THE OHIO STATE UNIVERSITY

PROJECT 8: PROGRAM AND STATEMENT PARSE

Daniil Gofman

Ansh Pachauri

SW 2: Dev & Dsgn

Paolo Bucci

Yiyang Chen

Shivam Gupta

November 14, 2023

```
 1 import static org.junit.Assert.assertEquals;
 2
 3 import org.junit.Test;
 4
 5 import components.queue.Queue;
 6 import components.simplereader.SimpleReader;
 7 import components.simplereader.SimpleReader1L;
 8 import components.statement.Statement;
 9 import components.utilities.Tokenizer;
10
11 /**
12  * JUnit test fixture for {@code Statement}'s constructor and kernel
    methods.
13  *
14  * @author Daniil Gofman and Ansh Pachauri
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/statement1.bl",
23             FILE_NAME_2 = "test/statement2.bl",
24             FILE_NAME_3 = "test/statement3.bl",
25             FILE_NAME_4 = "test/statement4.bl",
26             FILE_NAME_5 = "test/statement5.bl",
27             FILE_NAME_6 = "test/statement6.bl",
28             FILE_NAME_7 = "test/statement7.bl",
29             FILE_NAME_8 = "test/statement8.bl";
30
31     /**
32      * Invokes the {@code Statement} constructor for the implementation
    under
33      * test and returns the result.
34      *
35      * @return the new statement
36      * @ensures constructorTest = compose((BLOCK, ?, ?), <>)
37      */
38     protected abstract Statement constructorTest();
39
40     /**
41      * Invokes the {@code Statement} constructor for the reference
42      * implementation and returns the result.
43      *
44      * @return the new statement
45      * @ensures constructorRef = compose((BLOCK, ?, ?), <>)
46      */
47     protected abstract Statement constructorRef();
48
```

```
49      /**
50       * Test of parse on syntactically valid input.
51       */
52      @Test
53      public final void testParseValidExample() {
54          /*
55           * Setup
56           */
57          Statement sRef = this.constructorRef();
58          SimpleReader file = new SimpleReader1L(FILE_NAME_1);
59          Queue<String> tokens = Tokenizer.tokens(file);
60          sRef.parse(tokens);
61          file.close();
62          Statement sTest = this.constructorTest();
63          file = new SimpleReader1L(FILE_NAME_1);
64          tokens = Tokenizer.tokens(file);
65          file.close();
66          /*
67           * The call
68           */
69          sTest.parse(tokens);
70          /*
71           * Evaluation
72           */
73          assertEquals(sRef, sTest);
74      }
75
76      /**
77       * Test of parse on syntactically invalid input.
78       */
79      @Test(expected = RuntimeException.class)
80      public final void testParseErrorExample() {
81          /*
82           * Setup
83           */
84          Statement sTest = this.constructorTest();
85          SimpleReader file = new SimpleReader1L(FILE_NAME_2);
86          Queue<String> tokens = Tokenizer.tokens(file);
87          file.close();
88          /*
89           * The call--should result in an error being caught
90           */
91          sTest.parse(tokens);
92      }
93
94      /**
95       * Test of parse on syntactically invalid input.
96       */
97      @Test(expected = RuntimeException.class)
98      public final void testParseErrorNotCompleteLoop() {
```

```
 99          /*
100           * Setup
101           */
102          Statement sTest = this.constructorTest();
103          SimpleReader file = new SimpleReader1L(FILE_NAME_3);
104          Queue<String> tokens = Tokenizer.tokens(file);
105          file.close();
106          /*
107           * The call--should result in an error being caught
108           */
109          sTest.parse(tokens);
110      }
111
112      /**
113       * Test of parse on syntactically invalid input.
114       */
115      @Test(expected = RuntimeException.class)
116      public final void testParseErrorPythonSyntax() {
117          /*
118           * Setup
119           */
120          Statement sTest = this.constructorTest();
121          SimpleReader file = new SimpleReader1L(FILE_NAME_4);
122          Queue<String> tokens = Tokenizer.tokens(file);
123          file.close();
124          /*
125           * The call--should result in an error being caught
126           */
127          sTest.parse(tokens);
128      }
129
130      /**
131       * Test of parse on syntactically invalid input.
132       */
133      @Test(expected = RuntimeException.class)
134      public final void testParseErrorJavaStyle() {
135          /*
136           * Setup
137           */
138          Statement sTest = this.constructorTest();
139          SimpleReader file = new SimpleReader1L(FILE_NAME_5);
140          Queue<String> tokens = Tokenizer.tokens(file);
141          file.close();
142          /*
143           * The call--should result in an error being caught
144           */
145          sTest.parse(tokens);
146      }
147
148      /**
```

```
149     * Test of parse on syntactically valid input.
150     */
151    @Test
152    public final void testParseValidEmptyLoop() {
153        /*
154         * Setup
155         */
156        Statement sRef = this.constructorRef();
157        SimpleReader file = new SimpleReader1L(FILE_NAME_6);
158        Queue<String> tokens = Tokenizer.tokens(file);
159        sRef.parse(tokens);
160        file.close();
161        Statement sTest = this.constructorTest();
162        file = new SimpleReader1L(FILE_NAME_6);
163        tokens = Tokenizer.tokens(file);
164        file.close();
165        /*
166         * The call
167         */
168        sTest.parse(tokens);
169        /*
170         * Evaluation
171         */
172        assertEquals(sRef, sTest);
173    }
174
175    /**
176     * Test of parse on syntactically valid input.
177     */
178    @Test
179    public final void testParseValidBestBugLogic() {
180        /*
181         * Setup
182         */
183        Statement sRef = this.constructorRef();
184        SimpleReader file = new SimpleReader1L(FILE_NAME_7);
185        Queue<String> tokens = Tokenizer.tokens(file);
186        sRef.parse(tokens);
187        file.close();
188        Statement sTest = this.constructorTest();
189        file = new SimpleReader1L(FILE_NAME_7);
190        tokens = Tokenizer.tokens(file);
191        file.close();
192        /*
193         * The call
194         */
195        sTest.parse(tokens);
196        /*
197         * Evaluation
198         */
```

```
199         assertEquals(sRef, sTest);
200     }
201 
202     /**
203      * Test of parse on syntactically invalid input.
204      */
205     @Test(expected = RuntimeException.class)
206     public final void testParseErrorIncorrectCommand() {
207         /*
208          * Setup
209          */
210         Statement sTest = this.constructorTest();
211         SimpleReader file = new SimpleReader1L(FILE_NAME_8);
212         Queue<String> tokens = Tokenizer.tokens(file);
213         file.close();
214         /*
215          * The call--should result in an error being caught
216          */
217         sTest.parse(tokens);
218     }
219 }
220 
```