

```

1  import static org.junit.Assert.assertEquals;
10
11 /**
12  * JUnit test fixture for {@code Program}'s constructor and kernel methods.
13  *
14  * @author Put your name here
15  *
16  */
17 public abstract class ProgramTest {
18
19     /**
20      * The names of a files containing a (possibly invalid) BL programs.
21      */
22     private static final String FILE_NAME_1 = "test/program1.bl",
23         FILE_NAME_2 = "test/program2.bl", FILE_NAME_3 = "test/program3.bl",
24         FILE_NAME_4 = "test/program4.bl", FILE_NAME_5 = "test/program5.bl",
25         FILE_NAME_6 = "test/program6.bl", FILE_NAME_7 = "test/program7.bl",
26         FILE_NAME_8 = "test/program8.bl", FILE_NAME_9 = "test/program9.bl";
27
28     /**
29      * Invokes the {@code Program} constructor for the implementation under test
30      * and returns the result.
31      *
32      * @return the new program
33      * @ensures constructorTest = ("Unnamed", {}, compose((BLOCK, ?, ?), <>))
34      */
35     protected abstract Program constructorTest();
36
37     /**
38      * Invokes the {@code Program} constructor for the reference implementation
39      * and returns the result.
40      *
41      * @return the new program
42      * @ensures constructorRef = ("Unnamed", {}, compose((BLOCK, ?, ?), <>))
43      */
44     protected abstract Program constructorRef();
45
46     /**
47      * Test of parse on syntactically valid input.
48      */
49     @Test
50     public final void testParseValid1() {
51         /**
52          * Setup
53          */
54         Program pRef = this.constructorRef();
55         SimpleReader file = new SimpleReader1L(FILE_NAME_1);
56         pRef.parse(file);
57         file.close();
58         Program pTest = this.constructorTest();
59         file = new SimpleReader1L(FILE_NAME_1);
60         Queue<String> tokens = Tokenizer.tokens(file);
61         file.close();
62         /**
63          * The call
64          */
65         pTest.parse(tokens);
66         /**
67          * Evaluation
68          */
69         assertEquals(pRef, pTest);
70     }
71
72     /**
73      * Test of parse on syntactically invalid input.
74      */
75     @Test(expected = RuntimeException.class)
76     public final void testParseError2() {
77         /**
78          * Setup
79          */
80         Program pTest = 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 a syntax error being found
86          */
87         pTest.parse(tokens);
88     }
89
90     /**
91      * Test of parse on syntactically invalid input.

```

```

92  */
93  @Test(expected = RuntimeException.class)
94  public final void testParseError3() {
95      /*
96       * Setup
97       */
98      Program pTest = 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 a syntax error being found
104      */
105     pTest.parse(tokens);
106 }
107
108 /**
109  * Test of parse on syntactically invalid input.
110  */
111  @Test(expected = RuntimeException.class)
112  public final void testParseError4() {
113      /*
114       * Setup
115       */
116      Program pTest = 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 a syntax error being found
122       */
123      pTest.parse(tokens);
124  }
125
126  /**
127   * Test of parse on syntactically valid input.
128   */
129  @Test
130  public final void testParseValid5() {
131      /*
132       * Setup
133       */
134      Program pRef = this.constructorRef();
135      SimpleReader file = new SimpleReader1L(FILE_NAME_5);
136      pRef.parse(file);
137      file.close();
138      Program pTest = this.constructorTest();
139      file = new SimpleReader1L(FILE_NAME_5);
140      Queue<String> tokens = Tokenizer.tokens(file);
141      file.close();
142      /*
143       * The call
144       */
145      pTest.parse(tokens);
146      /*
147       * Evaluation
148       */
149      assertEquals(pRef, pTest);
150  }
151
152  /**
153   * Test of parse on syntactically invalid input.
154   */
155  @Test(expected = RuntimeException.class)
156  public final void testParseError6() {
157      /*
158       * Setup
159       */
160      Program pTest = this.constructorTest();
161      SimpleReader file = new SimpleReader1L(FILE_NAME_6);
162      Queue<String> tokens = Tokenizer.tokens(file);
163      file.close();
164      /*
165       * The call--should result in a syntax error being found
166       */
167      pTest.parse(tokens);
168  }
169
170  /**
171   * Test of parse on syntactically invalid input.
172   */
173  @Test(expected = RuntimeException.class)
174  public final void testParseError7() {

```

```

175     /*
176     * Setup
177     */
178     Program pTest = this.constructorTest();
179     SimpleReader file = new SimpleReader1L(FILE_NAME_7);
180     Queue<String> tokens = Tokenizer.tokens(file);
181     file.close();
182     /*
183     * The call--should result in a syntax error being found
184     */
185     pTest.parse(tokens);
186 }
187
188 /**
189  * Test of parse on syntactically invalid input.
190  */
191 @Test(expected = RuntimeException.class)
192 public final void testParseError8() {
193     /*
194     * Setup
195     */
196     Program pTest = this.constructorTest();
197     SimpleReader file = new SimpleReader1L(FILE_NAME_8);
198     Queue<String> tokens = Tokenizer.tokens(file);
199     file.close();
200     /*
201     * The call--should result in a syntax error being found
202     */
203     pTest.parse(tokens);
204 }
205
206 /**
207  * Test of parse on syntactically invalid input.
208  */
209 @Test(expected = RuntimeException.class)
210 public final void testParseError9() {
211     /*
212     * Setup
213     */
214     Program pTest = this.constructorTest();
215     SimpleReader file = new SimpleReader1L(FILE_NAME_9);
216     Queue<String> tokens = Tokenizer.tokens(file);
217     file.close();
218     /*
219     * The call--should result in a syntax error being found
220     */
221     pTest.parse(tokens);
222 }
223 // TODO - add more test cases for valid inputs
224 // TODO - add more test cases for as many distinct syntax errors as possible
225 }
226 }
227

```