Abstract Syntax	Code generation
<Program> ::= IDENTIFIER <Block>	Create a class file for the program. The name is given by the identifier. **The visit method should return an array of bytes containing the class.**
<Block> ::= <DecOrCommand> *	
<Declaration> ::= <Type> IDENTIFIER	Add a static field with name given by the indicator and appropriate type to the classfile. **If a Compound Type, instantiate an empty HashMap when declared.**
<SimpleType> ::= int | boolean | string (use enum in Kind class)	These types are represented by java int, boolean, and java.lang.String respectively.
<CompoundType> ::=<SimpleType> <Type>	**The type is represented as a java/lang/HashMap instance**
<AssignExprCommand> ::= <LValue><Expression>	Generate code to evaluate the expression and store the results in the variable indicated by the LValue. **If the LValue is an <ExprLValue> First visit the LValue to leave the address of the map and the LValue's expression (the key) on top of the stack. Then visit the <Expression> to generate code to leave its value on top of the stack (the value). Then invoke the HashMap put method to add the (key,value) pair to the map.** **For example, if your program contains x[k] = e, then this would correspond to the java method invocation x.put(k,e).**
<AssignPairListCommand> ::= <LValue><PairList>	**Visit the <LValue> to leave the address of the HashMap on top of the stack. Visit the <PairList> to leave the address of new HashMap containing all the pairs in the <PairList> on top of the stack. Invoke putAll to move the pairs from the temporary Map to the one used in the program. Note that the semantics of this assignment replace all existing entries in the map with the ones in the <PairList> so you also need to clear the map before putAll.** **i.e.** **if m contains {[2,"hello"]} and you do** **m = {[3,"world"]}** **then after the assignment m contains only {[3,"world]}.**
<PrintCommand> ::= <Expression>	Generate code to invoke System.out.print with the value of the Expression as a parameter. Note that this method is overloaded—you must invoke the correct one. **If the**

	expression is a map, the type of the argument is java.lang.Object.
<PrintlnCommand> ::= <Expression>	Generate code to invoke System.out.println with the value of the Expression as a parameter. Note that this method is overloaded. You must invoke the correct one. **If the expression is a map, the type of the argument is java.lang.Object.**
<DoCommand> ::= <Expression> <Block>	Generate as if for a Java while loop.
<DoEachCommand> ::= <LValue> IDENTIFIER$_0$ IDENTIFIER$_1$ <Block>	**See below**
<IfCommand> ::= <Expression> <Block>	Generate code as if for a Java if statement
<IfElseCommand> ::= <Expression><Block> <Block>	Generate code as if for a Java if-else statement
<SimpleLValue> ::= IDENTIFIER	When this appears on the lhs of an assignment, use as target of the assignment
<ExprLValue> ::= IDENTIFIER <Expression>	**When this appears on the lhs side of an assignment, visit the <ExprLValue> to generate code to push the address of the map and the value of the exprLValue.expression (the key) on top of the stack.** **When an <ExprLValue> appears on the rhs of an assignment, as in y = x[k], then this corresponds to y = x.get(k).**
<Pair> ::= <Expression> <Expression>	**Generate code to leave the value of the two expressions on top of the stack.**
<PairList> ::= <Pair> *	**Create a new temporary HashMap. For each pair in the list, visit the pair to leave a (key,value) pair on top of the stack and add it to the temporary HashMap. Hint: you might find the instruction**
<BinaryOpExpression>::= <Expression$_0$> op <Expression$_1$> (use enum in Kind class for Op)	Generate code to leave the value of the expression on top of the stack. See the table below for more details. **See below**
<LValueExpression> := <LValue>	Generate code to leave the value of the expression on top of the stack
<IntegerLiteralExpression> ::= INTEGER_LITERAL	Generate code to leave the value of the literal on top of the stack.
<BooleanLiteralExpression> ::= BOOLEAN_LITERAL	Generate code to leave the value of the literal on top of the stack.
<StringLiteralExpression> ::= STRING_LITERAL	Generate code to leave the value of the literal on top of the stack.
<Expression>	Generally, for all expressions, generate code to leave the value of the expression on top of the stack.
<UnaryOpExpression> ::= op <Expression>	if op is -, negate the value on top of the stack, if op is !, logically negate the value on top of the stack.

(use enum in Kind class for Op)	

BinaryOpExpressions

operator\type of arguments	int	boolean	string	map
+	int addition	not defined	string concatenation	**Create a new temporary hashmap that contains the union of the two maps. If a key is duplicated, the value of the second entry is used. You may use Runtime.plus to implement this.**
-	int subtraction	not defined	not defined	**Create a new temporary hashmap that contains the entries in the first minus the duplicate entries in the second. You may use Runtime.minus to implement this.**
*	int multiplication	not defined	not defined	**Create a new temporary hasmap that contains the intersection of the two maps. You may use Runtime.times to implement this.**
/	int division	not defined	not defined	not defined
&	not defined	conditional and	not defined	not defined
|	not defined	conditional or	not defined	not defined
==	the two ints have the same value	the two booleans have the same value	the two strings contain the same characters	**returns true if the two maps have the same entries. You may use Runtime.equals to implement this.**
!=	the two ints have different values	the two booleans have different values	the two strings have different values	**returns true if the two maps do not have the same entries. You may use Runtime.not_equals to implement this.**
<	as expected	as expected (false < true)	Not defined	**You may use Runtime.less_than to**

				implement this.
>	as expected	as expected	Not defined	**You may use Runtime.greater_than to implement this.**
<=	as expected	as expected	Prefix or equals relation (invoke the startsWith method of the String class)	**You may use Runtime.at_most to implement this.**
>=	as expected	as expected	Not defined	**You may use Runtime.at_least to implement this.**

I have given you a class called Runtime which you may optionally use to implement the binary expressions involving maps. Invoke the corresponding static method instead of generating the code directly. *You may find it convenient to add other things to this class. You may do so, but do not change the name or package of the class.*

If only one of the arguments is a String, the other one is coerced into a string, then String concatenation is performed.

The startsWith and the various valueOf methods in the java.lang.String class may be useful.

Boxing, unboxing, and checkcast

The HashMap implementation assumes that all entries are of type java/lang/Object. This means that elements of primitive type must be boxed and unboxed when added to or removed from the map. Also, when an object is removed, a checkcast instruction must be invoked to verify the type. In some cases, the return may be a map that hasn't been instantiated. In my implementation, I used the following methods, which are designed to be able to be put anywhere that something is added to a map with put, or removed with get. You probably won't be able to copy and paste these but they should give you an idea.

```
	// if primitive type, generates code to box
	void box(MethodVisitor mv, Type t) {
		if (t.equals(Type._integer)) {
			mv.visitMethodInsn(INVOKESTATIC, "java/lang/Integer", "valueOf",
					"(I)Ljava/lang/Integer;");
		}
```

```
            if (t.equals(Type._boolean)) {
                  mv.visitMethodInsn(INVOKESTATIC, "java/lang/Boolean", "valueOf",
                              "(Z)Ljava/lang/Boolean;");
            }
      }

      // if primitive type, generates code to unbox, also inserts checkcast
      // instructions for all types
      // if returned type is a HashMap that is null, instantiates the hashmap
      void unbox(MethodVisitor mv, Type t) {
            if (t.equals(Type._integer)) {
                  mv.visitTypeInsn(CHECKCAST, "java/lang/Integer");
                  mv.visitMethodInsn(INVOKEVIRTUAL, "java/lang/Integer",
"intValue",
                              "()I");
            }
            if (t.equals(Type._boolean)) {
                  mv.visitTypeInsn(CHECKCAST, "java/lang/Boolean");
                  mv.visitMethodInsn(INVOKEVIRTUAL, "java/lang/Boolean",
                              "booleanValue", "()Z");
            }
            if (t.equals(Type._string)) {
                  mv.visitTypeInsn(CHECKCAST, "java/lang/String");
            }
            if (t instanceof CompoundType) {
                  Label ubl = new Label();
                  mv.visitInsn(DUP);
                  mv.visitJumpInsn(IFNONNULL, ubl);
                  mv.visitInsn(POP);
                  instantiateHashMap(mv);
                  mv.visitLabel(ubl);
                  mv.visitTypeInsn(CHECKCAST, HASHMAPNAME);
            }
      }
```

The doeach command

1. **Load the address of the lValue map onto the stack. For simplicity, you may assume that this will be a SimpleLValue.**
2. **Get its entry set (HashMap.entrySet)**
3. **Get an iterator for the entry set (Set.iterator). A set is an interface, not a class, so use the INVOKEINTERFACE instruction**
4. **Now this looks like a do loop: the first time, jump to the guard**
5. **Now do the loop body code which means get the next element from the iterator (Iterator.next).**
6. **This returns an element of type Map$Entry. Add a checkcast instruction.**
7. **Get the key from the entry, unbox it, and store in the variable for the key**
8. **Get the value from the entry, unbox it, and store in the variable for the value.**

9. **Visit the block**
10. **Now you are at the code for the guard. Find out if there are more entries (Iterator.hasNext)**
11. **If so, jump to the top of the loop.**
12. **Note that my description does not include the instructions needed to manage the stack. You will need to pay attention to where you are consuming references to the iterator and also make sure that the stack is empty when you are finished.**

Examples

Here are some code examples. The source program and expected output are given as Java string literals.

```
                         "prog TestForEach2 " +
                    "int k; " +
                    "map[boolean, int]  v;  " +
                    "map[int, map[boolean, int]] m; " +
                    "map [boolean, int] m1; " +
                         "m1 = {[true,43],[false,95]}; " +
                    " m[100] = m1;    " +
                    "do m : [k,v] " +
                    "print k; " +
                    "boolean kb; " +
                    "int vi; " +
                    "do v : [kb,vi] " +
                    "print kb; " +
                    "print vi; " +
                    "od; " +
                    "od ; " +
                    "gorp";
             "100false95true43";
```

```
======================
```

```
                  "prog TestMap3 " +
                    "int k; " +
                    "map[boolean, int]  v;  " +
                    "map[int, map[boolean, int]] m; " +
                    "map [boolean, int] m1; " +
                         "m1 = {[true,43],[false,95]}; " +
                         "print m;" +
                    "gorp";
             "{}";
```

```
======================
                         "prog TestMap4 " +
                    "int k; " +
                    "map[boolean, int]  v;  " +
                    "map[int, map[boolean, int]] m; " +
                    "map [boolean, int] m1; " +
                         "m1 = {[true,43],[false,95]}; " +
                    " m[100] = m1;    " +
                         "print m;" +
                    "gorp";
            "{100={false=95, true=43}}";

======================

                         "prog TestMap2 " +
                    "int k; " +
                    "map[int,int] m0; "+
                          "map[int,int] m1; "  +
                     "m0 = {[0,100],[2,102], [3,103]}; " +
                          "m1 = {[0,100],[2,102], [3,103]}; " +
                          "print m0; "+
                          "print m1; " +
                    "gorp";
            "{0=100, 2=102, 3=103}{0=100, 2=102, 3=103}";
======================

                         "prog TestMapPlus " +
                    "int k; " +
                    "map[int,int] m0; "+
                          "map[int,int] m1; "  +
                     "m0 = {[0,100],[2,102], [3,103]}; " +
                          "m1 = {[0,100],[2,102], [3,103]}; " +
                     "m0 = m0 + m1; " +
                          "print m0; "+
                          "print m1; " +
                    "gorp";
            "{0=100, 2=102, 3=103}{0=100, 2=102, 3=103}";
=======================

                         "prog TestMapPlus2 " +
                    "int k; " +
                    "map[int,int] m0; "+
                          "map[int,int] m1; "  +
                     "m0 = {[0,100],[2,102], [3,103]}; " +
                          "m1 = {[0,100],[2,102], [3,104]}; " +
                     "m0 = m0 + m1; " +
                          "print m0; "+
                          "print m1; " +
                    "gorp";
            "{0=100, 2=102, 3=104}{0=100, 2=102, 3=104}";
==========================

                         "prog TestMapPlus3 " +
                    "int k; " +
                    "map[int,int] m0; "+
                          "map[int,int] m1; "  +
                                   "m0 = {[0,100],[2,102], [3,103]}; " +
                                        "m1 = {[5,100],[6,102], [7,104]}; " +
                                   "m0 = m0 + m1; " +
                                        "print m0; "+
                                        "print m1; " +
                                  "gorp";
                          "{0=100, 2=102, 3=103, 5=100, 6=102, 7=104}{5=100, 6=102, 7=104}";
============================

                                       "prog TestMapEq " +
                                  "int k; " +
                                  "boolean b;" +
                                  "map[int,int] m0; "+
                                        "map[int,int] m1; "  +
                                   "m0 = {[0,100],[2,102], [3,103]}; " +
                                        "m1 = {[0,100],[2,102], [3,104]}; " +
                                   "b = m0 == m1; " +
                                        "print m0; "+
                                        "print m1; " +
                                        "print b; " +
                                  "gorp";
                          "{0=100, 2=102, 3=103}{0=100, 2=102, 3=104}false";
===========================

                                       "prog TestMapEq1 " +
                                  "int k; " +
                                  "boolean b;" +
                                  "map[int,int] m0; "+
                                        "map[int,int] m1; "  +
                                   "m0 = {[0,100],[2,102], [3,103]}; " +
                                        "m1 = {[0,100],[2,102], [3,103]}; " +
                                   "b = m0 == m1; " +
                                        "print m0; "+
                                        "print m1; " +
                                        "print b; " +
                                  "gorp";
                          "{0=100, 2=102, 3=103}{0=100, 2=102, 3=103}true";
===========================

                                       "prog TestMapNeq " +
                                  "int k; " +
                                  "boolean b;" +
                                  "map[int,int] m0; "+
                                        "map[int,int] m1; "  +
                                   "m0 = {[0,100],[2,102], [3,103]}; " +
                                        "m1 = {[0,100],[2,102], [3,104]}; " +
                                   "b = m0 != m1; " +
                                        "print m0; "+
                                        "print m1; " +
                                        "print b; " +
                                  "gorp";
                          "{0=100, 2=102, 3=103}{0=100, 2=102, 3=104}true";
============================

                                       "prog TestMapNeq1 " +
                                  "int k; " +
                                  "boolean b;" +
                                  "map[int,int] m0; "+
                                        "map[int,int] m1; "  +
                                   "m0 = {[0,100],[2,102], [3,103]}; " +
                                        "m1 = {[0,100],[2,102], [3,103]}; " +
                                   "b = m0 != m1; " +
                                        "print m0; "+
                                        "print m1; " +
                                        "print b; " +
                                  "gorp";
                          "{0=100, 2=102, 3=103}{0=100, 2=102, 3=103}false";
```