Assignment 4

Due: 20 March at 11:59pm

Implement a LeBlanc-Cook symbol table by completing the given class. You should create unit tests for the given methods.

Create a Visitor class called TypeCheckVisitor by completing the given class. Your visitor should traverse the AST generated by your parser from Assignment 3 and perform type checking according to the rules described below. Your Visitor class will both decorate the tree and check conditions along the way. If a condition is violated, throw a TypeCheckException.

Global rules: no variable may be declared more than once in the same scope.
The name of the program is not a variable name—it is used to name the class implemented by the program but does not preclude the same name begin declared as a variable.

SymbolTable symbab = new SymbolTable()

Program ::= List<ParamDec> Block
ParamDec ::= type ident symtab.insert(ident.getText(), ParamDec);
Block ::= symtab.enterScope() List<Dec> List<Statement> symtab.leaveScope()
Dec ::= type ident symtab.insert(ident.getText(), Dec);
Statement ::= SleepStatement | WhileStatement | IfStatement | Chain
    | AssignmentStatement
SleepStatement ::= Expression condition: Expression.type==INTEGER
AssignmentStatement ::= IdentLValue Expression
        condition: IdentLValue.type== Expression.type
Chain ::= ChainElem | BinaryChain
ChainElem ::= IdentChain | FilterOpChain | FrameOpChain | ImageOpChain
IdentChain ::= ident
    IdentChain.type <- ident.type
    ident.type <- symtab.lookup(ident.getText()).getType()
FilterOpChain ::= filterOp Tuple
    condition: Tuple.length == 0
    FilterOpChain.type <- IMAGE
FrameOpChain ::= frameOp Tuple
    FrameOpChain.kind <- frameOp.kind
    if (FrameOP.isKind(KW_SHOW, KW_HIDE) {
       condition: Tuple.length == 0

```
        FrameOpChain.type <- NONE
    }
    else if (FrameOp.isKind(KW_XLOC, KW_YLOC){
        condition: Tuple.length == 0
        FrameOpChain.type <- INTEGER
    }
    else if(FrameOp.isKind(KW_MOVE){
     condition: Tuple.length == 2
    FrameOpChain.type <- NONE
    }
    else there is a bug in your parser

ImageOpChain ::= imageOp Tuple
    ImageOpChain.kind <- imageOp.kind
    if (imageOp.isKind(OP_WIDTH, OP_HEIGHT){
        condition:  Tuple.length == 0
        ImageOpChain.type <- INTEGER
    }
    else if (imageOP.isKind(KW_SCALE)){
        condition: Tuple.length==1
        ImageOpChain.type <- IMAGE
   }
BinaryChain ::= Chain (arrow | bararrow)  ChainElem
```

Legal combinations shown:

BinaryChain	Chain	op	ChainElem
type <-IMAGE	type =URL	arrow	type = IMAGE
type <-IMAGE	type = FILE	arrow	type = IMAGE
type <-INTEGER	type = FRAME	arrow	instanceof FrameOp & firstToken ∈ { KW_XLOC, KW_YLOC}
type <-FRAME	type = FRAME	arrow	instanceof FrameOp & firstToken ∈ { KW_SHOW, KW_HIDE, KW_MOVE}
type <-IMAGE	type = IMAGE	arrow	instanceof ImageOpChain) && firstToken ∈ { OP_WIDTH, OP_HEIGHT}
type <-FRAME	type = IMAGE	arrow	type = FRAME
type <-NONE	type = IMAGE	arrow	type = FILE
type <-IMAGE	type = IMAGE	arrow | barrow	instanceof FilterOpChain & firstToken ∈ {OP_GRAY, OP_BLUR, OP_CONVOLVE}
type <-IMAGE	type = IMAGE	arrow	instanceof ImageOpChain &

			firstToken ∈ {KW_SCALE}
type <-IMAGE	type = IMAGE	arrow	instanceof IdentChain

WhileStatement ::= Expression Block
condition: Expression.type = Boolean

IfStatement ::= Expression Block
condition: Expression.type = Boolean

Expression ::= IdentExpression | IntLitExpression | BooleanLitExpression|
ConstantExpression | BinaryExpression
IdentExpression ::= ident
condition: ident has been declared and is visible in the current scope
IdentExpression.type <- ident.type
IdentExpression.dec <- Dec of ident
IdentLValue ::= ident
condition: ident has been declared and is visible in the current scope
IdentLValue.dec <- Dec of ident
IntLitExpression ::= intLit
IntLitExpression.type <- INTEGER
BooleanLitExpression ::= booleanLiteral
BooleanLitExpression.type <- BOOLEAN
ConstantExpression ::= screenWidth | screenHeight
ConstantExpression.type <- INTEGER
BinaryExpression ::= Expression op Expression

Legal combinations shown:

BinaryExpression.type	Expression0.type	op	Expression1.type
INTEGER	INTEGER	PLUS, MINUS	INTEGER
IMAGE	IMAGE	PLUS, MINUS	IMAGE
INTEGER	INTEGER	TIMES,DIV	INTEGER
IMAGE	INTEGER	TIMES	IMAGE
IMAGE	IMAGE	TIMES	INTEGER
BOOLEAN	INTEGER	LT,GT,LE,GE	INTEGER
BOOLEAN	BOOLEAN	LT,GT,LE,GE	BOOLEAN
BOOLEAN		EQUAL, NOTEQUAL	condition: Expression0.type = Expression1.type

Tuple ::= List<Expression>

condition: for all expression in List<Expression>: Expression.type = INTEGER

op ::= relOp | weakOp | strongOp

type ::= integer | image | frame | file | boolean | url

You will need to add an attribute along with get and set methods to some of the AST classes to record the type. In particular, add a TypeName field to Expression, Dec, and Chain. TypeName is defined in class cop5556sp17.AST and provided for you. Note that it is inherited by all of the subclasses of the given classes. If your Parser and Scanner have been implemented correctly, you shouldn't need to modify them.

A couple of junit tests have been provided in TypeCheckVisitorTest.java. You need to add more. As before you can work incrementally by calling the parser with a different method than parse.

Turn in a jar file containing your source code ASTVisitor.java, SymbolTable.java, Parser.java, Scanner.java, all of the AST nodes, Type.java, and your ASTVisitorTest.java.

Your ASTVisitorTest.java will not be graded, but may be looked at in case of academic honesty issues. We will subject your classes to our set of junit 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_hw4.jar

Additional requirements:

- Your code must remain package cop5556sp17(case sensitive): do not create additional packages.
- The provided class Type is in package cop5556sp17.AST. As before, all of the classes in this package should be in a folder called AST inside the cop5556sp17 folder
- Names (of classes, method, variables, etc.) in the provided AST classes must not be changed.
- Unless otherwise specified, your code should not import any classes other than those from the standard Java distribution or those that are part of the project.

Submission Checklist

See the checklist from Assignment 1.

Comments and suggestions:

- Work incrementally, starting with small constructs and moving to bigger ones, and adding tests each time. As in previous assignments, you can call the routines corresponding to fragments of the grammar in Junit tests.
- You can use the firstToken field in the case of errors to provide