

```

1 import static org.junit.Assert.assertEquals;
10
11 /**
12  * JUnit test fixture for {@code Statement}'s constructor and kernel methods.
13  *
14  * @author Chloe Feller and Krish Patel
15  *
16  */
17 public abstract class StatementTest {
18
19     /**
20      * The name of a file containing a sequence of BL statements.
21      */
22     private static final String FILE_NAME_1 = "test/statement/statement1.bl",
23         FILE_NAME_2 = "test/statement/statement2.bl",
24         FILE_NAME_3 = "test/statement/statement3.bl",
25         FILE_NAME_4 = "test/statement/statement4.bl";
26
27     /**
28      * Invokes the {@code Statement} constructor for the implementation under
29      * test and returns the result.
30      *
31      * @return the new statement
32      * @ensures constructorTest = compose((BLOCK, ?, ?), <>)
33      */
34     protected abstract Statement constructorTest();
35
36     /**
37      * Invokes the {@code Statement} constructor for the reference
38      * implementation and returns the result.
39      *
40      * @return the new statement
41      * @ensures constructorRef = compose((BLOCK, ?, ?), <>)
42      */
43     protected abstract Statement constructorRef();
44
45     /**
46      * Test of parse on syntactically valid input.
47      */
48     @Test
49     public final void testParseValidExample() {
50         /**
51          * Setup
52          */
53         Statement sRef = this.constructorRef();
54         SimpleReader file = new SimpleReader1L(FILE_NAME_1);
55         Queue<String> tokens = Tokenizer.tokens(file);
56         sRef.parse(tokens);
57         file.close();
58         Statement sTest = this.constructorTest();
59         file = new SimpleReader1L(FILE_NAME_1);
60         tokens = Tokenizer.tokens(file);
61         file.close();
62         /**
63          * The call
64          */
65         sTest.parse(tokens);

```

```
66      /*
67      * Evaluation
68      */
69      assertEquals(sRef, sTest);
70  }
71
72  /**
73   * Test of parse on syntactically invalid input.
74   */
75  @Test(expected = RuntimeException.class)
76  public final void testParseErrorExample() {
77      /*
78       * Setup
79       */
80      Statement sTest = this.constructorTest();
81      SimpleReader file = new SimpleReader1L(FILE_NAME_2);
82      Queue<String> tokens = Tokenizer.tokens(file);
83      file.close();
84      /*
85       * The call--should result in an error being caught
86       */
87      sTest.parse(tokens);
88  }
89
90  /**
91   * Test of parse on syntactically invalid input.
92   */
93  @Test(expected = RuntimeException.class)
94  public final void testParseFile3() {
95      /*
96       * Setup
97       */
98      Statement sTest = this.constructorTest();
99      SimpleReader file = new SimpleReader1L(FILE_NAME_3);
100     Queue<String> tokens = Tokenizer.tokens(file);
101     file.close();
102     /*
103      * The call--should result in an error being caught
104      */
105     sTest.parse(tokens);
106 }
107
108 /**
109  * Test of parse on syntactically invalid input.
110  */
111  @Test(expected = RuntimeException.class)
112  public final void testParseFile4() {
113      /*
114       * Setup
115       */
116      Statement sTest = this.constructorTest();
117      SimpleReader file = new SimpleReader1L(FILE_NAME_4);
118      Queue<String> tokens = Tokenizer.tokens(file);
119      file.close();
120      /*
121       * The call--should result in an error being caught
122       */
```

```
123         sTest.parse(tokens);  
124     }  
125  
126 }  
127
```