```java
import static org.junit.Assert.assertEquals;

/**
 * JUnit test fixture for {@code Program}'s constructor and kernel methods.
 *
 * @author Wayne Heym
 * @author Zhuoyang Li
 *
 */
public abstract class ProgramTest {

    /**
     * The name of a file containing a BL program.
     */
    private static final String FILE_NAME_1 = "data/program-sample.bl";

    // TODO - define file names for additional test inputs
    /**
     * The name of a file containing a BL program.
     */
    private static final String FILE_NAME_2 = "data/program-sampleTest.bl";

    /**
     * Invokes the {@code Program} constructor for the implementation under test
     * and returns the result.
     *
     * @return the new program
     * @ensures constructor = ("Unnamed", {}, compose((BLOCK, ?, ?), <>))
     */
    protected abstract Program constructorTest();

    /**
     * Invokes the {@code Program} constructor for the reference implementation
     * and returns the result.
     *
     * @return the new program
     * @ensures constructor = ("Unnamed", {}, compose((BLOCK, ?, ?), <>))
     */
    protected abstract Program constructorRef();

```

```
50      /**
51       *
52       * Creates and returns a {@code Program}, of the type of
   the implementation
53       * under test, from the file with the given name.
54       *
55       * @param filename
56       *            the name of the file to be parsed to create
   the program
57       * @return the constructed program
58       * @ensures createFromFile = [the program as parsed from
   the file]
59       */
60      private Program createFromFileTest(String filename) {
61          Program p = this.constructorTest();
62          SimpleReader file = new SimpleReader1L(filename);
63          p.parse(file);
64          file.close();
65          return p;
66      }
67
68      /**
69       *
70       * Creates and returns a {@code Program}, of the
   reference implementation
71       * type, from the file with the given name.
72       *
73       * @param filename
74       *            the name of the file to be parsed to create
   the program
75       * @return the constructed program
76       * @ensures createFromFile = [the program as parsed from
   the file]
77       */
78      private Program createFromFileRef(String filename) {
79          Program p = this.constructorRef();
80          SimpleReader file = new SimpleReader1L(filename);
81          p.parse(file);
82          file.close();
83          return p;
84      }
85
86      /**
87       * Test constructor.
88       */
89      @Test
90      public final void testConstructor() {
```

```
 91         /*
 92          * Setup
 93          */
 94         Program pRef = this.constructorRef();
 95
 96         /*
 97          * The call
 98          */
 99         Program pTest = this.constructorTest();
100
101         /*
102          * Evaluation
103          */
104         assertEquals(pRef, pTest);
105     }
106
107     /**
108      * Test name.
109      */
110     @Test
111     public final void testName() {
112         /*
113          * Setup
114          */
115         Program pTest = this.createFromFileTest(FILE_NAME_1);
116         Program pRef = this.createFromFileRef(FILE_NAME_1);
117
118         /*
119          * The call
120          */
121         String result = pTest.name();
122
123         /*
124          * Evaluation
125          */
126         assertEquals(pRef, pTest);
127         assertEquals("Test", result);
128     }
129
130     /**
131      * Test setName.
132      */
133     @Test
134     public final void testSetName() {
135         /*
136          * Setup
137          */
```

```
138         Program pTest = this.createFromFileTest(FILE_NAME_1);
139         Program pRef = this.createFromFileRef(FILE_NAME_1);
140         String newName = "Replacement";
141         pRef.setName(newName);
142 
143         /*
144          * The call
145          */
146         pTest.setName(newName);
147 
148         /*
149          * Evaluation
150          */
151         assertEquals(pRef, pTest);
152     }
153 
154     /**
155      * Test newContext.
156      */
157     @Test
158     public final void testNewContext() {
159         /*
160          * Setup
161          */
162         Program pTest = this.createFromFileTest(FILE_NAME_1);
163         Program pRef = this.createFromFileRef(FILE_NAME_1);
164         Map<String, Statement> cRef = pRef.newContext();
165 
166         /*
167          * The call
168          */
169         Map<String, Statement> cTest = pTest.newContext();
170 
171         /*
172          * Evaluation
173          */
174         assertEquals(pRef, pTest);
175         assertEquals(cRef, cTest);
176     }
177 
178     /**
179      * Test swapContext.
180      */
181     @Test
182     public final void testSwapContext() {
183         /*
184          * Setup
```

```
185          */
186         Program pTest = this.createFromFileTest(FILE_NAME_1);
187         Program pRef = this.createFromFileRef(FILE_NAME_1);
188         Map<String, Statement> contextRef =
    pRef.newContext();
189         Map<String, Statement> contextTest =
    pTest.newContext();
190         String oneName = "one";
191         pRef.swapContext(contextRef);
192         Pair<String, Statement> oneRef =
    contextRef.remove(oneName);
193         /* contextRef now has just "two" */
194         pRef.swapContext(contextRef);
195         /* pRef's context now has just "two" */
196         contextRef.add(oneRef.key(), oneRef.value());
197         /* contextRef now has just "one" */
198 
199         /* Make the reference call, replacing, in pRef, "one"
    with "two": */
200         pRef.swapContext(contextRef);
201 
202         pTest.swapContext(contextTest);
203         Pair<String, Statement> oneTest =
    contextTest.remove(oneName);
204         /* contextTest now has just "two" */
205         pTest.swapContext(contextTest);
206         /* pTest's context now has just "two" */
207         contextTest.add(oneTest.key(), oneTest.value());
208         /* contextTest now has just "one" */
209 
210         /*
211          * The call
212          */
213         pTest.swapContext(contextTest);
214 
215         /*
216          * Evaluation
217          */
218         assertEquals(pRef, pTest);
219         assertEquals(contextRef, contextTest);
220     }
221 
222     /**
223      * Test newBody.
224      */
225     @Test
226     public final void testNewBody() {
```

```
227         /*
228          * Setup
229          */
230         Program pTest = this.createFromFileTest(FILE_NAME_1);
231         Program pRef = this.createFromFileRef(FILE_NAME_1);
232         Statement bRef = pRef.newBody();
233 
234         /*
235          * The call
236          */
237         Statement bTest = pTest.newBody();
238 
239         /*
240          * Evaluation
241          */
242         assertEquals(pRef, pTest);
243         assertEquals(bRef, bTest);
244     }
245 
246     /**
247      * Test swapBody.
248      */
249     @Test
250     public final void testSwapBody() {
251         /*
252          * Setup
253          */
254         Program pTest = this.createFromFileTest(FILE_NAME_1);
255         Program pRef = this.createFromFileRef(FILE_NAME_1);
256         Statement bodyRef = pRef.newBody();
257         Statement bodyTest = pTest.newBody();
258         pRef.swapBody(bodyRef);
259         Statement firstRef = bodyRef.removeFromBlock(0);
260         /* bodyRef now lacks the first statement */
261         pRef.swapBody(bodyRef);
262         /* pRef's body now lacks the first statement */
263         bodyRef.addToBlock(0, firstRef);
264         /* bodyRef now has just the first statement */
265 
266         /* Make the reference call, replacing, in pRef,
    remaining with first: */
267         pRef.swapBody(bodyRef);
268 
269         pTest.swapBody(bodyTest);
270         Statement firstTest = bodyTest.removeFromBlock(0);
271         /* bodyTest now lacks the first statement */
272         pTest.swapBody(bodyTest);
```

```
273         /* pTest's body now lacks the first statement */
274         bodyTest.addToBlock(0, firstTest);
275         /* bodyTest now has just the first statement */
276
277         /*
278          * The call
279          */
280         pTest.swapBody(bodyTest);
281
282         /*
283          * Evaluation
284          */
285         assertEquals(pRef, pTest);
286         assertEquals(bodyRef, bodyTest);
287     }
288
289     // TODO - provide additional test cases to thoroughly
    test ProgramKernel
290
291     /**
292      * Test name.
293      */
294     @Test
295     public final void testName2() {
296         /*
297          * Setup
298          */
299         Program pTest = this.createFromFileTest(FILE_NAME_2);
300         Program pRef = this.createFromFileRef(FILE_NAME_2);
301
302         /*
303          * The call
304          */
305         String result = pTest.name();
306
307         /*
308          * Evaluation
309          */
310         assertEquals(pRef, pTest);
311         assertEquals("Test2", result);
312     }
313
314     /**
315      * Test setName.
316      */
317     @Test
318     public final void testSetName2() {
```

```
319         /*
320          * Setup
321          */
322         Program pTest = this.createFromFileTest(FILE_NAME_2);
323         Program pRef = this.createFromFileRef(FILE_NAME_2);
324         String newName = "Replacement";
325         pRef.setName(newName);
326 
327         /*
328          * The call
329          */
330         pTest.setName(newName);
331 
332         /*
333          * Evaluation
334          */
335         assertEquals(pRef, pTest);
336     }
337 
338     /**
339      * Test newContext.
340      */
341     @Test
342     public final void testNewContext2() {
343         /*
344          * Setup
345          */
346         Program pTest = this.createFromFileTest(FILE_NAME_2);
347         Program pRef = this.createFromFileRef(FILE_NAME_2);
348         Map<String, Statement> cRef = pRef.newContext();
349 
350         /*
351          * The call
352          */
353         Map<String, Statement> cTest = pTest.newContext();
354 
355         /*
356          * Evaluation
357          */
358         assertEquals(pRef, pTest);
359         assertEquals(cRef, cTest);
360     }
361 
362     /**
363      * Test swapContext.
364      */
365     @Test
```

```
366     public final void testSwapContext2() {
367         /*
368          * Setup
369          */
370         Program pTest = this.createFromFileTest(FILE_NAME_2);
371         Program pRef = this.createFromFileRef(FILE_NAME_2);
372         Map<String, Statement> contextRef =
    pRef.newContext();
373         Map<String, Statement> contextTest =
    pTest.newContext();
374         String oneName = "one2";
375         pRef.swapContext(contextRef);
376         Pair<String, Statement> oneRef =
    contextRef.remove(oneName);
377         /* contextRef now has just "two" */
378         pRef.swapContext(contextRef);
379         /* pRef's context now has just "two" */
380         contextRef.add(oneRef.key(), oneRef.value());
381         /* contextRef now has just "one" */
382 
383         /* Make the reference call, replacing, in pRef, "one"
    with "two": */
384         pRef.swapContext(contextRef);
385 
386         pTest.swapContext(contextTest);
387         Pair<String, Statement> oneTest =
    contextTest.remove(oneName);
388         /* contextTest now has just "two" */
389         pTest.swapContext(contextTest);
390         /* pTest's context now has just "two" */
391         contextTest.add(oneTest.key(), oneTest.value());
392         /* contextTest now has just "one" */
393 
394         /*
395          * The call
396          */
397         pTest.swapContext(contextTest);
398 
399         /*
400          * Evaluation
401          */
402         assertEquals(pRef, pTest);
403         assertEquals(contextRef, contextTest);
404     }
405 
406     /**
407      * Test newBody.
```

```
408      */
409     @Test
410     public final void testNewBody2() {
411         /*
412          * Setup
413          */
414         Program pTest = this.createFromFileTest(FILE_NAME_2);
415         Program pRef = this.createFromFileRef(FILE_NAME_2);
416         Statement bRef = pRef.newBody();
417 
418         /*
419          * The call
420          */
421         Statement bTest = pTest.newBody();
422 
423         /*
424          * Evaluation
425          */
426         assertEquals(pRef, pTest);
427         assertEquals(bRef, bTest);
428     }
429 
430     /**
431      * Test swapBody.
432      */
433     @Test
434     public final void testSwapBody2() {
435         /*
436          * Setup
437          */
438         Program pTest = this.createFromFileTest(FILE_NAME_2);
439         Program pRef = this.createFromFileRef(FILE_NAME_2);
440         Statement bodyRef = pRef.newBody();
441         Statement bodyTest = pTest.newBody();
442         pRef.swapBody(bodyRef);
443         Statement firstRef = bodyRef.removeFromBlock(0);
444         /* bodyRef now lacks the first statement */
445         pRef.swapBody(bodyRef);
446         /* pRef's body now lacks the first statement */
447         bodyRef.addToBlock(0, firstRef);
448         /* bodyRef now has just the first statement */
449 
450         /* Make the reference call, replacing, in pRef,
    remaining with first: */
451         pRef.swapBody(bodyRef);
452 
453         pTest.swapBody(bodyTest);
```

```
454         Statement firstTest = bodyTest.removeFromBlock(0);
455         /* bodyTest now lacks the first statement */
456         pTest.swapBody(bodyTest);
457         /* pTest's body now lacks the first statement */
458         bodyTest.addToBlock(0, firstTest);
459         /* bodyTest now has just the first statement */
460
461         /*
462          * The call
463          */
464         pTest.swapBody(bodyTest);
465
466         /*
467          * Evaluation
468          */
469         assertEquals(pRef, pTest);
470         assertEquals(bodyRef, bodyTest);
471     }
472
473 }
474
```