Assignment 3

Due: 12 October at 11:59pm

Modify your parser from Assignment 2 so that it returns an Abstract Syntax Tree.

Concrete Syntax	**ASTNode**
Program ::= IDENTIFIER (Declaration SEMI | Statement SEMI)*	Program
Declaration :: = VariableDeclaration | ImageDeclaration | SourceSinkDeclaration	
VariableDeclaration ::= VarType IDENTIFIER (OP_ASSIGN Expression | ε)	Declaration_Variable
VarType ::= KW_int | KW_boolean	
SourceSinkDeclaration ::= SourceSinkType IDENTIFIER OP_ASSIGN Source	Declaration_SourceSink
Source ::= STRING_LITERAL	Source_StringLiteral
Source ::= OP_AT Expression	Source_CommandLInePar am
Source ::= IDENTIFIER	Source_Ident
SourceSinkType := KW_url | KW_file	
ImageDeclaration ::= KW_image (LSQUARE Expression COMMA Expression RSQUARE | ε) IDENTIFIER (OP_LARROW Source | ε)	Declaration_Image
Statement ::= AssignmentStatement | ImageOutStatement | ImageInStatement	
ImageOutStatement ::= IDENTIFIER OP_RARROW Sink	Statement_Out
Sink ::= IDENTIFIER	Sink_Ident
Sink ::= KW_SCREEN	Sink_SCREEN
ImageInStatement ::= IDENTIFIER OP_LARROW Source	Statement_In
AssignmentStatement ::= Lhs OP_ASSIGN Expression	Statement_Assign
Expression ::= OrExpression OP_Q Expression OP_COLON Expression	Expression_Conditional
Expression ::= OrExpression	
OrExpression ::= AndExpression (OP_OR AndExpression)*	Expression_Binary
AndExpression ::= EqExpression (OP_AND EqExpression)*	Expression_Binary
EqExpression ::= RelExpression ((OP_EQ | OP_NEQ) RelExpression)*	Expression_Binary
RelExpression ::= AddExpression ((OP_LT | OP_GT | OP_LE | OP_GE) AddExpression)*	Expression_Binary
AddExpression ::= MultExpression ((OP_PLUS | OP_MINUS) MultExpression)*	Expression_Binary
MultExpression := UnaryExpression ((OP_TIMES | OP_DIV | OP_MOD) UnaryExpression)*	Expression_Binary
UnaryExpression ::= OP_PLUS UnaryExpression | OP_MINUS UnaryExpression | UnaryExpressionNotPlusMinus	Expression_Unary
UnaryExpressionNotPlusMinus ::= OP_EXCL UnaryExpression	Expression_Unary

UnaryExpressionNotPlusMinus ::= Primary	
UnaryExpressionNotPlusMinus ::= IdentOrPixelSelectorExpression	
UnaryExpressionNotPlusMinus ::= KW_x | KW_y | KW_r | KW_a | KW_X | KW_Y | KW_Z | KW_A | KW_R | KW_DEF_X | KW_DEF_Y	Expression_PredefinedName
Primary ::= INTEGER_LITERAL	Expression_IntLit
Primary LPAREN Expression RPAREN	
Primary ::= FunctionApplication	
Primary ::= BOOLEAN_LITERAL	Expression_BooleanLit
IdentOrPixelSelectorExpression::= IDENTIFIER LSQUARE Selector RSQUARE	Expression_PixelSelector
IdentOrPixelSelectorExpression::= IDENTIFIER	Expression_Ident
Lhs::= IDENTIFIER (LSQUARE LhsSelector RSQUARE | ε)	LHS
FunctionApplication ::= FunctionName LPAREN Expression RPAREN	Expression_FunctionAppWithExprArg
FunctionApplication ::= FunctionName LSQUARE Selector RSQUARE	Expression_FunctionAppWithIndexArg
FunctionName ::= KW_sin | KW_cos | KW_atan | KW_abs | KW_cart_x | KW_cart_y | KW_polar_a | KW_polar_r	
LhsSelector ::= LSQUARE (XySelector | RaSelector) RSQUARE	
XySelector ::= KW_x COMMA KW_y	Index
RaSelector ::= KW_r COMMA KW_A	Index
Selector ::= Expression COMMA Expression	Index

- A starter implementation of ParserTest.java with a few test cases has been provided. You will need to rename your parser from Assignment 2 to Parser.java and modify it so that the parse method returns an instance of cop5556fa17.AST.Program. For example:

```
public Program parse() throws SyntaxException {
	Program p = program();
	matchEOF();
	return p;
}
```

  The expression method must return an Expression object.
- All of the classes for AST nodes have been provided in the jar file. You should not modify any of these classes in this assignment (although you will modify them later.) These classes include the boilerplate code needed to implement the visitor pattern, but that will not be needed until assignment 4, so ignore it for now.
- The abstract superclass of all of the abstract syntax tree nodes is ASTNode.java. It contains a single field Token firstToken which must be provided in the constructor. This is there to connect the AST with the program source for error messages later and should be the first Token in the construct being parsed.

Turn in a jar file containing your source code for Parser.java, Scanner.java, and ParserTest.java. Also include the source for the provided classes AST node classes so that your jar file is complete.

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_hw3.jar*

Additional requirements:

- Your parser should remain in package cop5556fa17(case sensitive) and the parse and expression methods must be public. The former should return a Program object, the latter an Expression object.
- Your code should not import any classes other than those from the standard Java distribution, Scanner.java, or the provided cop5556fa17.AST package
- All code, including the Scanner code and the Parser code you are using as a starting point must be your own work developed by you this semester.
- Your Parser should throw exceptions for exactly the same input as a correctly implemented SimpleParser from Assignment 2 would. An AST will only be returned for valid input.

Submission Checklist

See the checklist from Assignment 1.

Comments and suggestions:

Don't attempt to do this assignment before you have looked at the relevant lecture.

Spend some time understanding the structure of the provided code. What is the inheritance hierarchy? How does that relate to the syntax?

You will need to look inside each class in order to see which fields it contains and what the constructor expects. If a field is optional in the syntax and is not provided in the input, you should set the corresponding field in the AST node to null. (The exception is the list of statements and declarations in Program. If there are no statements of declarations, the list should be empty, but not null.)

Each class contains methods visit, hashCode equals, and toString. The latter 3 were generated by eclipse; the visit method was systematically constructed to support the visitor pattern. It may be useful for you to use some of these methods (like toString) but otherwise you can ignore them.