

```

1 options {
2     LOOKAHEAD=1;
3 }
4
5 PARSER_BEGIN(JMM)
6
7 import java.util.List;
8 import java.util.ArrayList;
9 import pt.up.fe.comp.jmm.report.Report;
10 import pt.up.fe.comp.jmm.report.ReportType;
11 import pt.up.fe.comp.jmm.report.Stage;
12 public class JMM {
13     private List<Report> reports = new ArrayList<Report>();
14
15     public static void main(String[] args) throws ParseException {
16         System.out.println("Write a Java-- program:");
17         JMM jmm = new JMM(System.in);
18
19         SimpleNode root = jmm.Program(); // Returns reference to root node
20
21         System.out.println("Finished parsing");
22     }
23
24     public List<Report> getReports() {
25         return this.reports;
26     }
27 }
28
29 PARSER_END(JMM)
30
31 SKIP : {
32     " " | "\r" | "\t" | "\n"
33 }
34
35 // Multi-line comment
36
37 SKIP : {
38     "/*" : WithinComment
39 }
40
41 <WithinComment> SKIP : {
42     "*/" : DEFAULT
43 }
44
45 <WithinComment> MORE : {
46     <~[]>
47 }
48
49 // Single-line comment
50
51 SPECIAL_TOKEN : {
52     <SINGLE_LINE_COMMENT: "//" (~["\n", "\r"])* ("\"|\"\\r\"|\"\\n\"")>
53 }
54
55 // Keywords
56
57 TOKEN : {
58     <IMPORT: "import">
59     | <CLASS: "class">
60     | <EXTENDS: "extends">
61     | <PUBLIC: "public">
62     | <RETURN: "return">
63     | <STATIC: "static">
64     | <VOID: "void">
65     | <MAIN: "main">
66     | <STRING: "String">
67     | <TRUE: "true">
68     | <FALSE: "false">
69     | <INT: "int">
70     | <BOOLEAN: "boolean">
71     | <IF: "if">
72     | <ELSE: "else">

```

```

73     | <WHILE: "while">
74     | <THIS: "this">
75     | <NEW: "new">
76     | <LENGTH: "length">
77 }
78
79 // Delimiters
80
81 TOKEN : {
82     <LEFT_PARENTHESSES: "(">
83     | <RIGHT_PARENTHESSES: ")">
84     | <LEFT_BRACE: "{">
85     | <RIGHT_BRACE: "}">
86     | <LEFT_BRACKET: "[">
87     | <RIGHT_BRACKET: "]">
88     | <SEMICOLON: ";">
89     | <COMMA: ",">
90 }
91
92 // Operators
93
94 TOKEN : {
95     <ADD: "+">
96     | <SUB: "-">
97     | <MUL: "*">
98     | <DIV: "/">
99     | <ASSIGN: "=">
100    | <DOT: ".">
101    | <NOT: "!">
102    | <AND: "&&">
103    | <LT: "<">
104 }
105
106 // Symbols
107
108 TOKEN: {
109     <IDENTIFIER:
110         ([ "A"-"Z", "a"-"z", "$" ] ( [ "0"-"9", "A"-"Z", "a"-"z", "$", "_" ] )*) |
111         ([ "_" ] ( [ "0"-"9", "A"-"Z", "a"-"z", "$", "_" ] )+ )
112     > |
113     <INTEGER: ([ "0"-"9" ] )+>
114 }
115
116 SimpleNode Program(): {} {
117     ImportDeclaration()
118     ClassDeclaration()
119     <EOF>
120     { return jjtThis; }
121 }
122
123 void ImportDeclaration() #ImportDeclaration: {} {
124     (<IMPORT> <IDENTIFIER> (<DOT> <IDENTIFIER>)* <SEMICOLON> )*
125 }
126
127 void ClassDeclaration() #ClassDeclaration : {Token t;} {
128     <CLASS> t=<IDENTIFIER> { jjtThis.put("name", t.image); } [ "extends" <IDENTIFIER> ]
129     <LEFT_BRACE>
130         ClassBody()
131     <RIGHT_BRACE>
132 }
133
134 void ClassBody(): {} {
135     ( VarDeclaration() )* ( MethodDeclaration() )*
136 }
137
138 void VarDeclaration(): {Token t;} {
139     Type() t=<IDENTIFIER> <SEMICOLON>
140     { jjtThis.put("name", t.image); }
141 }
142
143 void MethodDeclaration(): {} {
144     <PUBLIC>

```

```

145     (
146         (
147             <STATIC> <VOID> <MAIN>
148             <LEFT_PARENTHESSES> <STRING> <LEFT_BRACKET> <RIGHT_BRACKET> <IDENTIFIER> <RIGHT_PARENTHESSES> <
LEFT_BRACE>
149             MethodBody()
150         ) |
151         (
152             Type() <IDENTIFIER> <LEFT_PARENTHESSES>
153             [ Type() <IDENTIFIER> ("," Type() <IDENTIFIER>)* ]
154             <RIGHT_PARENTHESSES> <LEFT_BRACE>
155             MethodBody()
156             <RETURN> Expression() <SEMICOLON>
157         )
158     )
159 <RIGHT_BRACE>
160 }
161
162 void MethodBody(): {} {
163     ( LOOKAHEAD(2) VarDeclaration() )*
164     ( Statement() )*
165 }
166
167 void Type() #void : {} {
168     ( <INT> [<LEFT_BRACKET> <RIGHT_BRACKET>] )
169     | <BOOLEAN>
170     | <IDENTIFIER>
171 }
172
173 void Statement() #void : {} {
174     ( <LEFT_BRACE> ( Statement() )* <RIGHT_BRACE> )
175     | IfStatement()
176     | WhileStatement()
177     | LOOKAHEAD(2) Assignment()
178     | Expression() <SEMICOLON>
179 }
180
181 void IfStatement() #IfStatement : {} {
182     <IF> <LEFT_PARENTHESSES> Expression() <RIGHT_PARENTHESSES>
183     Statement()
184     <ELSE>
185     Statement()
186 }
187
188 void Assignment() #Assignment : {} {
189     <IDENTIFIER> [ <LEFT_BRACKET> Expression() <RIGHT_BRACKET> ] <ASSIGN> Expression() <SEMICOLON>
190 }
191
192 void WhileStatement() #WhileStatement : {} {
193     <WHILE> (
194         <LEFT_PARENTHESSES>
195         | recover_error(
196             new int[] {LEFT_BRACE, INTEGER, TRUE, FALSE, IDENTIFIER, THIS, NEW, NOT, LEFT_PARENTHESSES},
197             "Got '" + getToken(1).toString() + "' expected '(' token"
198         )
199     )
200
201     try {
202         Expression()
203     }
204
205     catch (ParseException e) {
206         recover_error(
207             new int[] {LEFT_BRACE, INTEGER, TRUE, FALSE, IDENTIFIER, THIS, NEW, NOT, LEFT_PARENTHESSES},
208             "Found invalid expression '" + e.currentToken + "'"
209         );
210     }
211
212     (
213         <RIGHT_PARENTHESSES>
214         | recover_error(
215             new int[] {LEFT_BRACE, IF, WHILE, INTEGER, TRUE, FALSE, IDENTIFIER, THIS, NEW, NOT,

```

```

215 LEFT_PARENTHESSES},
216         "Got '" + getToken(1).toString() + "' expected ')' token"
217     )
218 }
219
220 try {
221     Statement()
222 }
223
224 catch (ParseException e) {
225     recover_error(new int[]{RIGHT_BRACE}, "Found invalid statement '" + e.currentToken + "'");
226 }
227 }
228
229 JAVACODE
230 void recover_error(int[] skipTo, String msg) {
231     ParseException e = generateParseException();
232
233     /* System.err.println("Found error on line " + e.currentToken.beginLine + ", column " + e.
currentToken.beginColumn);
234     System.err.println("\t" + msg);*/
235
236     Report report_error = Report.newError(Stage.SYNTATIC, e.currentToken.beginLine, e.currentToken.
beginColumn, msg, e);
237
238     this.reports.add(report_error);
239
240     Token t = getToken(1);
241
242     String skipped_tokens = "";
243     boolean match = false;
244
245     while (!match) {
246         for (int matcher : skipTo) {
247             if (t.kind == matcher) {
248                 match = true;
249                 break;
250             }
251         }
252
253         if (!match) { // skip current token
254             skipped_tokens += token.toString() + " ";
255             getNextToken();
256             t = getToken(1);
257         }
258     }
259
260     // System.err.println("Tokens skipped " + skipped_tokens);
261 }
262
263 void Expression() #void: {} {
264     LT() (<AND> LT() #And(2))*
265 }
266
267 void LT() #void: {} {
268     AddSub() (<LT> AddSub() #Less_Than(2))*
269 }
270
271 void AddSub() #void: {} {
272     MultDiv() (<ADD> MultDiv() #Add(2) | <SUB> MultDiv() #Sub(2) )*
273 }
274
275 void MultDiv() #void: {} {
276     Negate() (<MUL> Negate() #Mul(2) | <DIV> Negate() #Div(2) )*
277 }
278
279 void Negate() #void: {} {
280     (<NOT>)* Length() #Negate
281 }
282
283 void Length() #void: {} {
284     Parentheses() (<DOT> Call() )*

```

```

285 }
286
287 void Parentheses() #void: {} {
288     ( <LEFT_PARENTHESSES> Expression() <RIGHT_PARENTHESSES> ) |
289     ( <LEFT_BRACKET> Expression() <RIGHT_BRACKET> ) |
290     _Expression()
291 }
292
293 void Call(): {} {
294     "length" |
295     (<IDENTIFIER> <LEFT_PARENTHESSES> [ Expression() ( <COMMA> Expression() )* ] <RIGHT_PARENTHESSES> )
296 }
297
298 void _Expression() #void: {} {
299     (
300         <INTEGER> #Int
301         | <TRUE> #True
302         | <FALSE> #False
303         | <IDENTIFIER> #Identifier
304         | <THIS> #This
305         | NewExpression() #NewExpression
306     )
307     __Expression()
308 }
309
310 void NewExpression(): {} {
311     <NEW>
312     (
313         <INT> <LEFT_BRACKET> Expression() #IntArray <RIGHT_BRACKET> |
314         <IDENTIFIER> #VarCreation <LEFT_PARENTHESSES> <RIGHT_PARENTHESSES>
315     )
316 }
317
318 void __Expression() #Index: {} {
319     [ <LEFT_BRACKET> Expression() <RIGHT_BRACKET> __Expression() ]
320 }
321

```