

```
1 import static org.junit.Assert.assertEquals;
11
12 /**
13  * JUnit test fixture for {@code Program}'s constructor and kernel methods.
14  *
15  * @author Wayne Heym
16  * @author Krish Patel and Chloe Feller
17  *
18  */
19 public abstract class ProgramTest {
20
21     /**
22      * The name of a file containing a BL program.
23      */
24     private static final String FILE_NAME_1 = "data/program-sample.bl";
25
26     /**
27      * The name of a second file containing a BL program.
28      */
29     private static final String FILE_NAME_2 = "data/program-test1.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 constructor = ("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 constructor = ("Unnamed", {}, compose((BLOCK, ?, ?), <>))
46      */
47     protected abstract Program constructorRef();
48
49     /**
50      *
51      * Creates and returns a {@code Program}, of the type of the implementation
52      * under test, from the file with the given name.
53      *
54      * @param filename
55      *         the name of the file to be parsed to create the program
56      * @return the constructed program
57      * @ensures createFromFile = [the program as parsed from the file]
58      */
59     private Program createFromFileTest(String filename) {
60         Program p = this.constructorTest();
61         SimpleReader file = new SimpleReader1L(filename);
62         p.parse(file);
63         file.close();
64         return p;
65     }
66 }
```

```
67  /**
68   *
69   * Creates and returns a {@code Program}, of the reference implementation
70   * type, from the file with the given name.
71   *
72   * @param filename
73   *         the name of the file to be parsed to create the program
74   * @return the constructed program
75   * @ensures createFromFile = [the program as parsed from the file]
76   */
77  private Program createFromFileRef(String filename) {
78      Program p = this.constructorRef();
79      SimpleReader file = new SimpleReader1L(filename);
80      p.parse(file);
81      file.close();
82      return p;
83  }
84
85  /**
86   * Test constructor.
87   */
88  @Test
89  public final void testConstructor() {
90      /*
91       * Setup
92       */
93      Program pRef = this.constructorRef();
94
95      /*
96       * The call
97       */
98      Program pTest = this.constructorTest();
99
100     /*
101      * Evaluation
102      */
103     assertEquals(pRef, pTest);
104 }
105
106 /**
107  * Test name.
108  */
109  @Test
110  public final void testName() {
111      /*
112       * Setup
113       */
114      Program pTest = this.createFromFileTest(FILE_NAME_1);
115      Program pRef = this.createFromFileRef(FILE_NAME_1);
116
117      /*
118       * The call
119       */
120      String result = pTest.name();
121
122      /*
123       * Evaluation
```

```
124         */
125         assertEquals(pRef, pTest);
126         assertEquals("Test", result);
127     }
128
129     /**
130     * Test setName.
131     */
132     @Test
133     public final void testSetName() {
134         /*
135         * Setup
136         */
137         Program pTest = this.createFromFileTest(FILE_NAME_1);
138         Program pRef = this.createFromFileRef(FILE_NAME_1);
139         String newName = "Replacement";
140         pRef.setName(newName);
141
142         /*
143         * The call
144         */
145         pTest.setName(newName);
146
147         /*
148         * Evaluation
149         */
150         assertEquals(pRef, pTest);
151     }
152
153     /**
154     * Test setName.
155     */
156     @Test
157     public final void testSetNameTwo() {
158         /*
159         * Setup
160         */
161         Program pTest = this.createFromFileTest(FILE_NAME_2);
162         Program pRef = this.createFromFileRef(FILE_NAME_2);
163         String newName = "Replacement";
164         pRef.setName(newName);
165
166         /*
167         * The call
168         */
169         pTest.setName(newName);
170
171         /*
172         * Evaluation
173         */
174         assertEquals(pRef, pTest);
175     }
176
177     /**
178     * Test newContext.
179     */
180     @Test
```

```
181     public final void testNewContext() {
182         /*
183          * Setup
184          */
185         Program pTest = this.createFromFileTest(FILE_NAME_1);
186         Program pRef = this.createFromFileRef(FILE_NAME_1);
187         Map<String, Statement> cRef = pRef.newContext();
188
189         /*
190          * The call
191          */
192         Map<String, Statement> cTest = pTest.newContext();
193
194         /*
195          * Evaluation
196          */
197         assertEquals(pRef, pTest);
198         assertEquals(cRef, cTest);
199     }
200
201     /**
202      * Test newContext.
203      */
204     @Test
205     public final void testNewContextTwo() {
206         /*
207          * Setup
208          */
209         Program pTest = this.createFromFileTest(FILE_NAME_2);
210         Program pRef = this.createFromFileRef(FILE_NAME_2);
211         Map<String, Statement> cRef = pRef.newContext();
212
213         /*
214          * The call
215          */
216         Map<String, Statement> cTest = pTest.newContext();
217
218         /*
219          * Evaluation
220          */
221         assertEquals(pRef, pTest);
222         assertEquals(cRef, cTest);
223     }
224
225     /**
226      * Test swapContext.
227      */
228     @Test
229     public final void testSwapContext() {
230         /*
231          * Setup
232          */
233         Program pTest = this.createFromFileTest(FILE_NAME_1);
234         Program pRef = this.createFromFileRef(FILE_NAME_1);
235         Map<String, Statement> contextRef = pRef.newContext();
236         Map<String, Statement> contextTest = pTest.newContext();
237         String oneName = "one";
```

```
238     pRef.swapContext(contextRef);
239     Pair<String, Statement> oneRef = contextRef.remove(oneName);
240     /* contextRef now has just "two" */
241     pRef.swapContext(contextRef);
242     /* pRef's context now has just "two" */
243     contextRef.add(oneRef.key(), oneRef.value());
244     /* contextRef now has just "one" */
245
246     /* Make the reference call, replacing, in pRef, "one" with "two": */
247     pRef.swapContext(contextRef);
248
249     pTest.swapContext(contextTest);
250     Pair<String, Statement> oneTest = contextTest.remove(oneName);
251     /* contextTest now has just "two" */
252     pTest.swapContext(contextTest);
253     /* pTest's context now has just "two" */
254     contextTest.add(oneTest.key(), oneTest.value());
255     /* contextTest now has just "one" */
256
257     /*
258     * The call
259     */
260     pTest.swapContext(contextTest);
261
262     /*
263     * Evaluation
264     */
265     assertEquals(pRef, pTest);
266     assertEquals(contextRef, contextTest);
267 }
268
269 /**
270  * Test swapContext.
271  */
272 @Test
273 public final void testSwapContextTwo() {
274     /*
275     * Setup
276     */
277     Program pTest = this.createFromFileTest(FILE_NAME_2);
278     Program pRef = this.createFromFileRef(FILE_NAME_2);
279     Map<String, Statement> contextRef = pRef.newContext();
280     Map<String, Statement> contextTest = pTest.newContext();
281     String oneName = "testone";
282     pRef.swapContext(contextRef);
283     Pair<String, Statement> oneRef = contextRef.remove(oneName);
284     /* contextRef now has just "two" */
285     pRef.swapContext(contextRef);
286     /* pRef's context now has just "two" */
287     contextRef.add(oneRef.key(), oneRef.value());
288     /* contextRef now has just "one" */
289
290     /* Make the reference call, replacing, in pRef, "one" with "two": */
291     pRef.swapContext(contextRef);
292
293     pTest.swapContext(contextTest);
294     Pair<String, Statement> oneTest = contextTest.remove(oneName);
```

```
295     /* contextTest now has just "two" */
296     pTest.swapContext(contextTest);
297     /* pTest's context now has just "two" */
298     contextTest.add(oneTest.key(), oneTest.value());
299     /* contextTest now has just "one" */
300
301     /*
302     * The call
303     */
304     pTest.swapContext(contextTest);
305
306     /*
307     * Evaluation
308     */
309     assertEquals(pRef, pTest);
310     assertEquals(contextRef, contextTest);
311 }
312
313 /**
314  * Test newBody.
315  */
316 @Test
317 public final void testNewBody() {
318     /*
319     * Setup
320     */
321     Program pTest = this.createFromFileTest(FILE_NAME_1);
322     Program pRef = this.createFromFileRef(FILE_NAME_1);
323     Statement bRef = pRef.newBody();
324
325     /*
326     * The call
327     */
328     Statement bTest = pTest.newBody();
329
330     /*
331     * Evaluation
332     */
333     assertEquals(pRef, pTest);
334     assertEquals(bRef, bTest);
335 }
336
337 /**
338  * Test newBody.
339  */
340 @Test
341 public final void testNewBodyTwo() {
342     /*
343     * Setup
344     */
345     Program pTest = this.createFromFileTest(FILE_NAME_2);
346     Program pRef = this.createFromFileRef(FILE_NAME_2);
347     Statement bRef = pRef.newBody();
348
349     /*
350     * The call
351     */
352 }
```

```
352     Statement bTest = pTest.newBody();
353
354     /*
355     * Evaluation
356     */
357     assertEquals(pRef, pTest);
358     assertEquals(bRef, bTest);
359 }
360
361 /**
362  * Test swapBody.
363  */
364 @Test
365 public final void testSwapBody() {
366     /*
367     * Setup
368     */
369     Program pTest = this.createFromFileTest(FILE_NAME_1);
370     Program pRef = this.createFromFileRef(FILE_NAME_1);
371     Statement bodyRef = pRef.newBody();
372     Statement bodyTest = pTest.newBody();
373     pRef.swapBody(bodyRef);
374     Statement firstRef = bodyRef.removeFromBlock(0);
375     /* bodyRef now lacks the first statement */
376     pRef.swapBody(bodyRef);
377     /* pRef's body now lacks the first statement */
378     bodyRef.addToBlock(0, firstRef);
379     /* bodyRef now has just the first statement */
380
381     /* Make the reference call, replacing, in pRef, remaining with first: */
382     pRef.swapBody(bodyRef);
383
384     pTest.swapBody(bodyTest);
385     Statement firstTest = bodyTest.removeFromBlock(0);
386     /* bodyTest now lacks the first statement */
387     pTest.swapBody(bodyTest);
388     /* pTest's body now lacks the first statement */
389     bodyTest.addToBlock(0, firstTest);
390     /* bodyTest now has just the first statement */
391
392     /*
393     * The call
394     */
395     pTest.swapBody(bodyTest);
396
397     /*
398     * Evaluation
399     */
400     assertEquals(pRef, pTest);
401     assertEquals(bodyRef, bodyTest);
402 }
403
404 /**
405  * Test swapBody.
406  */
407 @Test
408 public final void testSwapBodyTwo() {
```

```
409      /*
410      * Setup
411      */
412      Program pTest = this.createFromFileTest(FILE_NAME_2);
413      Program pRef = this.createFromFileRef(FILE_NAME_2);
414      Statement bodyRef = pRef.newBody();
415      Statement bodyTest = pTest.newBody();
416      pRef.swapBody(bodyRef);
417      Statement firstRef = bodyRef.removeFromBlock(0);
418      /* bodyRef now lacks the first statement */
419      pRef.swapBody(bodyRef);
420      /* pRef's body now lacks the first statement */
421      bodyRef.addToBlock(0, firstRef);
422      /* bodyRef now has just the first statement */
423
424      /* Make the reference call, replacing, in pRef, remaining with first: */
425      pRef.swapBody(bodyRef);
426
427      pTest.swapBody(bodyTest);
428      Statement firstTest = bodyTest.removeFromBlock(0);
429      /* bodyTest now lacks the first statement */
430      pTest.swapBody(bodyTest);
431      /* pTest's body now lacks the first statement */
432      bodyTest.addToBlock(0, firstTest);
433      /* bodyTest now has just the first statement */
434
435      /*
436      * The call
437      */
438      pTest.swapBody(bodyTest);
439
440      /*
441      * Evaluation
442      */
443      assertEquals(pRef, pTest);
444      assertEquals(bodyRef, bodyTest);
445  }
446 }
447
```