTIFIER (= Expression | ε)
VarType ::= KW_int | KW_boolean
SourceSinkDeclaration ::= SourceSinkType IDENTIFIER OP_ASSIGN Source
Source ::= STRING_LITERAL
Source ::= OP_AT Expression
Source ::= IDENTIFIER
SourceSinkType := KW_url | KW_file
ImageDeclaration ::= KW_image (LSQUARE Expression COMMA Expression RSQUARE | ε)
IDENTIFIER (OP_LARROW Source | ε)
Statement ::= AssignmentStatement
| ImageOutStatement
| ImageInStatement
ImageOutStatement ::= IDENTIFIER OP_RARROW Sink
Sink ::= IDENTIFIER | KW_SCREEN //ident must be file
ImageInStatement ::= IDENTIFIER OP_LARROW Source
AssignmentStatement ::= Lhs OP_ASSIGN Expression
Expression ::= OrExpression OP_Q Expression OP_COLON Expression
| OrExpression
OrExpression ::= AndExpression (OP_OR AndExpression)*
AndExpression ::= EqExpression (OP_AND EqExpression)*
EqExpression ::= RelExpression ((OP_EQ | OP_NEQ) RelExpression)*
RelExpression ::= AddExpression ((OP_LT | OP_GT | OP_LE | OP_GE) AddExpression)*
AddExpression ::= MultExpression ((OP_PLUS | OP_MINUS) MultExpression)*
MultExpression := UnaryExpression ((OP_TIMES | OP_DIV | OP_MOD) UnaryExpression)*
UnaryExpression ::= OP_PLUS UnaryExpression
| OP_MINUS UnaryExpression
| UnaryExpressionNotPlusMinus
UnaryExpressionNotPlusMinus ::= OP_EXCL UnaryExpression | Primary
| IdentOrPixelSelectorExpression | KW_x | KW_y | KW_r | KW_a | KW_X | KW_Y | KW_Z | KW_A | KW_R | KW_DEF_X | KW_DEF_Y
Primary ::= INTEGER_LITERAL | LPAREN Expression RPAREN | FunctionApplication
IdentOrPixelSelectorExpression::= IDENTIFIER LSQUARE Selector RSQUARE | IDENTIFIER
Lhs::= IDENTIFIER (LSQUARE LhsSelector RSQUARE | ε)
FunctionApplication ::= FunctionName LPAREN Expression RPAREN
| FunctionName LSQUARE Selector RSQUARE

```
FunctionName ::= KW_sin | KW_cos | KW_atan | KW_abs
        | KW_cart_x | KW_cart_y | KW_polar_a | LW_polar_r
LhsSelector ::= LSQUARE  ( XySelector  | RaSelector  )  RSQUARE
XySelector ::= KW_x COMMA KW_y
RaSelector ::= KW_r , KW_A
Selector ::=  Expression COMMA Expression
```

- Use the provided SimpleParser.java and SimpleParserTest.java as a starting point. Your Scanner.java from assignment 1 (with any errors corrected) will also be needed.
- As given, the code should compile, but all of the test cases will fail. They should all pass when your parser has been completely and correctly implemented. Some of the methods in SimpleParser.java throw an UnsupportedOperationException. This is for convenience during development—you can use it to mark something as incomplete in a way that allows the code to be compiled, but will still let you know if you accidentally call it. The final version of your parser should NOT throw an UnsupportedOperationException.
- The parser should simply determine whether the given sentence is legal or not. If not, the parser should throw a SyntaxException. If the sentence is legal, the parse method should simply return.
- Use the approach described in the lectures to systematically build the parser. If the grammar is not LL(1), you may need to transform it, or use an ad hoc solution.

Turn in a jar file containing your source code for Parser.java, Scanner.java, and ParserTest.java.

Your ParserTest will not be graded, but may be looked at in case of academic honesty issues. We will subject your parser to our set of unit tests and your grade will be determined solely by how many tests are passed. Name your jar file in the following format:
firstname_lastname_ufid_hw2.jar

Additional requirements:

- This code must remain in package cop5556fa17(case sensitive): do not create additional packages.
- Names (of classes, method, variables, etc.) in starter code must not be changed.
- Unless otherwise specified, your code should not import any classes other than those from the standard Java distribution or your Scanner.java.
- All code, including the Scanner code must be your own work. Using someone else's Scanner code is not permitted.

Submission Checklist

See the checklist from Assignment 1.

Comments and suggestions:

Work incrementally. Note that you can call the routines corresponding to fragments of the grammar in Junit tests. For example, see the expression1 test in the provided code.

It is useful when working incrementally to ensure that you are not calling an unimplemented procedure without realizing it. To this end, the sample code throws an UnimplementedOperationException in methods that are needed to allow the code to compile,, but not yet implemented. After all of your methods have been implemented, you should have not more need of this.

We will not grade the contents of error message, but you will be much happier later if you make them informative. In particular, you should include the line number, etc. of the offending token.