Project 5 (Type Checking and Code Generation)

Goal

1. Implement type checking for the Java programming constructs that were introduced in *j--* as part of Project 3 (Parsing).
2. Implement JVM code generation for those Java programming constructs.

Download the Project Tests

Download and unzip the tests for this project under `$j/j--`.

Run the following command inside the `$j/j--` directory to compile the *j--* compiler with your changes:

```
>_ ~/workspace/j--
$ ant
```

To compile a *j--* program `project5/XYZ.java`, run the following command:

```
>_ ~/workspace/j--
$ bash ./bin/j-- project5/XYZ.java
```

Run the following command to run the *j--* program `XYZ.class`:

```
>_ ~/workspace/j--
$ java XYZ
```

Problem 1. (*Long and Double Basic Types*) Add support for `long` and `double` basic types.

Directions:

- Implement `analyze()` and `codegen()` in `JLiteralLong` and `JLiteralDouble`.
- Modify `JCastOp` and `Conversions`, and add new converters.
- Modify `partialCodegen()` in `JMethodDeclaration`.
- Modify `analyze()` in `JConstructorDeclaration`, `JMethodDeclaration`, and `JVariableDeclaration` to skip an offset for longs and doubles.
- Modify `codegen()` in `JReturnStatement`.
- Modify the 1-argument `codegen()` method and the `codegenStore()` method in `JVariable`.
- Modify 1-argument `codegen()`, `codegenLoadLhsRvalue`, and `codegenStore()` in `JArrayExpression`.
- Modify `codegen()` in `JArrayInitializer`.

Note: the programs below will not compile/run properly till you complete Problem 2 (Operators).

```
>_ ~/workspace/j--
$ java BasicTypes 1 -5 6 6
Roots of 1.0x^2 + -5.0x + 6.0 = 0: 3.0, 2.0
fibonacci(6) = 8
$ java Stats
Mean   = 5.5
Stddev = 2.8722813232690143
```

Problem 2. (*Operators*) Add support for the following operators. Note that parsing support for some of the operators was added to *j--* in Project 1.

`!=`	`/=`	`-=`	`*=`	`%=`	`>>=`	`>>>=`	`>=`
`<<=`	`<`	`^=`	`|=`	`||`	`&=`	`++`	`--`
`/`	`%`	`<<`	`>>`	`>>>`	`~`	`|`	`^`
`&`	`+`						

Directions:

- Modify `analyze()` in `JNegateOp` and `JUnaryPlusOp`; the operand can be an int, long, or double.
- Implement `analyze()` and `codegen()` in `JPostIncrementOp` and `JPreDecrementOp`; the operand must be an int.
- Implement `analyze()` and `codegen()` in `JLogicalOrOp` and `JNotEqualOp`.
- Implement `analyze()` and `codegen()` in `JGreaterEqualOp` and `JLessThanOp`; the operands can be an ints, longs, or doubles.
- Modify `analyze()` and `codegen()` in `JPlusOp`, `JSubtractOp`, `JMultiplyOp`, `JDivideOp`, and `JRemainderOp`; the operands can be an ints, longs, or doubles.
- Modify `analyze()` and `codegen()` in `JPlusAssignOp`; the operands can be an ints, longs, or doubles.
- Implement `analyze()` and `codegen()` in `JMinusAssignOp`, `JStarAssignOp`, `JDivAssignOp`, and `JRemAssignOp`; the operands can be an ints, longs, or doubles.
- Implement `analyze()` and `codegen()` in `JOrAssignOp`, `JAndAssignOp`, `JXorAssignOp`, `JALeftShiftAssignOp`, `JARightShiftAssignOp`, and `JLRightShiftAssignOp`; the operands must be ints.

```
>_ ~/workspace/j--
$ java Operators 23 3
true
7
4
12
0
0
0
false
0
true
3
3
true
3
3
2
2
0
16
1
1
-5
6
6
0
4
```

Problem 3. (*Conditional Expression*) Add support for conditional expression (`e1 ? e2 : e3`).

Directions:

- Analyze the condition and make sure it's a boolean.
- Analyze the consequent and alternate and make sure they have the same type.
- Set the type of the expression to that of the consequent (or alternate).
- Implement `codegen()`.

```
>_ ~/workspace/j--
$ java ConditionalExpression
Tails
$ java ConditionalExpression
Tails
$ java ConditionalExpression
Heads
```

Problem 4. (*Switch Statement*) Add support for a switch statement. Here's some code you may want to use to decide which instruction (`TABLESWITCH` or `LOOKUPSWITCH`) to emit:

```
long tableSpaceCost = 5 + hi - lo;
long tableTimeCost = 3;
long lookupSpaceCost = 3 + 2 * nLabels;
long lookupTimeCost = nLabels;
int opcode = nLabels > 0 && (tableSpaceCost + 3 * tableTimeCost <= lookupSpaceCost + 3 * lookupTimeCost) ?
                 TABLESWITCH : LOOKUPSWITCH;
```

Where `hi` is the highest case label value, `lo` is the lowest case label value, and `nlabels` are the total real case labels in the switch statement.

Directions:

- Analyze the condition and make sure it is an integer.
- Anayze the case expressions and make sure they are integer literals.
- Create a new `LocalContext` with `context` as the parent, and analyze the statements in each case group in the new context.
- In `codegen()` decide which instruction (`TABLESWITCH` or `LOOKUPSWITCH`) to emit using the above heuristic.
- Call the appropriate `CLEmitter` method to emit that instruction — you will first need to gather all the information that must be passed as arguments to the method.
- Generate code for the case group statements, adding labels at the appropriate places.
- Consult `$j/j--/tests/clemitter/GenTableSwitch.java` and `$j/j--/tests/clemitter/GenLookupSwitch.java` for more hints on codegen.

Note: the program below will not compile/run properly till you complete Problem 7 (Break Statement).

```
>_ ~/workspace/j--
$ java SwitchStatement
Queen of Hearts
$ java SwitchStatement
Jack of Spades
```

Problem 5. (*Do Statement*) Add support for a do-while statement.

Directions:

- Analyze the condition and make sure it's a boolean.
- Analyze the body.
- Implement `codegen()`.

```
>_ ~/workspace/j--
$ java DoStatement 100
5050
```

Problem 6. (*For Statement*) Add support for a for statement.

Directions:

- Create a new `LocalContext` with `context` as the parent.
- Analyze the init in the new context.
- Analyze the condition in the new context and make sure it's a boolean.
- Analyze the update in the new context.
- Analyze the body in the new context.
- Implement `codegen()` .

```
>_ ~/workspace/j--
$ java ForStatement 100
5050
```

Problem 7. (*Break Statement*) Add support for a break statement.

Directions:

- Create an empty stack in `JMember` to keep track of the surrounding control-flow statement

  ```
  public static Stack<JStatement> enclosingStatement = new Stack<JStatement>();
  ```

- Declare two instance variables in each control-flow statement (do, while, for, and switch): `boolean hasBreak` and `String breakLabel`.
- Each control-flow statement (do, while, for, and switch), during analysis, must push a reference to self onto `JMember.enclosingStatement` upon entry, and pop the reference upon exit.
- Each control-flow statement (do, while, for, and switch), during codegen, must set `breakLabel` to an appropriate label if `hasBreak` is `true`, and add the label at the appropriate place.
- Declare an instance variable `JStatement enclosingStatement` in `JBreakStatement`, and during analysis, set it to the value at the top of `JMember.enclosingStatement` (use `peek()`). Then set the enclosing statement's `hasBreak` variable to `true`.
- During codegen in `JBreakStatement`, access the break label via the enclosing statement, and generate an unconditional jump to that label.

```
>_ ~/workspace/j--
$ java BreakStatement 1000
168
```

Problem 8. (*Continue Statement*) Add support for a continue statement.

Directions:

- Declare two instance variables in each control-flow statement (do, while, and for): `boolean hasContinue` and `String continueLabel`.
- Each control-flow statement (do, while, and for), during codegen, must set `continueLabel` to an appropriate label if `hasContinue` is `true`, and add the label at the appropriate place.
- During analysis in `JContinueStatement`, set the enclosing statement's `hasContinue` variable to `true`.
- During codegen in `JContinueStatement`, access the continue label via the enclosing statement, and generate an unconditional jump to that label.

```
>_ ~/workspace/j--
$ java ContinueStatement 100
3.121594652591011
```

Problem 9. (*Exception Handlers*) Add support for exception handling, which involves supporting the `try`, `catch`, `finally`, `throw`, and `throws` clauses.

Directions:

- Implement `analyze()` and `codegen()` in `JThrowStatement`.
- During the analysis of `JConstructorDeclaration` and `JMethodDeclaration`, convert the list of exceptions (stored as `TypeName` objects) into a list of their JVM names (stored as strings). During codegen, include this list in the method header.
- In `analyze()` in `JTryStatement`:
  - Analyze the try block.
  - Analyze each catch block in a new `LocalContext` created from `context` as the parent — the catch parameter must be declared in this new context.
  - Analyze the optional finally block in a new `LocalContext` created from `context` as the parent.
- In `codegen()` in `JTryStatement`:
  - Add a “start try” label, generate code for the try block, generate code for the optional finally block and an unconditional jump to an “end finally” label, and add an “end try” label.
  - For each catch block, add a “start catch” label, generate code to store the catch variable, generate code for the catch block, add “end catch” label, add an exception handler with the appropriate arguments, and generate code for the optional finally block and an unconditional jump to an “end finally” label.
  - For the optional finally block: add a “start finally” label, generate an `ASTORE` instruction with the offset `o` obtained from the context for the finally block, add a “start finally plus one” label, generate code for the finally block, generate an `ALOAD` instruction with the offset `o` and an `ATHROW` instruction, add an “end finally” label, and add an exception handler with arguments “start try”, “end try”, “start finally”, and `null`; for each catch block, add an exception handler with the arguments “start catch”, “end catch”, “start finally”, and `null`; and add an exception handler with the arguments “start finally”, “start finally plus one”, “start finally”, and `null`.
  - Consult `$j/j--/tests/clemitter/GenExceptionHandler.java` for more hints on codegen.

```
>_ ~/workspace/j--
$ java ExceptionHandlers
x not specified
Done!
$ java ExceptionHandlers "two"
x must be a double
Done!
$ java ExceptionHandlers -2
x must be positve
Done!
$ java ExceptionHandlers 2
1.4142135623730951
Done!
```

Problem 10. (*Interface Type Declaration*) Implement support for interface declaration.

Directions:

- In `interfaceMemberDecl()` in `Parser`, implicitly add `"abstract"` and `"public"` to the list of modifiers for interface methods.
- In the constructor of `JInterfaceDeclaration`, implicitly add `"abstract"` and `"interface"` to the list of modifiers.
- Modify the `codegen()` method in `JClassDeclaration` to include a list of implemented interfaces in the class header.
- Implement the rest of `JInterfaceDeclaration` using `JClassDeclaration` as a model.

```
>_ ~/workspace/j--
$ java Interface 10
fIter(10) = 3628800
fRec(10)  = 3628800
```

Before you submit your files, make sure:

- Your code is adequately commented and follows good programming principles.
- You use the template file `report.txt` for your report.
- Your report meets the prescribed guidelines.

Files to submit:

1. `JArrayExpression.java`
2. `JArrayInitializer.java`
3. `JAssignment.java`
4. `JBinaryExpression.java`
5. `JBooleanBinaryExpression.java`
6. `JBreakStatement.java`
7. `JCastOp.java`
8. `JClassDeclaration.java`
9. `JComparisonExpression.java`
10. `JConditionalExpression.java`
11. `JConstructorDeclaration.java`
12. `JContinueStatement.java`
13. `JDoStatement.java`
14. `JForStatement.java`
15. `JInterfaceDeclaration.java`
16. `JLiteralDouble.java`
17. `JLiteralLong.java`
18. `JMember.java`
19. `JMethodDeclaration.java`
20. `JReturnStatement.java`
21. `JSwitchStatement.java`
22. `JThrowStatement.java`

23. `JTryStatement.java`
24. `JUnaryExpression.java`
25. `JVariable.java`
26. `JVariableDeclaration.java`
27. `JWhileStatement.java`
28. `Parser.java`
29. `Scanner.java`
30. `TokenInfo.java`
31. `report.txt`