Assignment 3

Modify your parser from assignment 2 to generate an AST (abstract syntax tree) specified by the following abstract syntax. The translation schemes for a few cases have been given in blue. You will need to do the rest yourself.

Concrete Syntax	**Abstract Syntax**
Program ::= Identifier Block	Program ::= IDENTIFIER Block
Block ::= { ((Declaration | Statement) ;)* }	Block ::= (Declaration | Statement)*
Declaration ::= Type IDENTIFIER | image IDENTIFIER [Expression , Expression]	Declaration ::= Type IDENTIFIER (ε | Expression Expression)
Type ::= int | float | boolean | image | filename	Type ::= int | float | boolean | image | filename
Statement ::= StatementInput | StatementWrite | StatementAssignment | StatementWhile | StatementIf | StatementShow | StatementSleep	Statement ::= StatementInput | StatementWrite | StatementAssign | StatementWhile | StatementIf | StatementShow | StatementSleep
StatementInput ::= first = t; input name = IDENTIFIER from @ e = Expression return new StatementInput(first, name, e)	StatementInput ::= IDENTIFIER Expression
StatementWrite ::= write IDENTIFIER to IDENTIFIER	StatementWrite ::= IDENTIFIER IDENTIFIER
StatementAssignment ::= LHS := Expression	StatementAssign ::= LHS Expression
StatementWhile ::= while (Expression) Block	StatementWhile ::= Expression Block
StatementIf ::= if (Expression) Block	StatementIf ::= Expression Block
StatementShow ::= show Expression	StatementShow ::= Expression
StatementSleep ::= sleep Expression	StatementSleep ::= Expression
LHS ::= IDENTIFIER	LHSIdent ::= IDENTIFIER
LHS ::= IDENTIFIER PixelSelector	LHSPixel ::= IDENTIFIER PixelSelector
LHS ::= Color (IDENTIFIER PixelSelector)	LHSSample ::= IDENTIFIER PixelSelector Color
Color ::= red | green | blue | alpha	Color ::= red | green | blue | alpha
PixelSelector ::= first = t; [e0 = Expression , e1 = Expression] return new PixelSelector(first, e0,e1)	PixelSelector ::= Expression Expression
	Expression ::= ExpressionBinary | ExpressionConditional | ExpressionFunctionAppWithExpressionArg |

	ExpressinFunctionAppWithPixelArg | ExpressionPixel | ExpressionPixelConstructor | ExpressionPredefinedName | ExpressionUnary | ExpressionIdent | ExpressionIntegerLiteral | ExpressionBooleanLiteral | ExpressionFloatLiteral
Expression ::= OrExpression ? Expression : Expression | OrExpression	ExpressionConditional ::= Expression Expression Expression
OrExpression ::=first = t; e0 = AndExpression (op = | e1 = AndExpression e0 = new ExpressionBinary(first, e0,op,e1)) * return e0	ExpressionBinary ::= Expression op Expression
AndExpression ::= EqExpression (& EqExpression)*	ExpressionBinary ::= Expression op Expression
EqExpression ::= RelExpression ((== | !=) RelExpression)*	ExpressionBinary ::= Expression op Expression
RelExpression ::= AddExpression ((< | > | <= | >=) AddExpression)*	ExpressionBinary ::= Expression op Expression
AddExpression ::= MultExpression ((+ | -) MultExpression)*	ExpressionBinary ::= Expression op Expression
MultExpression := PowerExpression ((* | / | %) PowerExpression)*	ExpressionBinary ::= Expression op Expression
PowerExpression := UnaryExpression (** PowerExpression | ε)	ExpressionBinary ::= Expression op Expression
UnaryExpression ::= + UnaryExpression | - UnaryExpression | UnaryExpressionNotPlusMinus	ExpressionUnary ::= Op Expression
UnaryExpressionNotPlusMinus ::= ! UnaryExpression | Primary	ExpressionUnary ::= Op Expression
Primary ::= IDENTIFIER	ExpressionIdent
Primary ::= INTEGER_LITERAL	ExpressionIntegerLiteral
Primary ::= BOOLEAN_LITERAL	ExpressionBooleanLiteral
Primary ::=FLOAT_LITERAL	ExpressionFloatLiteral
Primary ::= (Expression) | FunctionApplication | PixelExpression | PredefinedName | PixelConstructor	
PixelConstructor ::= << Expression , Expression , Expression , Expression >>	ExpressionPixelConstructor ::= Expression Expression Expression Expression
PixelExpression ::= IDENTIFIER PixelSelector	ExpressionPixel ::= IDENTIFIER PixelSelector
FunctionApplication ::= FunctionName (Expression)	ExpressionFunctionAppWithExpressionArg ::= FunctionName Expression
FunctionApplication ::= FunctionName [Expression , Expression]	ExpressionFunctionAppWithPixel ::= FunctionName Expression Expression

PredefinedName ::= Z | default_height | default_width	ExpressionPredefinedName
FunctionName ::= sin | cos | atan | abs | log | cart_x | cart_y | polar_a | polar_r | int | float | width | height | Color	FunctionName ::= sin | cos | atan | abs | log | cart_x | cart_y | polar_a | polar_r | int | float | width | height | Color

- Rename your SimpleParser.java from Assignment 2 to Parser.java, and then modify it to implement the assignment. In particular, the parse method should return an instance of cop5556sp18.AST.Program.
- Code for all the of the AST nodes and an interface called ASTVisitor has been provided. Do NOT modify these classes for Assignment 3. (You will modify them in later assignments to add fields to hold attribute values.) These classes include code, such as the visit method, that provides the plumbing for the visitor pattern. You can ignore this for now—it will not be needed until Assignment 4.
- Some of the AST nodes have synthesized attributes, typically `name` or `value`, whose value is obtained from the Scanner via a Token.
- Each parser method returns a subclass of ASTNode. To reduce the amount of casting necessary, the declared return type of each parser method should be as specific as possible. For example, the return type of method expression() should be Expression, not ASTNode.
- A starter implementation of ParserTest.java with a few test cases has been provided.
- It is convenient for test cases to invoke some of the parser's methods directly. As an example, one of the methods in the provided ParserTest.java directly invokes expression(). To ensure that all of our test work with your parser, make sure that your Parser has the following methods (with the indicated case-sensitive name and return type and that they are package visible (i.e. not private):
  - Expression expression()
  - Statement statement()
  - Declaration declaration()
- The abstract superclass of all of the abstract syntax tree nodes is ASTNode.java. It contains a single field Token firstToken, which should contain the first token in the construct represented by a subclass. The purpose is to allow you to connect the AST nodes with the program source so that you can give good error message including the position of the error when these are detected while traversing the AST in future assignments. The easiest way to implement this is to simply save the current token at the beginning of every parser method and pass that saved token to the constructor of any node you instantiate in that method.

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 cop5556fa18(case sensitive)
- Your code should not import any classes other than those from the standard Java distribution, Scanner.java, or the provided cop5556fa18.AST package
- All code, including the Scanner code and the SimpleParser 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.
- It may be convenient during testing to call the routines corresponding to fragments of the grammar in Junit tests. An example is shown in ParserTest.java.
- 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.