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.program.Program;
 6 import components.queue.Queue;
 7 import components.simplereader.SimpleReader;
 8 import components.simplereader.SimpleReader1L;
 9 import components.utilities.Tokenizer;
10
11 /**
12  * JUnit test fixture for {@code Program}'s constructor and kernel methods.
13  *
14  * @author Daniil Gofman and Ansh Pachauri
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             FILE_NAME_10 = "test/program10.bl",
28             FILE_NAME_11 = "test/program11.bl",
29             FILE_NAME_12 = "test/program12.bl";
30
31     /**
32      * Invokes the {@code Program} constructor for the implementation under
   test
33      * and returns the result.
34      *
35      * @return the new program
36      * @ensures constructorTest = ("Unnamed", {}, compose((BLOCK, ?, ?),
   <>))
37      */
38     protected abstract Program constructorTest();
39
40     /**
41      * Invokes the {@code Program} constructor for the reference
   implementation
42      * and returns the result.
43      *
```

```
44      * @return the new program
45      * @ensures constructorRef = ("Unnamed", {}, compose((BLOCK, ?, ?), <>))
46      */
47     protected abstract Program constructorRef();
48
49     /**
50      * Test of parse on syntactically valid input.
51      */
52     @Test
53     public final void testParseValidExample() {
54         /*
55          * Setup
56          */
57         Program pRef = this.constructorRef();
58         SimpleReader file = new SimpleReader1L(FILE_NAME_1);
59         pRef.parse(file);
60         file.close();
61         Program pTest = this.constructorTest();
62         file = new SimpleReader1L(FILE_NAME_1);
63         Queue<String> tokens = Tokenizer.tokens(file);
64         file.close();
65         /*
66          * The call
67          */
68         pTest.parse(tokens);
69         /*
70          * Evaluation
71          */
72         assertEquals(pRef, pTest);
73     }
74
75     /**
76      * Test of parse on syntactically invalid input.
77      */
78     @Test(expected = RuntimeException.class)
79     public final void testParseErrorExample() {
80         /*
81          * Setup
82          */
83         Program pTest = this.constructorTest();
84         SimpleReader file = new SimpleReader1L(FILE_NAME_2);
85         Queue<String> tokens = Tokenizer.tokens(file);
86         file.close();
87         /*
88          * The call--should result in a syntax error being found
89          */
90         pTest.parse(tokens);
91     }
92
93     /**
```

```
 94     * Test of parse on syntactically invalid input.
 95     */
 96    @Test(expected = RuntimeException.class)
 97    public final void testParseErrorNonExistingGrammar() {
 98        /*
 99         * Setup
100         */
101        Program pTest = this.constructorTest();
102        SimpleReader file = new SimpleReader1L(FILE_NAME_3);
103        Queue<String> tokens = Tokenizer.tokens(file);
104        file.close();
105        /*
106         * The call--should result in a syntax error being found
107         */
108        pTest.parse(tokens);
109    }
110
111    /**
112     * Test of parse on syntactically invalid input.
113     */
114    @Test
115    public final void testParseValidEmptyInstruction() {
116        /*
117         * Setup
118         */
119        Program pRef = this.constructorRef();
120        SimpleReader file = new SimpleReader1L(FILE_NAME_4);
121        pRef.parse(file);
122        file.close();
123        Program pTest = this.constructorTest();
124        file = new SimpleReader1L(FILE_NAME_4);
125        Queue<String> tokens = Tokenizer.tokens(file);
126        file.close();
127        /*
128         * The call
129         */
130        pTest.parse(tokens);
131        /*
132         * Evaluation
133         */
134        assertEquals(pRef, pTest);
135    }
136
137    /**
138     * Test of parse on syntactically invalid input.
139     */
140    @Test(expected = RuntimeException.class)
141    public final void testParseErrorNonCorrectSyntax() {
142        /*
143         * Setup
```

```
144          */
145         Program pTest = this.constructorTest();
146         SimpleReader file = new SimpleReader1L(FILE_NAME_5);
147         Queue<String> tokens = Tokenizer.tokens(file);
148         file.close();
149         /*
150          * The call--should result in a syntax error being found
151          */
152         pTest.parse(tokens);
153     }
154 
155     /**
156      * Test of parse on syntactically invalid input.
157      */
158     @Test(expected = RuntimeException.class)
159     public final void testParseErrorNoBody() {
160         /*
161          * Setup
162          */
163         Program pTest = this.constructorTest();
164         SimpleReader file = new SimpleReader1L(FILE_NAME_6);
165         Queue<String> tokens = Tokenizer.tokens(file);
166         file.close();
167         /*
168          * The call--should result in a syntax error being found
169          */
170         pTest.parse(tokens);
171     }
172 
173     /**
174      * Test of parse on syntactically invalid input.
175      */
176     @Test
177     public final void testParseValidExampleBestBug() {
178         /*
179          * Setup
180          */
181         Program pRef = this.constructorRef();
182         SimpleReader file = new SimpleReader1L(FILE_NAME_7);
183         pRef.parse(file);
184         file.close();
185         Program pTest = this.constructorTest();
186         file = new SimpleReader1L(FILE_NAME_7);
187         Queue<String> tokens = Tokenizer.tokens(file);
188         file.close();
189         /*
190          * The call
191          */
192         pTest.parse(tokens);
193         /*
```

```
194         * Evaluation
195         */
196        assertEquals(pRef, pTest);
197    }
198
199    /**
200     * Test of parse on syntactically invalid input.
201     */
202    @Test(expected = RuntimeException.class)
203    public final void testParseErrorInstructionError() {
204        /*
205         * Setup
206         */
207        Program pTest = this.constructorTest();
208        SimpleReader file = new SimpleReader1L(FILE_NAME_8);
209        Queue<String> tokens = Tokenizer.tokens(file);
210        file.close();
211        /*
212         * The call--should result in a syntax error being found
213         */
214        pTest.parse(tokens);
215    }
216
217    /**
218     * Test of parse on syntactically invalid input.
219     */
220    @Test(expected = RuntimeException.class)
221    public final void testParseErrorNonMatchingNames() {
222        /*
223         * Setup
224         */
225        Program pTest = this.constructorTest();
226        SimpleReader file = new SimpleReader1L(FILE_NAME_9);
227        Queue<String> tokens = Tokenizer.tokens(file);
228        file.close();
229        /*
230         * The call--should result in a syntax error being found
231         */
232        pTest.parse(tokens);
233    }
234
235    /**
236     * Test of parse on syntactically invalid input.
237     */
238    @Test(expected = RuntimeException.class)
239    public final void testParseErrorNonMatchingNamesInstr() {
240        /*
241         * Setup
242         */
243        Program pTest = this.constructorTest();
```

```
244         SimpleReader file = new SimpleReader1L(FILE_NAME_10);
245         Queue<String> tokens = Tokenizer.tokens(file);
246         file.close();
247         /*
248          * The call--should result in a syntax error being found
249          */
250         pTest.parse(tokens);
251     }
252 
253     /**
254      * Test of parse on syntactically invalid input.
255      */
256     @Test(expected = RuntimeException.class)
257     public final void testParseErrorMatchingInstr() {
258         /*
259          * Setup
260          */
261         Program pTest = this.constructorTest();
262         SimpleReader file = new SimpleReader1L(FILE_NAME_11);
263         Queue<String> tokens = Tokenizer.tokens(file);
264         file.close();
265         /*
266          * The call--should result in a syntax error being found
267          */
268         pTest.parse(tokens);
269     }
270 
271     /**
272      * Test of parse on syntactically valid input.
273      */
274     @Test
275     public final void testParseValidEmptyProgram() {
276         /*
277          * Setup
278          */
279         Program pRef = this.constructorRef();
280         SimpleReader file = new SimpleReader1L(FILE_NAME_12);
281         pRef.parse(file);
282         file.close();
283         Program pTest = this.constructorTest();
284         file = new SimpleReader1L(FILE_NAME_12);
285         Queue<String> tokens = Tokenizer.tokens(file);
286         file.close();
287         /*
288          * The call
289          */
290         pTest.parse(tokens);
291         /*
292          * Evaluation
293          */
```

```
294         assertEquals(pRef, pTest);
295     }
296
297 }
298
```