```
 1 package test;
 2
 3 import static org.junit.Assert.fail;
 4
 5 import java.io.StringReader;
 6
 7 import lexer.Lexer;
 8
 9 import org.junit.Test;
10
11 import parser.Parser;
12
13 public class ParserTests {
14     private void runtest(String src) {
15         runtest(src, true);
16     }
17
18     private void runtest(String src, boolean succeed) {
19         Parser parser = new Parser();
20         try {
21             parser.parse(new Lexer(new StringReader(src)));
22             if(!succeed) {
23                 fail("Test was supposed to fail, but succeeded");
24             }
25         } catch (beaver.Parser.Exception e) {
26             if(succeed) {
27                 e.printStackTrace();
28                 fail(e.getMessage());
29             }
30         } catch (Throwable e) {
31             e.printStackTrace();
32             fail(e.getMessage());
33         }
34     }
35
36     @Test
37     public void testModule() {
38         runtest("module Test { }", true);
39         runtest("module Foo {}", true);
40         runtest("module Bar { import foo; import bar; import baz; void
    func() {return;}}", true);
41         runtest("module {}", false);
42     }
43     @Test
44     public void testImportStmts() {
45         runtest("module Bar { import foo; import bar; import baz;}",
   true);
46         runtest("module Bar { import foo; import; import baz;}", false
   );
47         runtest("module Bar { foo; import foo; import baz;}", false);
```

```
48      }
49      @Test
50      public void testFuncDeclarations() {
51          runtest("module Bar { void () {return;} }", false);
52          runtest("module Bar { void foo() {return;}}", true);
53          runtest("module Bar { int bar() {}}", true);
54          runtest("module Bar { int bar(int a, String b, Int[] c, string
    [] d) {}}", true);
55          runtest("module Bar { int bar(int a, String b, int[] c, string
    [] d) {}}", false);
56          runtest("module Bar { int bar(int a, String b, int[] c,) {}}",
     false);
57          runtest("module Bar { int bar(int a, String b, int[] c {}}",
    false);
58          runtest("module Bar { public int bar(int a, String b) {}}",
    true);
59          runtest("module Bar { publc int bar(int a, String b) {}}",
    false);
60      }
61      @Test
62      public void testFieldsDeclarations() {
63          runtest("module Bar { public int a;}", true);
64          runtest("module Bar { String b;}", true);
65          runtest("module Bar { String[] b;}", true);
66          runtest("module Bar { pubd int b;}", false);
67          runtest("module Bar { public int b}", false);
68      }
69      @Test
70      public void testTypeDeclarations() {
71          runtest("module Bar { type FooInt = \"something\";}", true);
72          runtest("module Bar { type FooInt = 123;}", false);
73      }
74      @Test
75      public void testLocalVariableAssignments() {
76          runtest("module Bar { void foo(){int a;}}", true);
77          runtest("module Bar { void foo(){int a; a=2;}}", true);
78          runtest("module Bar { void foo(){int a = b;}}", false);
79          runtest("module Bar { void foo(){String c =;}}", false);
80          runtest("module Bar { void foo(){String c; c = \"qwerty\";}}",
     true);
81          runtest("module Bar { void foo(){String[] d; d = [1,2,3];}}",
    true);
82      }
83      @Test
84      public void testRetrnStatement() {
85          runtest("module Bar { String foo(){return a;}}", true);
86          runtest("module Bar { String foo(){return;}}", true);
87          runtest("module Bar { String foo(){return String;}}", true);
88          runtest("module Bar { String foo(){return a, b;}}", false);
89      }
```

```
 90     @Test
 91     public void testBlockStatement() {
 92         runtest("module Bar { void foo(){ int a; {} }}", true);
 93         runtest("module Bar { void foo(){ int a; {{}}{} }}", true);
 94         runtest("module Bar { void foo(){ int a; {{}}{int b; b=3
    ;} }}", true);
 95         runtest("module Bar { void foo(){ int a; {{} }}", false);
 96     }
 97     @Test
 98     public void testIfStatement() {
 99         runtest("module Bar { void foo() { if(true) {} else {}}}",
    true);
100         runtest("module Bar { void foo() { if(true) {}}}", false);
101         runtest("module Bar { void foo() { if(true) {return;} else
     {}}}", true);
102         runtest("module Bar { void foo() { if(foo) {} else {}}}",
    true);
103         runtest("module Bar { void foo() { if(1) {} else {}}}", true)
    ;
104 
105     }
106     @Test
107     public void testWhileStatement() {
108         runtest("module Bar { void foo(){ while(true){ }}}", true);
109         runtest("module Bar { void foo(){ while(true){ int a; a=3
    ;}}}", true);
110         runtest("module Bar { void foo(){ while(){ int a; a=3;}}}",
    false);
111     }
112     @Test
113     public void testBreakStatement() {
114         runtest("module Bar { void foo(){ break;}}", true);
115         runtest("module Bar { void foo(){ break}}", false);
116     }
117     @Test
118     public void testExprStatement() {
119         runtest("module Bar { void foo(){ a == 1;}}", true);
120         runtest("module Bar { void foo(){ a != 1;}}", true);
121         runtest("module Bar { void foo(){ a < 1;}}", true);
122         runtest("module Bar { void foo(){ a <= 1;}}", true);
123         runtest("module Bar { void foo(){ a > 1;}}", true);
124         runtest("module Bar { void foo(){ a >= 1;}}", true);
125         runtest("module Bar { void foo(){ a=a+1;}}", true);
126         runtest("module Bar { void foo(){ a=a-1;}}", true);
127         runtest("module Bar { void foo(){ a=a++;}}", false);
128         runtest("module Bar { void foo(){ a=a*b;}}", true);
129         runtest("module Bar { void foo(){ a=a/1;}}", true);
130         runtest("module Bar { void foo(){ a=a%5;}}", true);
131         runtest("module Bar { void foo(){ foo();}}", true);
132         runtest("module Bar { void foo(){ bar(1,2,3);}}", true);
```

```
133         runtest("module Bar { void foo(){ foo(a,b,\"abc\");}}", true)
    ;
134         runtest("module Bar { void foo(){ a=(a+(b/(c*(d-(1%(2
    ))))));}}", true);
135         runtest("module Bar { void foo(){ a=a+(foo+());}}", false);
136         runtest("module Bar { void foo(){ a=a+\"foo\";}}", true);
137         runtest("module Bar { void foo(){ a=true;}}", true);
138         runtest("module Bar { void foo(){ a=false;}}", true);
139         runtest("module Bar { void foo(){ a=true+false;}}", true);
140         runtest("module Bar { void foo(){ true=false;}}", false);
141         runtest("module Bar { void foo(){ true=a;}}", false);
142     }
143 }
144
```