Code generation for Assignment 5

Program ::= Type Ident NameDef* Block	Generate and return a string containing a valid Java class
	
	public class NAME {
	public static TYPE apply(PARAMS) {
	BLOCK
	}
	where
	• NAME is from the Ident,
	• TYPE is the Java type corresponding to Type
	• PARAMS are from NameDef* and separated with a comma
	• BLOCK contains the declarations and statements in Block
	
	Depending on the contents of Block, you may need some import statements as well.

Block ::= DecList StatementList	visit children
DecList ::= Declaration*	visit each declaration, terminate with semicolon
StatementList ::= Statement *	visit each statement, terminate with semicolon
Declaration::= NameDef (Expr | ε)	visit nameDef if there is an Expr, = visit Expr
NameDef ::= Type Ident (Dimension | ε)	TYPE NAME where TYPE is the java type corresponding to Type and NAME is the name of the Ident. (Do not implement dimensions in assignment 5)
Expr ::= ConditionalExpr | BinaryExpr | UnaryExpr | StringLitExpr | IdentExpr | NumLitExpr | ZExpr | RandExpr | UnaryExprPostFix | PixelFuncExpr |PredeclaredVarExpr	
UnaryExprPostfix::=PrimaryExpr (PixelSelector | ε) (ChannelSelector | ε)	Do not implement in assignment 5
PixelFunctionExpr ::= (x_cart | y_cart | a_polar | r_polar) PixelSelector	Do not implement in assignment 5
PredeclaredVarExpr ::= x | y | a | r	Do not implement in assignment 5

ConditionalExpr ::= $Expr_0$ $Expr_1$ $Expr_2$	Implement corresponding Java code, something like (EXPR0 ? EXPR1 : EXPR2) where EXPR0, EXPR1, and EXPR2 are obtained by visiting the corresponding expression. Note that you may need to do more than simply visit EXPR0 since in our language we are using ints, and java expects a Boolean. You will need to figure out how to do this. It is fine to do this in a uniform way, even if it is suboptimal. The same solution can be used in WhileStatement
BinaryExpr ::= $Expr_0$ (+| - | * | / | % | < | > | <= | >= |== | | | || & | && | **) $Expr_1$	(EXPR0 OP EXPR1) where EXPR0, EXPR1 are obtained by visiting the corresponding expression, and OP is the corresponding java binary operator. In our language, Boolean values are represented as ints were 0 = false. Something like a<b will be Boolean in Java, but should be 0 or 1 in our language. (How to handle this is left for you to figure out.) The exception to the above is ** which can be implemented using java.lang.Math.pow. (Details left for you to figure out)
UnaryExpr ::= (! | - | sin | cos | atan) Expr	Do not implement in assignment 5
StringLitExpr	Generate the Java string literal corresponding to this one. (You may ignore escape sequences)
IdentExpr	Generate name
ZExpr	This is a constant with value 255
RandExpr	Generate code for a random int in [0,256) using Math.floor(Math.random() * 256) This will require an import statement.
PredefinedVarExpr	Do not implement in assignment 5
ChannelSelector ::= red | grn | blu	Do not implement in assignment 5
PixelSelector ::= $Expr_0$ $Expr_1$	Do not implement in assignment 5
ExpandedPixelExpr ::= $Expr_0$ $Expr_1$ $Expr_2$	Do not implement in assignment 5
Dimension ::= $Expr_0$ $Expr_1$	Do not implement in assignment 5
LValue ::= Ident (PixelSelector | ε) (ChannelSelector | ε)	For assignment 5, only handle the case where there is no PixelSelector and no ChannelSelector. Generate name of Ident

Statement::= AssignmentStatement | WriteStatement | WhileStatement | ReturnStatement	
AssignmentStatement ::= LValue Expr	LVALUE = EXPR where LVALUE is obtained by visiting LValue, and EXPR is obtained by visiting Expr
WriteExpr ::= Expr	Generate code to invoke ConsoleIO.write(EXPR) where EXPR is obtained by visiting Expr. This will also require an import statement

WhileStatement ::= Expr Block	while (EXPR) {
	  BLOCK
	}
	where EXPR and BLOCK are obtained by visiting the corresponding children.
	
	Note that you may need to do more than simply visit EXPR since in our language we are using ints, and java expects a Boolean. You will need to figure out how to do this. It is fine to do this in a uniform way, even if it is suboptimal. The same solution can be used in ConditionalExpr
	
	If your input program has redeclared an identifier in the inner scope, a straightforward translation into Java will not work. To get full credit, you will need to handle this case. One easy way to do it is to give each variable a unique name in the generated java code. You may find it easiest to do this in the type checking pass.

ReturnStatement ::= Expr	return EXPR where EXPR is obtained by visiting the corresponding child