Code generation for Assignment 6

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 If the type of NameDef is a pixel, it has Java type int. If the type of NameDef is an image, it has Java type BufferedImage. Initializers when type is image: It is easier to handle dimensions here rather than in child NameDef. There are several cases: If NameDef.dimension == null There must be an initializer from which the size can be determined. Initializer has type string. Assume this is a url or filename. Use FileURLIO.readImage. *see cg20*. Initializer has type image. Use ImageOps.cloneImage. *see cg11* If NameDef.dimension != null, an image of this size is created. (Default pixel values are ff000000). If no initializer, use ImageOps.makeImage. *see cg10a* If string initializer, use readImage overload with size parameters. *see cg11b* If image initializer, use copyAndResize *see cg11a* Initializers when var is pixel Java type is int, rhs is pixel, use PixelOps.pack. *see cg11c*
NameDef ::= Type Ident (Dimension | ε)	TYPE NAME where TYPE is the java type corresponding to Type and NAME is the name of the Ident.

Expr ::= ConditionalExpr	BinaryExpr	UnaryExpr	StringLitExpr	IdentExpr	NumLitExpr	ZExpr	RandExpr	UnaryExprPostFix	PixelFuncExpr |PredeclaredVarExpr	
UnaryExprPostfix::=PrimaryExpr (PixelSelector | ε) (ChannelSelector | ε)	If PrimaryExpr has type image. 3 Cases: PrimayExpr PixelSelector ε Use ImageOps.getRGB Example a[x,y] ImageOps.getRGB(a,x,y) *see test cg6_0* PrimaryExpr PixelSelector ChannelSelector Use PixelOps method to get color from pixel and ImageOps.getRGB Example: a[x,y]:red PixelOps.red(ImageOps.getRGB(a,x,y) *see test cg6_1* PrimaryExpr ChannelSelector Use ImageOps extract routine Example: a:red ImageOps.extractRed(a) *see test cg6_2* If PrimryExpr has type pixel PrimaryExpr ChannelSelector Use PixelOps red,grn, or blu Example: a:red PixelOps.red(a) *see test cg6_3*
PixelFunctionExpr ::= (x_cart | y_cart | a_polar | r_polar) PixelSelector	*We will not implement these*
PredeclaredVarExpr ::= x | y | a | r	Assume that x and y will only appear in a limited context. See below in description of assignment statement for explanation.

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) There are several cases for images and pixels $Expr_0$.type == IMAGE $Expr_1$.type == IMAGE OP ∈ {+,-,*,/,%). Use ImageOps.binaryImageImageOp *See test cg6_4* $Expr_0$.type == IMAGE $Expr_1$.type == INT OP ∈ {+,-,*,/,%). Use ImageOps.binaryImageScalarOp *See test cg6_5* $Expr_0$.type == PIXEL $Expr_1$.type == PIXEL OP ∈ {+,-,*,/,%). Use PixelOps.binaryImagePixelOp *See test cg6_6* You do not need to implement == or != for images or pixels.
UnaryExpr ::= (! | - | sin | cos | atan) Expr	Implement ! and – for int. Here !x can be computed using x==0 ? 1 : 0 You may omit sin, cos, and atan.
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	You do not need to do anything with a and r. All you need to do is generate the name 'x' or 'y'. You may assume that x and y only appear in the scope of an implicit loop over x and y as in im1[x,y] = im2[y,x]. or im1[x,y] = [x,0,0].
ChannelSelector ::= red | grn | blu	This enum is used in expressions and LValues.
PixelSelector ::= $Expr_0$ $Expr_1$	Generate comma separated code to evaluate the two expressions. (Visit children, adding comma in between) see test cg21.
ExpandedPixelExpr ::= $Expr_0$ $Expr_1$ $Expr_2$	Invoke PixelOps.pack on the values of the three expressions. *See test cg6_7*
Dimension ::= $Expr_0$ $Expr_1$	Generate comma separated code to evaluate the two expressions
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 Handle PixelSelector and ChannelSelector in parent AssignmentStatement where context is known.
Statement::= AssignmentStatement | WriteStatement | WhileStatement | ReturnStatement	
AssignmentStatement ::= LValue Expr	LVALUE = EXPR where LVALUE is obtained by visiting LValue, and EXPR is obtained by visiting Expr *See below for how to handle assignment statements involving images and pixels.*

WriteExpr ::= Expr	Generate code to invoke ConsoleIO.write(EXPR) where EXPR is obtained by visiting Expr. This will also require an import statement There is an implementation of ConsoleIO.write where EXPR is an Image, so this case can be handled uniformly. However, if the expr has type Pixel, then ConsoleIO.writePixel must be invoked instead.

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. This should work the same way for images and pixels as the other types you have already implemented.

Handling assignment statements.

Cases handled in Assignment 5 remain the same.

Variable type is pixel

Example: yellow = [Z,Z,0]
Use PixelOps.pack.
See cg26

Variable type is image, no pixel selector, no color channel

Right side is string.
Read image from url or file, but copy into lhs image, resizing source image.
Use readImage and copyInto.
See cg26a

Right side is image.
Copy image into destination, resizing.
Use copyInto.
See cg20

Right side is pixel
Set all pixels to given pixel value.
Use ImageOps.setAllPixels
See cg20a

Variable type is image with pixel selector, no color channel

Define implicit loop over all pixels in image with loop
For example

```
int hshift = 0.
int vshift = h/2.
c[x,y]= im0[x+hshift, y+vshift].
```

Translates to

```
int hshift=0;
int vshift=(h/2);

for (int y = 0; y != c.getHeight(); y++){
for (int x = 0; x != c.getWidth(); x++){
ImageOps.setRGB(c,x,y,
     ImageOps.getRGB(im0,(x+hshift0),(y+vshift0)));
```

```
}
```

Variable type is image with pixel selector and color channel

Define implicit loop over all pixels in image with loop as before, but use PixelOps.setRed, (or setGrn, setBlu)

Example

```
image[w,h] im = [0,0,0].
im[x,y]:grn = Z.
im[x,y]:blu = Z.
```

Translates to code similar to

```
BufferedImage im = ImageOps.makeImage(w, h);
ImageOps.setAllPixels(im, PixelOps.pack(0, 0, 0));

for (int y = 0; y != im.getHeight(); y++){
  for (int x = 0; x != im.getWidth(); x++){
    ImageOps.setRGB(im,x,y,
      PixelOps.setGrn(ImageOps.getRGB(im,x,y),255));
  }
}

for (int y = 0; y != im.getHeight(); y++){
  for (int x = 0; x != im.getWidth(); x++){
    ImageOps.setRGB(im,x,y,
      PixelOps.setBlu(ImageOps.getRGB(im,x,y),255));
  }
}
```