Assignment 5

Implement a CodeGenerator class to traverse the decorated AST and generate code for part of our language. The abstract syntax has been annotated to indicate how our language maps into JVM elements and how to generate code. The complete language specification is given below, but you do not need to implement the parts that are shaded or otherwise indicated that they are not required for Assignment 5. The rest of the language will be implemented in Assignment 6.

Abstract Syntax	**Code generation**
Program ::= IDENTIFIER Block	Generate code for a class whose name is given by the Program.name attribute Create the public static void main(String args) method method and add local variable args Block becomes the body of this class's public static void main(String args) method.
Block ::= (Declaration | Statement)*	
Declaration ::= Type IDENTIFIER (ε | Expression_0 Expression_1)	Add a local variable with the Declaration's name and type to the class. Add an attribute to the Declaration for the assigned local variable slot number. If the type is image and Expression_0 and Expression_1 are not null, visit the Expressions to generate code to evaluate them and leave their value on the stack. Then generate code to instantiate an image (invoke RuntimeImageSupport.makeImage) If the type is image and Expression_0 and Expression_1 are null, generate code to load the predefined values defaultWidth and defaultHeight on the stack and instantiate an image. If the type is image, generate code to store the instantiated image in the variable. Variables of other types are left uninitialized.
Type ::= int | float | boolean | image | filename	<table><tr><td>Type</td><td>Java</td></tr></table>

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>JVM type</td></tr>
<tr><td>int</td><td>int
I</td></tr>
<tr><td>float</td><td>float
F</td></tr>
<tr><td>boolean</td><td>boolean
Z</td></tr>
<tr><td>image</td><td>java.awt.image.BufferedImage
Ljava/awt/image/BufferedImage;</td></tr>
<tr><td>filename</td><td>java.lang.String
Ljava/lang/String;</td></tr>
</table>
</td></tr>
<tr><td></td><td></td></tr>
<tr><td>Statement ::= StatementInput | StatementWrite | StatementAssign | StatementWhile | StatementIf | StatementShow | StatementSleep</td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td>StatementInput ::= IDENTIFIER Expression</td><td>Generate code to evaluate Expression.
This expression’s value is the index into the main method’s String[] args parameter that contains command line arguments.

Generate code to load the appropriate parameter from args.

Convert from String to the appropriate type and store in the variable.

If the the type is image, the parameter is a url or file, and the image should be read from its location (invoke RuntimeImageSupport.readImage). If a size was specified when the image variable was declared, the image should be resized to this value. Otherwise, the image retains its original size.</td></tr>
<tr><td>StatementWrite ::= $IDENTIFIER_0$ $IDENTIFIER_1$</td><td>Load the values of the variables on the stack and invoke RuntimeImageSupport.write to write the image to the file.</td></tr>
<tr><td>StatementAssign ::= LHS Expression</td><td>Visit the Expression to generate code to leave the expressions value on top of the stack.
Visit LHS to generate code to store the top of the stack into the indicated variable.</td></tr>
</table>

StatementWhile ::= Expression Block	Generate code code to implement a while loop.
StatementIf ::= Expression Block	Generate code code to implement an if statement.
StatementShow ::= Expression	Visit the Expression to generate code to evaluate the expression and and leave its value on top of the stack. If the expression type is int, boolean, or float, or filename, output the value to the console by invoking the println method of java/io/PrintStream. Note that this method is overloaded, you will need to call it with the correct type. If the expression type is image, display the image using the RuntimeImageSupport.makeFrame method. **IMPORTANT: For all types,, the CodeGenUtils.genLogTOS(GRADE, mv, type); method should be called before the top of the stack is consumed. The log will be used for grading, so it is crucial that this be done.**
StatementSleep ::= Expression	The value of the expression is the number of msecs that the program should sleep. Use java.lang.Thread.sleep.
LHSIdent ::= IDENTIFIER	Generate code to store the value already on top of the stack in the corresponding variable If the type is image, the value on top of the stack is actually a reference. Instead of copying the reference, a copy of the image should be created and the reference to the copy stored. Use RuntimeImageSupport.deepCopy to copy the image.
LHSPixel ::= IDENTIFIER PixelSelector	Generate code to store the int value already on top of the stack into the indicated pixel. After loading image and pixel location, invoke RuntimeImageSupport.setPixel.
LHSSample ::= IDENTIFIER PixelSelector Color	Generate code to store the int value already on top of the stack into the indicated sample. After loading image, pixel location, and color, invoke RuntimeImageSupport.updatePixelColor.
Color ::= red | green | blue | alpha	Represent colors with the constants defined in RuntimePixelOps
PixelSelector ::= Expression_0 Expression_1	Visit the Expressions to generate code to evaluate the expressions and leave the two values on top of the stack.
Expression ::= ExpressionBinary | ExpressionConditional | ExpressionFunctionAppWithExpressionArg | ExpressinFunctionAppWithPixelArg | ExpressionPixel | ExpressionPixelConstructor | ExpressionPredefinedName | ExpressionUnary | ExpressionIdent |	Always handle Expressions by generating code to evaluate the expression and leave its value on top of the stack. Other than a few hints and clarifications, how to do this is left for you to figure out. It may be useful for you to write little programs in Java with equivalent semantics and look at the code generated

ExpressionIntegerLiteral | ExpressionBooleanLiteral | ExpressionFloatLiteral	by the Java compiler. You may also use the asmifier eclipse plugin, or command line tool in class org.objectweb.asm.util.asmifier to print the ASM code that would generate a given class.
ExpressionConditional ::= $Expression_0$ $Expression_1$ $Expression_2$	
ExpressionBinary ::= $Expression_0$ op $Expression_1$	**Implement the binary operators in our language.** **For assignment 5, you may omit the relational operators--they will be required in assignment 6.** Hint: Implement POWER by converting both arguments to double, invoking java/lang/Math.pow, and converting the result back to the appropriate type. The others should be straightforward.
ExpressionUnary ::= Op Expression	Implement all of the unary operators in our language. Operator ! applied to an integer, should flip all the bits, including the sign bit.
ExpressionIdent	
ExpressionIntegerLiteral	
ExpressionBooleanLiteral	
ExpressionFloatLiteral	
ExpressionPixelConstructor ::= $Expression_{alpha}$ $Expression_{red}$ $Expression_{green}$ $Expression_{blue}$	Visit the Expressions to generate code to evaluate the four expressions and leave the four values on the stack.
ExpressionPixel ::= IDENTIFIER PixelSelector	Use method RuntimeImageSupport.getPixel
ExpressionFunctionAppWithExpressionArg ::= FunctionName Expression	For sin, cos, atan, log, abs: use functions in java.lang.Math. Some of these functions expect a double argument and return a double value, so you will need to cast to and from float. You may find it easier to write a wrapper routine in Java to do this and invoke your function instead of invoking the java.lang.Math function directly.

	function int converts a float to an int (JVM instruction F2I) or does nothing if the type is already int. function float converts an int to float(JVM instruction I2F) or does nothing if the type is already float.

ExpressionFunctionAppWithPixel ::= FunctionName $Expression_0$ $Expression_1$	cart_x and cart_y convert polar coordinates to cartesian:
	cart_x(r, theta) = r * Math.cos(theta)
	cart_y(r, theta) = r * Math.sin(theta)
	
	polar_r and polar_a convert Cartesian coordinates to polar (angle in radians)
	polar_a(x,y) = Math.atan2(y, x);
	polar_r(x,y) = Math.hypot(x,y);

ExpressionPredefinedName	Z = 255. default_width and default_height get their values from parameters passed to the CodeGenerator contructor.
FunctionName ::= sin | cos | atan | abs | log | cart_x | cart_y | polar_a | polar_r | int | float | width | height | Color	

You will need to add a getter method to Expression.java

```
public Type getType() {
        return type;
}
```

The provided CodeGenerator.java will generate code for the given test cases in TestCodeGen.java. You will need to extend the functionality. This may require refactoring the provided code in some cases.

As always, work incrementally. Create a new test program in a test case, then extend CodeGenerator to implement it. I suggest that you start with a simple program that can display a float value.

There are methods in CodeGenUtils that generate code to print or log the top of the stack (without consuming it) and print or log a string. These may be useful for debugging. If you use them that way, they should be turned on with DEVEL. You can see examples that use GRADE in visitProgram and visitStatementShow. In our test script, GRADE will be true, and DEVEL will be false. If you use these methods with GRADE in a way other than exactly as specified, the test cases will fail.

Turn in a jar file containing your source code for CodeGenerator.java, CodeGenTest.java, TypeChecker.java, Parser.java, Scanner.java, all of the AST classes, Types.java, TypeCheckerTest.java, RuntimeLog.java, RuntimeImageSupport.java, RuntimePixelOps.java, CodeGenUtils.java, and any classes that you have added.

Your CodeGenTest will not be graded, but may be looked at in case of academic honesty issues. We will subject your submission 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_hw5.jar***

Comments and Suggestions

- Remember that when you submit your assignment, you are attesting that have neither given nor received inappropriate help on the assignment. In this course, all assignments must be your own individual work, including the previous assignments after they have been graded.
- Work incrementally.
- See the troubleshooting tips in the JVM and ASM and CodeGeneration lectures.
- If you are getting errors in visitMaxs, you may want to try temporarily replacing the parameter in the ClassWriter constructor to 0. The result will not be a well-formed classfile, but you can at least see which instructions have been generated.