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

/**
 * JUnit test fixture for {@code Program}'s constructor and kernel methods.
 *
 * @author Shyam Sai Bethina and Yihone Chu
 *
 */
public abstract class ProgramTest {

    /**
     * The name of a file containing a BL program.
     */
    private static final String FILE_NAME_1 = "data/program-sample1.bl";
    private static final String FILE_NAME_2 = "data/program-sample2.bl";
    private static final String FILE_NAME_3 = "data/program-sample3.bl";

    // TODO - define file names for additional test inputs

    /**
     * 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();
```

```
46
47      /**
48       *
49       * Creates and returns a {@code Program}, of the type of
    the implementation
50       * under test, from the file with the given name.
51       *
52       * @param filename
53       *            the name of the file to be parsed to create
    the program
54       * @return the constructed program
55       * @ensures createFromFile = [the program as parsed from
    the file]
56       */
57      private Program createFromFileTest(String filename) {
58          Program p = this.constructorTest();
59          SimpleReader file = new SimpleReader1L(filename);
60          p.parse(file);
61          file.close();
62          return p;
63      }
64
65      /**
66       *
67       * Creates and returns a {@code Program}, of the reference
    implementation
68       * type, from the file with the given name.
69       *
70       * @param filename
71       *            the name of the file to be parsed to create
    the program
72       * @return the constructed program
73       * @ensures createFromFile = [the program as parsed from
    the file]
74       */
75      private Program createFromFileRef(String filename) {
76          Program p = this.constructorRef();
77          SimpleReader file = new SimpleReader1L(filename);
78          p.parse(file);
79          file.close();
80          return p;
81      }
82
83      /**
```

```
 84      * Test constructor.
 85      */
 86     @Test
 87     public final void testConstructor() {
 88         /*
 89          * Setup
 90          */
 91         Program pRef = this.constructorRef();
 92 
 93         /*
 94          * The call
 95          */
 96         Program pTest = this.constructorTest();
 97 
 98         /*
 99          * Evaluation
100          */
101         assertEquals(pRef, pTest);
102     }
103 
104     /**
105      * Test name routine.
106      */
107     @Test
108     public final void testName1() {
109         /*
110          * Setup
111          */
112         Program pTest = this.createFromFileTest(FILE_NAME_1);
113         Program pRef = this.createFromFileRef(FILE_NAME_1);
114 
115         /*
116          * The call
117          */
118         String result = pTest.name();
119 
120         /*
121          * Evaluation
122          */
123         assertEquals(pRef, pTest);
124         assertEquals("Test", result);
125     }
126 
127     /**
```

```
128      * Test name edge.
129      */
130     @Test
131     public final void testName2() {
132         /*
133          * Setup
134          */
135         Program pTest = this.createFromFileTest(FILE_NAME_2);
136         Program pRef = this.createFromFileRef(FILE_NAME_2);
137 
138         /*
139          * The call
140          */
141         String result = pTest.name();
142 
143         /*
144          * Evaluation
145          */
146         assertEquals(pRef, pTest);
147         assertEquals("H", result);
148     }
149 
150     /**
151      * Test name challenging(all caps).
152      */
153     @Test
154     public final void testName3() {
155         /*
156          * Setup
157          */
158         Program pTest = this.createFromFileTest(FILE_NAME_3);
159         Program pRef = this.createFromFileRef(FILE_NAME_3);
160 
161         /*
162          * The call
163          */
164         String result = pTest.name();
165 
166         /*
167          * Evaluation
168          */
169         assertEquals(pRef, pTest);
170         assertEquals("HELLO", result);
171     }
```

```
172
173     /**
174      * Test setName routine.
175      */
176     @Test
177     public final void testSetName1() {
178         /*
179          * Setup
180          */
181         Program pTest = this.createFromFileTest(FILE_NAME_1);
182         Program pRef = this.createFromFileRef(FILE_NAME_1);
183         String newName = "Replacement";
184         pRef.setName(newName);
185 
186         /*
187          * The call
188          */
189         pTest.setName(newName);
190 
191         /*
192          * Evaluation
193          */
194         assertEquals(pRef, pTest);
195     }
196 
197     /**
198      * Test setName edge.
199      */
200     @Test
201     public final void testSetName2() {
202         /*
203          * Setup
204          */
205         Program pTest = this.createFromFileTest(FILE_NAME_2);
206         Program pRef = this.createFromFileRef(FILE_NAME_2);
207         String newName = "R";
208         pRef.setName(newName);
209 
210         /*
211          * The call
212          */
213         pTest.setName(newName);
214 
215         /*
```

```
216          * Evaluation
217          */
218         assertEquals(pRef, pTest);
219     }
220 
221     /**
222      * Test setName challenging.
223      */
224     @Test
225     public final void testSetName3() {
226         /*
227          * Setup
228          */
229         Program pTest = this.createFromFileTest(FILE_NAME_3);
230         Program pRef = this.createFromFileRef(FILE_NAME_3);
231         String newName = "HELLO-";
232         pRef.setName(newName);
233 
234         /*
235          * The call
236          */
237         pTest.setName(newName);
238 
239         /*
240          * Evaluation
241          */
242         assertEquals(pRef, pTest);
243     }
244 
245     /**
246      * Test newContext.
247      */
248     @Test
249     public final void testNewContext() {
250         /*
251          * Setup
252          */
253         Program pTest = this.createFromFileTest(FILE_NAME_1);
254         Program pRef = this.createFromFileRef(FILE_NAME_1);
255         Map<String, Statement> cRef = pRef.newContext();
256 
257         /*
258          * The call
259          */
```

```
260         Map<String, Statement> cTest = pTest.newContext();
261 
262         /*
263          * Evaluation
264          */
265         assertEquals(pRef, pTest);
266         assertEquals(cRef, cTest);
267     }
268     /*
269      * Didn't add anymore newContext test cases since the dynamic type is the
270      * same for all of the files. There is nothing else to test with newContext.
271      */
272 
273     /**
274      * Test swapContext routine.
275      */
276     @Test
277     public final void testSwapContext1() {
278         /*
279          * Setup
280          */
281         Program pTest = this.createFromFileTest(FILE_NAME_1);
282         Program pRef = this.createFromFileRef(FILE_NAME_1);
283         Map<String, Statement> contextRef = pRef.newContext();
284         Map<String, Statement> contextTest = pTest.newContext();
285         String oneName = "one";
286         pRef.swapContext(contextRef);
287         Pair<String, Statement> oneRef = contextRef.remove(oneName);
288         /* contextRef now has just "two" */
289         pRef.swapContext(contextRef);
290         /* pRef's context now has just "two" */
291         contextRef.add(oneRef.key(), oneRef.value());
292         /* contextRef now has just "one" */
293 
294         /* Make the reference call, replacing, in pRef, "one" with "two": */
295         pRef.swapContext(contextRef);
296 
297         pTest.swapContext(contextTest);
298         Pair<String, Statement> oneTest =
```

```
    contextTest.remove(oneName);
299         /* contextTest now has just "two" */
300         pTest.swapContext(contextTest);
301         /* pTest's context now has just "two" */
302         contextTest.add(oneTest.key(), oneTest.value());
303         /* contextTest now has just "one" */
304 
305         /*
306          * The call
307          */
308         pTest.swapContext(contextTest);
309 
310         /*
311          * Evaluation
312          */
313         assertEquals(pRef, pTest);
314         assertEquals(contextRef, contextTest);
315     }
316 
317     /**
318      * Test swapContext edge.
319      */
320     @Test
321     public final void testSwapContext2() {
322         /*
323          * Setup
324          */
325         Program pTest = this.createFromFileTest(FILE_NAME_2);
326         Program pRef = this.createFromFileRef(FILE_NAME_2);
327         Map<String, Statement> contextRef = pRef.newContext();
328         Map<String, Statement> contextTest =
    pTest.newContext();
329         String oneName = "one";
330         pRef.swapContext(contextRef);
331         Pair<String, Statement> oneRef =
    contextRef.remove(oneName);
332         /* contextRef now has just "two" */
333         pRef.swapContext(contextRef);
334         /* pRef's context now has just "two" */
335         contextRef.add(oneRef.key(), oneRef.value());
336         /* contextRef now has just "one" */
337 
338         /* Make the reference call, replacing, in pRef, "one"
    with "two": */
```

```
339         pRef.swapContext(contextRef);
340
341         pTest.swapContext(contextTest);
342         Pair<String, Statement> oneTest =
    contextTest.remove(oneName);
343         /* contextTest now has just "two" */
344         pTest.swapContext(contextTest);
345         /* pTest's context now has just "two" */
346         contextTest.add(oneTest.key(), oneTest.value());
347         /* contextTest now has just "one" */
348
349         /*
350          * The call
351          */
352         pTest.swapContext(contextTest);
353
354         /*
355          * Evaluation
356          */
357         assertEquals(pRef, pTest);
358         assertEquals(contextRef, contextTest);
359     }
360
361     /**
362      * Test swapContext challenging(using characters other than
    letters and
363      * numbers).
364      */
365     @Test
366     public final void testSwapContext3() {
367         /*
368          * Setup
369          */
370         Program pTest = this.createFromFileTest(FILE_NAME_3);
371         Program pRef = this.createFromFileRef(FILE_NAME_3);
372         Map<String, Statement> contextRef = pRef.newContext();
373         Map<String, Statement> contextTest =
    pTest.newContext();
374         String oneName = "ONE-";
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 testNewBody() {
411         /*
412          * Setup
413          */
414         Program pTest = this.createFromFileTest(FILE_NAME_1);
415         Program pRef = this.createFromFileRef(FILE_NAME_1);
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      * Didn't add anymore newContext test cases since the dynamic type is the
431      * same for all of the files. There is nothing else to test with newContext.
432      */
433 
434     /**
435      * Test swapBody routine.
436      */
437     @Test
438     public final void testSwapBody1() {
439         /*
440          * Setup
441          */
442         Program pTest = this.createFromFileTest(FILE_NAME_1);
443         Program pRef = this.createFromFileRef(FILE_NAME_1);
444         Statement bodyRef = pRef.newBody();
445         Statement bodyTest = pTest.newBody();
446         pRef.swapBody(bodyRef);
447         Statement firstRef = bodyRef.removeFromBlock(0);
448         /* bodyRef now lacks the first statement */
449         pRef.swapBody(bodyRef);
450         /* pRef's body now lacks the first statement */
451         bodyRef.addToBlock(0, firstRef);
452         /* bodyRef now has just the first statement */
453 
454         /* Make the reference call, replacing, in pRef, remaining with first: */
455         pRef.swapBody(bodyRef);
456 
457         pTest.swapBody(bodyTest);
458         Statement firstTest = bodyTest.removeFromBlock(0);
459         /* bodyTest now lacks the first statement */
460         pTest.swapBody(bodyTest);
461         /* pTest's body now lacks the first statement */
```

```
462         bodyTest.addToBlock(0, firstTest);
463         /* bodyTest now has just the first statement */
464 
465         /*
466          * The call
467          */
468         pTest.swapBody(bodyTest);
469 
470         /*
471          * Evaluation
472          */
473         assertEquals(pRef, pTest);
474         assertEquals(bodyRef, bodyTest);
475     }
476 
477     /**
478      * Test swapBody edge.
479      */
480     @Test
481     public final void testSwapBody2() {
482         /*
483          * Setup
484          */
485         Program pTest = this.createFromFileTest(FILE_NAME_2);
486         Program pRef = this.createFromFileRef(FILE_NAME_2);
487         Statement bodyRef = pRef.newBody();
488         Statement bodyTest = pTest.newBody();
489         pRef.swapBody(bodyRef);
490         Statement firstRef = bodyRef.removeFromBlock(0);
491         /* bodyRef now lacks the first statement */
492         pRef.swapBody(bodyRef);
493         /* pRef's body now lacks the first statement */
494         bodyRef.addToBlock(0, firstRef);
495         /* bodyRef now has just the first statement */
496 
497         /* Make the reference call, replacing, in pRef,
    remaining with first: */
498         pRef.swapBody(bodyRef);
499 
500         pTest.swapBody(bodyTest);
501         Statement firstTest = bodyTest.removeFromBlock(0);
502         /* bodyTest now lacks the first statement */
503         pTest.swapBody(bodyTest);
504         /* pTest's body now lacks the first statement */
```

```
505         bodyTest.addToBlock(0, firstTest);
506         /* bodyTest now has just the first statement */
507 
508         /*
509          * The call
510          */
511         pTest.swapBody(bodyTest);
512 
513         /*
514          * Evaluation
515          */
516         assertEquals(pRef, pTest);
517         assertEquals(bodyRef, bodyTest);
518     }
519 
520     /**
521      * Test swapBody challenging.
522      */
523     @Test
524     public final void testSwapBody3() {
525         /*
526          * Setup
527          */
528         Program pTest = this.createFromFileTest(FILE_NAME_2);
529         Program pRef = this.createFromFileRef(FILE_NAME_2);
530         Statement bodyRef = pRef.newBody();
531         Statement bodyTest = pTest.newBody();
532         pRef.swapBody(bodyRef);
533         Statement firstRef = bodyRef.removeFromBlock(0);
534         /* bodyRef now lacks the first statement */
535         pRef.swapBody(bodyRef);
536         /* pRef's body now lacks the first statement */
537         bodyRef.addToBlock(0, firstRef);
538         /* bodyRef now has just the first statement */
539 
540         /* Make the reference call, replacing, in pRef,
    remaining with first: */
541         pRef.swapBody(bodyRef);
542 
543         pTest.swapBody(bodyTest);
544         Statement firstTest = bodyTest.removeFromBlock(0);
545         /* bodyTest now lacks the first statement */
546         pTest.swapBody(bodyTest);
547         /* pTest's body now lacks the first statement */
```

```
548         bodyTest.addToBlock(0, firstTest);
549         /* bodyTest now has just the first statement */
550 
551         /*
552          * The call
553          */
554         pTest.swapBody(bodyTest);
555 
556         /*
557          * Evaluation
558          */
559         assertEquals(pRef, pTest);
560         assertEquals(bodyRef, bodyTest);
561     }
562 
563     // TODO - provide additional test cases to thoroughly test
    ProgramKernel
564 
565 }
566 
```