In this assignment, we will implement code generation for part of the language. The abstract syntax shown below has been modified to indicate which part of the language we need to implement now and how they map into JVM elements.

Program : := Name List<ParamDec> Block

class Name implements Runnable{

variables declared in List<ParamDec> are instance variables of the class

public Name(String[] args){

initialize instance variables with values from args.

}

public static void main(String[] args){

Name instance = new Name(args);

instance.run();

}

public void run(){

declarations and statements from block

}

}

Param

in arg array

Block : := List<Dec> List<Statement>

Decs are local variables in outer

Statements are executed in run

Must label beginning and end of scope, and keep track of local variables, their slot in the local variable array, and their range of visibility.

If a statement was a BinaryChain, it will have left a value on top of the stack. Check for this and pop it if necessary.

Dec : := type ident

Assign a slot in the local variable array to this variable and save it in the new slot attribute in the Dec class.

frame maps to cop5556sp17.PLPRuntimeFrame

image maps to java.awt.image.BufferedImage

Statement : := SleepStatement | WhileStatement | IfStatement | Chain | AssignmentStatement

SleepStatement : := Expression

invoke java/lang/Thread/sleep.

Hint: You will need to change the integer expression to a long with "I2L"

AssignmentStatement ::= IdentLValue Expression

store value of Expression into location indicated by IdentLValue

- if the type of elements is image, this should copy the image.
    - use PLPRuntimeImageOps.copyImage

IMPORTANT:

insert the following statement into your code for an Assignment Statement

after value of expression is put on top of stack and before it is written into the

IdentLValue

```
CodeGenUtils.genPrintTOS(GRADE, mv,assignStatement.getE().getType());
```

Chain ::= ChainElem | BinaryChain

ChainElem ::= IdentChain | FilterOpChain | FrameOpChain | ImageOpChain

IdentChain ::= ident

- Han d whether it is on the l

    - If on the left side, load its val stack.
    - If this IdentChain is the right ion,
        - store the item on top of the stack into a variable (if INTEGER or IMAGE),
        - or write to file (if FILE),
        - or set as the image in the frame (if FRAME).

FilterOpChain ::= filterOp Tuple

- Assume that a reference to a BufferedImage is on top of the stack.
- Generate code to nvoke the appropriate method from PLPRuntimeFilterOps.

FrameOpChain ::= frameOp Tuple

- Assume that a reference to a PLPRuntimeFrame is on top of the stack.
- Visit the tuple elements to generate code to leave their values on top of the stack.
- Generate code to invokethe appropriate method from PLPRuntimeFrame.

ImageOpChain ::= imageOp Tuple

- Assume that a reference to a BufferedImage is on top of the stack.
- Visit the tuple elements to generate code to leave their values on top of the stack.
- Generate code to invoke the appropriate method from PLPRuntimeImageOps or PLPRuntimeImageIO .

BinaryChain ::= Chain (arrow | bararrow) ChainElem

- Visit the left expression.
    - If the left Chain is a URL, generate code to invoke PLPRuntimeImageIO.readFromURL and leave a reference to a BufferedImage object on top of the stack.
    - If the left expression is a File, generate code to invoke PLPRuntimeImageIO.readFromFile and leave a reference to a BufferedImage object on top of the stack.
    - Otherwise, generate code to leave the left object on top of the stack.
- Visit
- Hint: of a BinaryChain, in one the action is load, the other is store. You need to figure out a way to communicate to the IdentChain w
- Hint: although some combinations easiest to ex all binary chain instances leave som ck. In many cases, this value will be consumed by a parent. At the top level it should be popped.

WhileStatement ::= Expression Block

```
        goto GUARD
BODY    Block
GUARD   Expression
        IFNE  BODY
```

IfStatement ::= Expression Block

```
        Expression
        IFEQ AFTER

        Block

AFTER …
```

Expression ::= IdentExpression | IntLitExpression | BooleanLitExpression | ConstantExpression | BinaryExpression

always generate code to leave value of expression on top of stack.

IdentExpression ::= ident

load value of variable (this could be a field or a local var)

IdentLValue ::= ident

store value on top of stack to this variable (which could be a field or local var)

IntLitExpression ::= intLit

load constant

BooleanLitExpression ::= booleanLiteral

load constant

ConstantExpression ::= screenWidth | screenHeight

- Generate code to invoke PLPRuntimeFrame.getScreenWidth or PLPRuntimeFrame.getScreenHeight.

BinaryExpression ::= Expression op Expression

Visit children to generate code to leave values of arguments on stack.

perform operation, leaving result on top of the stack. Expressions should

- e structure of the AST.
- Visit children to generate code t ents on stack.
- Generate code to perform opera p of the stack.
- New in Assignment 6: methods to add two images, subtract two images, etc. Routines are provided in PLPRuntimeImageOps.
- New in assignment 6: implement &, |, and %.
- Expressions should be evaluated from left to write consistent with the structure of the AST.
- You may need to modify your TypeCheckVisitor t

Tuple ::= List<Expression>

- Visit expressions to generate code to leave values on top of the stack

op ::= relOp | weakOp | strongOp

implement operators and %

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

Corrections to TypeCheckVisitor (bold is new)

Modifications to the type rules for BinaryChain

type <-IMAGE	type = IMAGE	arrow	instanceof IdentChain & **IdentChain.type = IMAGE**
type <- INTEGER	**type = INTEGER**	**arrow**	**instance of IdentChain & IdentChain.type = INTEGER**

Provided Code:

- Compiler.java
  - Standalone program to read a source file, generate code and write the class file.
- PLPRuntimeFilterOps.java
  - provides method implementing the gray, blur, and convolve ops
- PLPRuntimeFrame
  - provides met is is a subclass of javax.swing.JFr
- PLPRuntim
  - provides methods to read and write images
- PLPRuntimeImageOps.java
  - provides methods to implement operations o
- PLPRuntimeLog.java
  - collects a trace of method calls in the PLPRuntime* classes.
  - Used for grading and debugging
- CodeGenUtils.java
  - updated version (added one method)

Note that all classes whose names start with PLPRuntime are needed at runtime to execute the programs in our language. This differs from asm, which is used during compilation.

Testing

Add these two methods to your junit test program to show the trace. The way output is handled has been changed slightly from before. Now it just writes everything to a StringBuffer which you can convert to a String and print. To see the output, add the

following to your JUnit test class. initLog and printLog will be run before and after every test, respectively.

```
@Before
public void initLog(){
if (devel || grade) PLPRuntimeLog.initLog();
}
@After
public void printLog(){
System.out.println(PLPRuntimeLog.getString());
}
```

-

Turn in: A jar file containing all files (with the exception of the asm packages) required to run the compiler as a standalone program and as called from a Junit test suite. Turn in your Junit test along with any files needed to execute your tests (with an appropriate directory structure)

Do not change the provided code.