The goal of this project is to modify the JavaCC specification file `$j/j--/src/jminusminus/j--.jj` for *j--* to add more Java tokens and programming constructs to the *j--* language. In the first part, you will modify the scanner section of the `j--.jj` file to support the Java tokens that you handled as part of Project 2 (Scanning). In the second part, you will modify the parser section of the file to support the Java programming constructs that you handled as part of Project 3 (Parsing). To compile the *j--* compiler with the JavaCC front-end, ie, with the scanner and parser generated by JavaCC, run the following command:

```
$ ant clean javacc compileJavaCC jar
```

Part I: Additions to JavaCC Scanner

To scan your *j--* programs using the JavaCC scanner, you need to run the `javaccj--` command as follows:

```
$ $j/j--/bin/javaccj-- -t P.java
```

which only scans `P.java` and prints the tokens in the program along with the line number where each token appears.

Problem 1. (*Multiline Comment*) Add support for multiline comment, where all the text from the ASCII characters `/*` to the ASCII characters `*/` is ignored.

```
$ $j/j--/bin/javaccj-- -t tests/MultiLineComment.java
5         : "public" = public
5         : "class" = class
5         : <IDENTIFIER> = MultiLineComment
5         : "{" = {
9         : "public" = public
9         : "static" = static
9         : "void" = void
9         : <IDENTIFIER> = main
9         : "(" = (
9         : <IDENTIFIER> = String
9         : "[" = [
9         : "]" = ]
9         : <IDENTIFIER> = args
9         : ")" = )
9         : "{" = {
13        : "}" = }
14        : "}" = }
14        : <EOF> =
```

Problem 2. (*Reserved Words*) Add support for the following reserved words.

```
abstract    const       finally       int          public          this
boolean     continue    float         interface    return          throw
break       default     for           long         short           throws
byte        do          goto          native       static          transient
case        double      if            new          strictfp        try
catch       else        implements    package      super           void
char        extends     import        private      switch          volatile
class       final       instanceof    protected    synchronized    while
```

```
$ $j/j--/bin/javaccj-- -t tests/ReservedWords.java
1         : "public" = public
1         : "class" = class
1         : <IDENTIFIER> = ReservedWords
1         : "extends" = extends
1         : <IDENTIFIER> = SomeClass
1         : "implements" = implements
1         : <IDENTIFIER> = SomeInterface
1         : "{" = {
2         : "public" = public
2         : "static" = static
2         : "void" = void
2         : <IDENTIFIER> = main
```

```
2         : "(" = (
2         : <IDENTIFIER> = String
2         : "[" = [
2         : "]" = ]
2         : <IDENTIFIER> = args
2         : ")" = )
2         : "{" = {
3         : "do" = do
3         : "{" = {
5         : "}" = }
5         : "while" = while
5         : "(" = (
5         : "true" = true
5         : ")" = )
5         : ";" = ;
6         : "for" = for
6         : "(" = (
6         : ";" = ;
6         : ";" = ;
6         : ")" = )
6         : "{" = {
7         : "try" = try
7         : "{" = {
8         : "if" = if
8         : "(" = (
8         : "true" = true
8         : ")" = )
8         : "{" = {
8         : "continue" = continue
8         : ";" = ;
8         : "}" = }
9         : "}" = }
10        : "catch" = catch
10        : "(" = (
10        : <IDENTIFIER> = SomeException
10        : <IDENTIFIER> = e
10        : ")" = )
10        : "{" = {
12        : "}" = }
13        : "}" = }
14        : "final" = final
14        : "int" = int
14        : <IDENTIFIER> = x
14        : ";" = ;
15        : "}" = }
16        : "}" = }
16        : <EOF> =
```

Problem 3. (*Operators*) Add support for the following operators.

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

```
$ $j/j--/bin/javaccj-- -t tests/Operators1.java
1         : "public" = public
1         : "class" = class
1         : <IDENTIFIER> = Operators1
1         : "{" = {
2         : "public" = public
2         : "static" = static
2         : "void" = void
2         : <IDENTIFIER> = main
2         : "(" = (
2         : <IDENTIFIER> = String
2         : "[" = [
2         : "]" = ]
```

```
2         : <IDENTIFIER> = args
2         : ")" = )
2         : "{" = {
3         : "int" = int
3         : <IDENTIFIER> = x
3         : "=" = =
3         : <INT_LITERAL> = 100
3         : ";" = ;
4         : <IDENTIFIER> = x
4         : "-=" = -=
4         : <INT_LITERAL> = 1
4         : ";" = ;
5         : <IDENTIFIER> = x
5         : "%=" = %=
5         : <INT_LITERAL> = 7
5         : ";" = ;
6         : "boolean" = boolean
6         : <IDENTIFIER> = y
6         : "=" = =
6         : <IDENTIFIER> = x
6         : ">=" = >=
6         : <INT_LITERAL> = 10
6         : "||" = ||
6         : <IDENTIFIER> = False
6         : ";" = ;
7         : "int" = int
7         : <IDENTIFIER> = z
7         : "=" = =
7         : <IDENTIFIER> = y
7         : "?" = ?
7         : <INT_LITERAL> = 2
7         : ":" = :
7         : <INT_LITERAL> = 0
7         : ";" = ;
8         : "}" = }
9         : "}" = }
9         : <EOF> =
```

Problem 4. (*Separators*) Add support for the following separators.

```
,    .    [    {    (    )    }    ]    ;    :
```

```
$ $j/j--/bin/javaccj-- -t tests/Separators.java
1         : "public" = public
1         : "class" = class
1         : <IDENTIFIER> = Separators
1         : "{" = {
2         : "public" = public
2         : "static" = static
2         : "void" = void
2         : <IDENTIFIER> = main
2         : "(" = (
2         : <IDENTIFIER> = String
2         : "[" = [
2         : "]" = ]
2         : <IDENTIFIER> = args
2         : ")" = )
2         : "{" = {
3         : "switch" = switch
3         : "(" = (
3         : <IDENTIFIER> = args
3         : "[" = [
3         : <INT_LITERAL> = 0
3         : "]" = ]
3         : ")" = )
3         : "{" = {
4         : "case" = case
```

```
4         : <STRING_LITERAL> = "1"
4         : ":" = :
6         : "break" = break
6         : ";" = ;
7         : "case" = case
7         : <STRING_LITERAL> = "2"
7         : ":" = :
8         : "for" = for
8         : "(" = (
8         : <IDENTIFIER> = String
8         : <IDENTIFIER> = x
8         : ":" = :
8         : <IDENTIFIER> = args
8         : ")" = )
8         : "{" = {
10        : "}" = }
11        : "break" = break
11        : ";" = ;
12        : "default" = default
12        : ":" = :
14        : "}" = }
15        : "}" = }
16        : "}" = }
16        : <EOF> =
```

Problem 5. (*Literals*) Add support for (just decimal for now) int, long, float, and double literals.

```
<int_literal> = 0|(1-9) {0-9} // decimal

<long_literal> = <int_literal> (l|L)

<float_literal> = (0-9) {0-9} . {0-9} [(e|E) [+|-] (0-9) {0-9}] (f|F)
                | . {0-9} [(e|E) [+|-] (0-9) {0-9}] (f|F)
                | (0-9) {0-9} [(e|E) [+|-] (0-9) {0-9}] (f|F)
                | (0-9) {0-9} ((e|E) ([+|-] (0-9) {0-9}) (f|F)

<double_literal> = {0-9} [[ . ] {0-9} [(e|E) [+|-] (0-9) {0-9}]] [d|D]
```

```
$ $j/j--/bin/javaccj-- -t tests/Literals.java
1         : "public" = public
1         : "class" = class
1         : <IDENTIFIER> = Literals
1         : "{" = {
2         : "public" = public
2         : "static" = static
2         : "void" = void
2         : <IDENTIFIER> = main
2         : "(" = (
2         : <IDENTIFIER> = String
2         : "[" = [
2         : "]" = ]
2         : <IDENTIFIER> = args
2         : ")" = )
2         : "{" = {
3         : "int" = int
3         : <IDENTIFIER> = a
3         : "=" = =
3         : <INT_LITERAL> = 372
3         : ";" = ;
4         : "long" = long
4         : <IDENTIFIER> = b
4         : "=" = =
4         : <LONG_LITERAL> = 777L
4         : ";" = ;
5         : "float" = float
5         : <IDENTIFIER> = c
5         : "=" = =
```

```
5         : <FLOAT_LITERAL> = 3.14f
5         : ";" = ;
6         : "double" = double
6         : <IDENTIFIER> = d
6         : "=" = =
6         : <DOUBLE_LITERAL> = 1e-9d
6         : ";" = ;
7         : "}" = }
8         : "}" = }
8         : <EOF> =
```

Part II: Additions to JavaCC Parser

To parse your *j--* programs using the JavaCC parser, you need to run the `javaccj--` command as follows:

```
$ $j/j--/bin/javaccj-- -p P.java
```

which will only parse `P.java` and print the AST for the program in XML format.

Note: The AST shown (as XML) for each problem is only a suggestion as to what the AST ought to look like once the syntactic constructs for that problem are implemented in *j--*. You are *not* expected to produce exactly the same AST, but just something similar. The autograder will not match your AST against ours for correctness, but instead will test if your parser parses our pass tests without errors and our fail tests with suitable error messages.

Problem 6. (*Double Basic Type*) Add support for the `double` basic type.

basicType ::= `boolean` | `char` | `int` | `double`

```
$ $j/j--/bin/javaccj-- -p tests/Double.java
<?xml version="1.0" encoding="utf-8"?>
<JCompilationUnit line="1">
  <Source fileName="tests/Double.java"/>
  <Imports>
    <Import name="java.lang.Double"/>
    <Import name="java.lang.System"/>
  </Imports>
  <TypeDeclarations>
    <JClassDeclaration line="4" name="Double" super="java.lang.Object">
      <Modifiers>
        <Modifier name="public"/>
      </Modifiers>
      <Implements>
      </Implements>
      <ClassBlock>
      <JMethodDeclaration line="5" name="main" returnType="void">
        <Modifiers>
          <Modifier name="public"/>
          <Modifier name="static"/>
        </Modifiers>
        <FormalParameters>
          <JFormalParameter line="5" name="args" type="String[]"/>
        </FormalParameters>
        <Exceptions>
        </Exceptions>
        <Body>
          <JBlock line="5">
            <JVariableDeclaration>
              <Modifiers>
              </Modifiers>
              <VariableDeclarators>
                <JVariableDeclarator line="6" name="r" type="double">
                  <Initializer>
                    <JMessageExpression line="6" name="parseDouble">
                      <Arguments>
                        <Argument>
```

```
                        <JArrayExpression>
                          <TheArray>
                            <JVariable name="args"/>
                          </TheArray>
                          <IndexExpression>
                            <JLiteralInt line="6" type="" value="0"/>
                          </IndexExpression>
                        </JArrayExpression>
                      </Argument>
                    </Arguments>
                  </JMessageExpression>
                </Initializer>
              </JVariableDeclarator>
            </VariableDeclarators>
          </JVariableDeclaration>
          <JStatementExpression line="7">
            <JMessageExpression line="7" name="println">
              <Arguments>
                <Argument>
                  <JBinaryExpression line="7" type="" operator="*">
                    <Lhs>
                      <JBinaryExpression line="7" type="" operator="*">
                        <Lhs>
                          <JLiteralDouble line="7" type="" value="3.14159D"/>
                        </Lhs>
                        <Rhs>
                          <JVariable name="r"/>
                        </Rhs>
                      </JBinaryExpression>
                    </Lhs>
                    <Rhs>
                      <JVariable name="r"/>
                    </Rhs>
                  </JBinaryExpression>
                </Argument>
              </Arguments>
            </JMessageExpression>
          </JStatementExpression>
        </JBlock>
      </Body>
    </JMethodDeclaration>
    </ClassBlock>
  </JClassDeclaration>
 </TypeDeclarations>
</JCompilationUnit>
```

Problem 7. (*Operators*) Add support for the logical or operator `||`, the assignment operators `-=`, `*=`, `/=`, `%=`, the prefix operator `--`, and the postfix operator `++`.

assignmentExpression ::= conditionalOrExpression // must be a valid lhs
[(`=`
| `+=`
| `-=`
| `*=`
| `/=`
| `%=`
) assignmentExpression
]

conditionalOrExpression ::= conditionalAndExpression { `||` conditionalAndExpression }

unaryExpression ::= `++` unaryExpression
| -- unaryExpression

| - unaryExpression
| + unaryExpression
| simpleUnaryExpression

postfixExpression ::= primary { selector } { -- | ++ }

```
$ $j/j--/bin/javaccj-- -p tests/Operators2.java
<?xml version="1.0" encoding="utf-8"?>
<JCompilationUnit line="1">
  <Source fileName="tests/Operators2.java"/>
  <Imports>
    <Import name="java.lang.System"/>
  </Imports>
  <TypeDeclarations>
    <JClassDeclaration line="3" name="Operators2" super="java.lang.Object">
      <Modifiers>
        <Modifier name="public"/>
      </Modifiers>
      <Implements>
      </Implements>
      <ClassBlock>
      <JMethodDeclaration line="4" name="main" returnType="void">
        <Modifiers>
          <Modifier name="public"/>
          <Modifier name="static"/>
        </Modifiers>
        <FormalParameters>
          <JFormalParameter line="4" name="args" type="String[]"/>
        </FormalParameters>
        <Exceptions>
        </Exceptions>
        <Body>
          <JBlock line="4">
            <JStatementExpression line="5">
              <JMessageExpression line="5" name="println">
                <Arguments>
                  <Argument>
                    <JBinaryExpression line="5" type="" operator="||">
                      <Lhs>
                        <JBinaryExpression line="5" type="" operator="&amp;amp;&amp;amp;">
                          <Lhs>
                            <JLiteralTrue line="5" type=""/>
                          </Lhs>
                          <Rhs>
                            <JLiteralFalse line="5" type=""/>
                          </Rhs>
                        </JBinaryExpression>
                      </Lhs>
                      <Rhs>
                        <JLiteralTrue line="5" type=""/>
                      </Rhs>
                    </JBinaryExpression>
                  </Argument>
                </Arguments>
              </JMessageExpression>
            </JStatementExpression>
            <JVariableDeclaration>
              <Modifiers>
              </Modifiers>
              <VariableDeclarators>
                <JVariableDeclarator line="6" name="x" type="int">
                  <Initializer>
                    <JLiteralInt line="6" type="" value="42"/>
                  </Initializer>
                </JVariableDeclarator>
              </VariableDeclarators>
            </JVariableDeclaration>
```

```
      <JStatementExpression line="7">
        <JBinaryExpression line="7" type="" operator="-=">
          <Lhs>
            <JVariable name="x"/>
          </Lhs>
          <Rhs>
            <JLiteralInt line="7" type="" value="2"/>
          </Rhs>
        </JBinaryExpression>
      </JStatementExpression>
      <JStatementExpression line="8">
        <JBinaryExpression line="8" type="" operator="*=">
          <Lhs>
            <JVariable name="x"/>
          </Lhs>
          <Rhs>
            <JLiteralInt line="8" type="" value="2"/>
          </Rhs>
        </JBinaryExpression>
      </JStatementExpression>
      <JStatementExpression line="9">
        <JBinaryExpression line="9" type="" operator="/=">
          <Lhs>
            <JVariable name="x"/>
          </Lhs>
          <Rhs>
            <JLiteralInt line="9" type="" value="10"/>
          </Rhs>
        </JBinaryExpression>
      </JStatementExpression>
      <JStatementExpression line="10">
        <JBinaryExpression line="10" type="" operator="%=">
          <Lhs>
            <JVariable name="x"/>
          </Lhs>
          <Rhs>
            <JLiteralInt line="10" type="" value="3"/>
          </Rhs>
        </JBinaryExpression>
      </JStatementExpression>
      <JStatementExpression line="11">
        <JMessageExpression line="11" name="println">
          <Arguments>
            <Argument>
              <JUnaryExpression line="11" type="" operator="post++">
                <Operand>
                  <JVariable name="x"/>
                </Operand>
              </JUnaryExpression>
            </Argument>
          </Arguments>
        </JMessageExpression>
      </JStatementExpression>
      <JStatementExpression line="12">
        <JMessageExpression line="12" name="println">
          <Arguments>
            <Argument>
              <JUnaryExpression line="12" type="" operator="--pre">
                <Operand>
                  <JVariable name="x"/>
                </Operand>
              </JUnaryExpression>
            </Argument>
          </Arguments>
        </JMessageExpression>
      </JStatementExpression>
    </JBlock>
  </Body>
```

```
      </JMethodDeclaration>
      </ClassBlock>
    </JClassDeclaration>
  </TypeDeclarations>
</JCompilationUnit>
```

Problem 8. (*Blocks*) Add support for static and instance blocks.

classBody ::= `{` { `static` block // static block
| block // instance block
| modifiers memberDecl
}
`}`

```
$ $j/j--/bin/javaccj-- -p tests/Blocks.java
<?xml version="1.0" encoding="utf-8"?>
<JCompilationUnit line="1">
  <Source fileName="tests/Blocks.java"/>
  <Imports>
  </Imports>
  <TypeDeclarations>
    <JClassDeclaration line="1" name="Blocks" super="java.lang.Object">
      <Modifiers>
        <Modifier name="public"/>
      </Modifiers>
      <Implements>
      </Implements>
      <ClassBlock>
        <StaticBlock line="2">
          <JBlock line="2">
            <JVariableDeclaration>
              <Modifiers>
              </Modifiers>
              <VariableDeclarators>
                <JVariableDeclarator line="3" name="x" type="double">
                  <Initializer>
                    <JLiteralDouble line="3" type="" value="3.14159"/>
                  </Initializer>
                </JVariableDeclarator>
              </VariableDeclarators>
            </JVariableDeclaration>
          </JBlock>
        </StaticBlock>
        <InstanceBlock line="2">
          <JBlock line="6">
            <JVariableDeclaration>
              <Modifiers>
              </Modifiers>
              <VariableDeclarators>
                <JVariableDeclarator line="7" name="y" type="int">
                  <Initializer>
                    <JLiteralInt line="7" type="" value="42"/>
                  </Initializer>
                </JVariableDeclarator>
              </VariableDeclarators>
            </JVariableDeclaration>
          </JBlock>
        </InstanceBlock>
      </ClassBlock>
    </JClassDeclaration>
  </TypeDeclarations>
</JCompilationUnit>
```

Problem 9. (*Interface Type Declaration*) Implement support for interface declaration.

typeDeclaration ::= modifiers (classDeclaration | interfaceDeclaration)

classDeclaration ::= `class <identifier>` [`extends` qualifiedIdentifier]
[`implements` qualifiedIdentifier { , qualifiedIdentifier }]
classBody

interfaceDeclaration ::= `interface <identifier>` // can't be final
[`extends` qualifiedIdentifier { , qualifiedIdentifier }]
interfaceBody

interfaceBody ::= `{` { modifiers interfaceMemberDecl } `}`

interfaceMemberDecl ::= (`void` | type) `<identifier>` // method
formalParameters ;
| type variableDeclarators ; // fields; must have inits

```
$ $j/j--/bin/javaccj-- -p tests/Interface.java
<?xml version="1.0" encoding="utf-8"?>
<JCompilationUnit line="1">
  <Source fileName="tests/Interface.java"/>
  <Imports>
  </Imports>
  <TypeDeclarations>
    <JInterfaceDeclaration line="1" name="A">
      <SuperTypes>
      </SuperTypes>
      <Modifiers>
      </Modifiers>
      <InterfaceBlock>
        <JMethodDeclaration line="2" name="f" returnType="int">
          <Modifiers>
            <Modifier name="public"/>
          </Modifiers>
          <FormalParameters>
            <JFormalParameter line="2" name="x" type="int"/>
          </FormalParameters>
          <Exceptions>
          </Exceptions>
        </JMethodDeclaration>
      </InterfaceBlock>
    </JInterfaceDeclaration>
    <JClassDeclaration line="5" name="B" super="java.lang.Object">
      <Modifiers>
        <Modifier name="public"/>
      </Modifiers>
      <Implements>
        <Interface name="A"/>
      </Implements>
      <ClassBlock>
      <JMethodDeclaration line="6" name="f" returnType="int">
        <Modifiers>
          <Modifier name="public"/>
        </Modifiers>
        <FormalParameters>
          <JFormalParameter line="6" name="x" type="int"/>
        </FormalParameters>
        <Exceptions>
        </Exceptions>
        <Body>
          <JBlock line="6">
            <JReturnStatement line="7">
              <JBinaryExpression line="7" type="" operator="*">
                <Lhs>
```

```
                <JVariable name="x"/>
              </Lhs>
              <Rhs>
                <JVariable name="x"/>
              </Rhs>
            </JBinaryExpression>
          </JReturnStatement>
        </JBlock>
      </Body>
    </JMethodDeclaration>
    </ClassBlock>
  </JClassDeclaration>
 </TypeDeclarations>
</JCompilationUnit>
```

Problem 10. (*Conditional Expression*) Add support for conditional expression (`e1 ? e2 : e3`).

conditionalExpression ::= conditionalOrExpression
[? assignmentExpression : conditionalExpression]

```
$ $j/j--bin/javaccj-- -p tests/ConditionalExpression.java
<?xml version="1.0" encoding="utf-8"?>
<JCompilationUnit line="1">
  <Source fileName="../ConditionalExpression.java"/>
  <Imports>
    <Import name="java.lang.Integer"/>
    <Import name="java.lang.System"/>
  </Imports>
  <TypeDeclarations>
    <JClassDeclaration line="4" name="ConditionalExpression" super="java.lang.Object">
      <Modifiers>
        <Modifier name="public"/>
      </Modifiers>
      <Implements>
      </Implements>
      <ClassBlock>
      <JMethodDeclaration line="5" name="main" returnType="void">
        <Modifiers>
          <Modifier name="public"/>
          <Modifier name="static"/>
        </Modifiers>
        <FormalParameters>
          <JFormalParameter line="5" name="args" type="String[]"/>
        </FormalParameters>
        <Exceptions>
        </Exceptions>
        <Body>
          <JBlock line="5">
            <JVariableDeclaration>
              <Modifiers>
              </Modifiers>
              <VariableDeclarators>
                <JVariableDeclarator line="6" name="x" type="int">
                  <Initializer>
                    <JMessageExpression line="6" name="parseInt">
                      <Arguments>
                        <Argument>
                          <JArrayExpression>
                            <TheArray>
                              <JVariable name="args"/>
                            </TheArray>
                            <IndexExpression>
                              <JLiteralInt line="6" type="" value="0"/>
                            </IndexExpression>
                          </JArrayExpression>
```

```
                    </Argument>
                  </Arguments>
                </JMessageExpression>
              </Initializer>
            </JVariableDeclarator>
          </VariableDeclarators>
        </JVariableDeclaration>
        <JStatementExpression line="7">
          <JMessageExpression line="7" name="println">
            <Arguments>
              <Argument>
                <JConditionalExpression line="7" type="" operator="?">
                  <TestExpression>
                    <JBinaryExpression line="7" type="" operator="&gt;">
                      <Lhs>
                        <JVariable name="x"/>
                      </Lhs>
                      <Rhs>
                        <JLiteralInt line="7" type="" value="2"/>
                      </Rhs>
                    </JBinaryExpression>
                  </TestExpression>
                  <TrueClause>
                    <JLiteralString line="7" type="" value="&quot;Yes&quot;"/>
                  </TrueClause>
                  <FalseClause>
                    <JLiteralString line="7" type="" value="&quot;No&quot;"/>
                  </FalseClause>
                </JConditionalExpression>
              </Argument>
            </Arguments>
          </JMessageExpression>
        </JStatementExpression>
      </JBlock>
    </Body>
  </JMethodDeclaration>
  </ClassBlock>
 </JClassDeclaration>
</TypeDeclarations>
</JCompilationUnit>
```

Problem 11. (*For Statements*) Add support for a for-statement, both the basic for-statement and the enhanced for-statement.

statement ::= block
| `<identifier>` : statement
| `if` parExpression statement [`else` statement]
| `for` ([forInit] ; [expression] ; [forUpdate]) statement
| `for` (type `<identifier>` : expression) statement
| `while` parExpression statement
| `return` [expression] ;
| ;
| statementExpression ;

forInit ::= statementExpression { , statementExpression }
| [`final`] type variableDeclarators

forUpdate ::= statementExpression { , statementExpression }

```
$ $j/j--/bin/javaccj-- -p tests/ForStatements.java
<?xml version="1.0" encoding="utf-8"?>
<JCompilationUnit line="1">
```

```
<Source fileName="tests/ForStatements.java"/>
<Imports>
  <Import name="java.lang.System"/>
</Imports>
<TypeDeclarations>
  <JClassDeclaration line="3" name="ForStatements" super="java.lang.Object">
    <Modifiers>
      <Modifier name="public"/>
    </Modifiers>
    <Implements>
    </Implements>
    <ClassBlock>
    <JMethodDeclaration line="4" name="main" returnType="void">
      <Modifiers>
        <Modifier name="public"/>
        <Modifier name="static"/>
      </Modifiers>
      <FormalParameters>
        <JFormalParameter line="4" name="args" type="String[]"/>
      </FormalParameters>
      <Exceptions>
      </Exceptions>
      <Body>
        <JBlock line="4">
          <JVariableDeclaration>
            <Modifiers>
            </Modifiers>
            <VariableDeclarators>
              <JVariableDeclarator line="5" name="sum1" type="int">
                <Initializer>
                  <JLiteralInt line="5" type="" value="0"/>
                </Initializer>
              </JVariableDeclarator>
              <JVariableDeclarator line="5" name="sum2" type="int">
                <Initializer>
                  <JLiteralInt line="5" type="" value="0"/>
                </Initializer>
              </JVariableDeclarator>
            </VariableDeclarators>
          </JVariableDeclaration>
          <JForStatement line="6">
            <InitialExpression>
              <JVariableDeclaration>
                <Modifiers>
                </Modifiers>
                <VariableDeclarators>
                  <JVariableDeclarator line="6" name="i" type="int">
                    <Initializer>
                      <JLiteralInt line="6" type="" value="1"/>
                    </Initializer>
                  </JVariableDeclarator>
                </VariableDeclarators>
              </JVariableDeclaration>
            </InitialExpression>
            <TestExpression>
              <JBinaryExpression line="6" type="" operator="&lt;=">
                <Lhs>
                  <JVariable name="i"/>
                </Lhs>
                <Rhs>
                  <JLiteralInt line="6" type="" value="10"/>
                </Rhs>
              </JBinaryExpression>
            </TestExpression>
            <UpdateExpression>
              <JStatementExpression line="6">
                <JUnaryExpression line="6" type="" operator="post++">
                  <Operand>
```

```
                    <JVariable name="i"/>
                  </Operand>
                </JUnaryExpression>
              </JStatementExpression>
            </UpdateExpression>
            <Statement>
              <JBlock line="6">
                <JStatementExpression line="7">
                  <JBinaryExpression line="7" type="" operator="+=">
                    <Lhs>
                      <JVariable name="sum1"/>
                    </Lhs>
                    <Rhs>
                      <JVariable name="i"/>
                    </Rhs>
                  </JBinaryExpression>
                </JStatementExpression>
              </JBlock>
            </Statement>
          </JForStatement>
          <JVariableDeclaration>
            <Modifiers>
            </Modifiers>
            <VariableDeclarators>
              <JVariableDeclarator line="9" name="a" type="int[]">
                <Initializer>
                  <JArrayInitializer>
                    <JLiteralInt line="9" type="" value="1"/>
                    <JLiteralInt line="9" type="" value="2"/>
                    <JLiteralInt line="9" type="" value="3"/>
                    <JLiteralInt line="9" type="" value="4"/>
                    <JLiteralInt line="9" type="" value="5"/>
                    <JLiteralInt line="9" type="" value="6"/>
                    <JLiteralInt line="9" type="" value="7"/>
                    <JLiteralInt line="9" type="" value="8"/>
                    <JLiteralInt line="9" type="" value="9"/>
                    <JLiteralInt line="9" type="" value="10"/>
                  </JArrayInitializer>
                </Initializer>
              </JVariableDeclarator>
            </VariableDeclarators>
          </JVariableDeclaration>
          <JEnhancedForStatement line="10">
            <Parameter name="i" type="int"/>
            <Expression>
              <JVariable name="a"/>
            </Expression>
            <Statement>
              <JBlock line="10">
                <JStatementExpression line="11">
                  <JBinaryExpression line="11" type="" operator="+=">
                    <Lhs>
                      <JVariable name="sum2"/>
                    </Lhs>
                    <Rhs>
                      <JVariable name="i"/>
                    </Rhs>
                  </JBinaryExpression>
                </JStatementExpression>
              </JBlock>
            </Statement>
          </JEnhancedForStatement>
          <JStatementExpression line="13">
            <JMessageExpression line="13" name="println">
              <Arguments>
                <Argument>
                  <JBinaryExpression line="13" type="" operator="==">
                    <Lhs>
```

```
                        <JVariable name="sum1"/>
                      </Lhs>
                      <Rhs>
                        <JVariable name="sum2"/>
                      </Rhs>
                    </JBinaryExpression>
                  </Argument>
                </Arguments>
              </JMessageExpression>
            </JStatementExpression>
          </JBlock>
        </Body>
      </JMethodDeclaration>
      </ClassBlock>
    </JClassDeclaration>
  </TypeDeclarations>
</JCompilationUnit>
```

Problem 12. (*Exception Handlers*) Add support for exception handling, which involves supporting the `try`, `catch`, `finally`, `throw`, and `throws` clauses.

statement ::= block
| `<identifier>` : statement
| `if` parExpression statement [`else` statement]
| `while` parExpression statement
| `try` block
{ `catch` (formalParameter) block }
[`finally` block] // must be present if no catches
| `return` [expression] ;
| `throw` expression ;
| ;
| statementExpression ;

memberDecl ::= `<identifier>` // constructor
formalParameters
[`throws` qualifiedIdentifier { , qualifiedIdentifier }] block
| (`void` | type) `<identifier>` // method
formalParameters
[`throws` qualifiedIdentifier { , qualifiedIdentifier }] ;
| type variableDeclarators ; // fields

interfaceMemberDecl ::= (`void` | type) `<identifier>` // method
formalParameters
[`throws` qualifiedIdentifier { , qualifiedIdentifier }] ;
| type variableDeclarators ; // fields; must have inits

```
$ $j/j--/bin/javaccj-- -p tests/ExceptionHandlers.java
<?xml version="1.0" encoding="utf-8"?>
<JCompilationUnit line="1">
  <Source fileName="../ExceptionHandlers.java"/>
  <Imports>
  </Imports>
  <TypeDeclarations>
    <JClassDeclaration line="1" name="ExceptionHandlers" super="java.lang.Object">
      <Modifiers>
        <Modifier name="public"/>
      </Modifiers>
      <Implements>
      </Implements>
      <ClassBlock>
```

```
    <JMethodDeclaration line="2" name="f" returnType="void">
      <Modifiers>
        <Modifier name="private"/>
        <Modifier name="static"/>
      </Modifiers>
      <FormalParameters>
      </FormalParameters>
      <Exceptions>
        <Exception type="Exception1"/>
        <Exception type="Exception2"/>
      </Exceptions>
      <Body>
        <JBlock line="2">
          <JThrowStatement line="3">
            <JNewOp line="3" type="Exception1"/>
              <Arguments>
              </Arguments>
            </JNewOp>
          </JThrowStatement>
        </JBlock>
      </Body>
    </JMethodDeclaration>
    <JMethodDeclaration line="6" name="main" returnType="void">
      <Modifiers>
        <Modifier name="public"/>
        <Modifier name="static"/>
      </Modifiers>
      <FormalParameters>
        <JFormalParameter line="6" name="args" type="String[]"/>
      </FormalParameters>
      <Exceptions>
      </Exceptions>
      <Body>
        <JBlock line="6">
          <JTryCatchFinallyStatement line="7">
            <TryBlock>
              <JBlock line="7">
                <JStatementExpression line="8">
                  <JMessageExpression line="8" name="f">
                    <Arguments>
                    </Arguments>
                  </JMessageExpression>
                </JStatementExpression>
              </JBlock>
            </TryBlock>
            <CatchBlock>
              <JFormalParameter line="10" name="e1" type="Exception1"/>
              <JBlock line="10">
                <JEmptyStatement line="10"/>
              </JBlock>
            </CatchBlock>
            <CatchBlock>
              <JFormalParameter line="11" name="e2" type="Exception2"/>
              <JBlock line="11">
                <JEmptyStatement line="11"/>
              </JBlock>
            </CatchBlock>
            <FinallyBlock>
              <JBlock line="12">
                <JEmptyStatement line="12"/>
              </JBlock>
            </FinallyBlock>
          </JTryCatchFinallyStatement>
        </JBlock>
      </Body>
    </JMethodDeclaration>
    </ClassBlock>
  </JClassDeclaration>
```

```
  </TypeDeclarations>
</JCompilationUnit>
```

Files to Submit

1. `j--.zip` (*j*-- source tree as a single zip file)
2. `report.txt` (project report)

Before you submit:

- Make sure you create the zip file `j--.zip` such that it only includes the source files and not the binaries, which can be done on the terminal as follows:

```
$ cd $j/j--
$ ant clean
$ cd ..
$ tar -cvf j--.tar j--/*
$ gzip j--.tar
```

- Make sure the files `$j/j--/lexicalgrammar` and `$j/j--/grammar` are updated with the syntactic changes you have made to the *j*-- language.
- Make sure your report isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling/grammatical mistakes