Introduction to Compiler Construction

Assignment 3 (Parsing) Discussion

Problem 1 (Operators)

Add support for the following operators

```
-=   *=   /=   %=   !=   >=   <   ||   ++   --
```

Modify `assignmentExpression()` in `Parser` to parse the `-=`, `*=`, `/=`, and `%=` operators, using `JMinusAssignOp`, `JStarAssignOp`, `JDivAssignOp`, and `JRemAssignOp` in `JAssignment` as the corresponding AST representations

Modify `equalityExpression()` in `Parser` to parse the `!=` operator, using `JNotEqualOp` in `JBooleanBinaryExpression` as the corresponding AST representation

Modify `relationalExpression()` in `Parser` to parse the `>=` and `<` operators, using `JGreaterEqualOp` and `JLessThanOp` in `JComparisonExpression` as the corresponding AST representations

Add `conditionalOrExpression()` in `Parser` to parse the `||` operator, using `JLogicalOrOp` in `JBooleanBinaryExpression` as the corresponding AST representation; modify `conditionalExpression()` in `Parser` to now call `conditionalOrExpression()`

Modify `unaryExpression()` in `Parser` to parse the pre `--` operator, using `JPreDecrementOp` and `JUnaryExpression` as the corresponding AST representation

Modify `postfixExpression()` in `Parser` to parse the post `++` operator, using `JPostIncrementOp` and `JUnaryExpression` as the corresponding AST representation

Problem 1 (Operators)

Testing

× ~/workspace/j--

```
$ ant
$ ./bin/j-- -p parsing/Operators.java
```

Compare your output with the reference output in `parsing/Operators.ast`

Problem 2 (Long and Double Basic Types)

Add support for the `long` and `double` basic types

Modify the following methods in `Parser` to support longs and doubles

- `basicType()`
- `literal()` (use `JLiteralLong` and `JLiteralDouble` as the AST representations for a long and double literal respectively)
- `seeBasicType()`
- `seeReferenceType()`

Testing

```
× ~/workspace/j--
$ ant
$ ./bin/j-- -p parsing/Factorial.java
$ ./bin/j-- -p parsing/Quadratic.java
```

Compare your output with the reference output in `parsing/Factorial.ast` and `parsing/Quadratic.ast`

Problem 3 (For Statement)

Add support for a for statement

Make the following changes in `Parser` to support a for statement

- Add `ArrayList<JStatement> forInit()` to parse the `forInit` part
  - If *not* looking at a local variable declaration (use `!seeLocalVariableDeclaration()`), then return a list of statement expressions
  - Otherwise, return a list containing a single `JVariableDeclaration` object encapsulating the variable declarators (see `localVariableDeclarationStatement()` for how to construct that object)
- Add `ArrayList<JStatement> forUpdate()` to parse the `forUpdate` part
- Modify `statement()` to parse a for statement, using `JForStatement` as the AST representation for a for statement

Testing

```
× ~/workspace/j--
```

```
$ ant
$ ./bin/j-- -p parsing/ForStatement.java
```

Compare your output with the reference output in `parsing/ForStatement.ast`

Problem 4 (Break Statement)

Add support for a break statement

Modify `statement()` to parse a break statement, using `JBreakStatement` as the AST representation

```
× ~/workspace/j--
```

```
$ ant
$ ./bin/j-- -p parsing/BreakStatement.java
```

Compare your output with the reference output in `parsing/BreakStatement.ast`

Problem 5 (Continue Statement)

Add support for a continue statement

Modify `statement()` to parse a continue statement, using `JContinueStatement` as the AST representation

```
× ~/workspace/j--
```

```
$ ant
$ ./bin/j-- -p parsing/ContinueStatement.java
```

Compare your output with the reference output in `parsing/ContinueStatement.ast`

Problem 6 (Switch Statement)

Add support for a switch statement

Make the following changes in `Parser` to support a switch statement

- Add `SwitchStatementGroup switchBlockStatementGroup()` to parse the `switchBlockStatementGroup` part
  - After parsing one or more `switchLabel`, parse zero or more `blockStatement` until you see a `CASE`, `DEFLT`, or `RCURLY`
- Add `JExpression switchLabel()` to parse the `switchLabel` part, which must return an expression for a case and `null` for default
- Modify `statement()` to parse a switch statement, using `JSwitchStatement` as the AST representation for a switch statement
  - After parsing `SWITCH parExpression LCURLY`, parse zero or more `switchBlockStatementGroup` until you see an `RCURLY` or `EOF`, and then scan an `RCURLY`

```
× ~/workspace/j--
$ ant
$ ./bin/j-- -p parsing/SwitchStatement.java
```

Compare your output with the reference output in `parsing/SwitchStatement.ast`