THE OHIO STATE UNIVERSITY

PROJECT 7: IMPLEMENTATION OF PROGRAM AND STATEMENT KERNELS

Daniil Gofman

Ansh Pachauri

SW 2: Dev & Dsgn

Paolo Bucci

Yiyang Chen

Shivam Gupta

November 1, 2023

```
 1 import components.program.Program;
 2 import components.program.Program1;
 3
 4 /**
 5  * Customized JUnit test fixture for {@code Program2}.
 6  */
 7 public class Program2Test extends ProgramTest {
 8
 9     @Override
10     protected final Program constructorTest() {
11         return new Program2();
12     }
13
14     @Override
15     protected final Program constructorRef() {
16         return new Program1();
17     }
18
19 }
20
```

```
 1 import static org.junit.Assert.assertEquals;
 2
 3 import org.junit.Test;
 4
 5 import components.map.Map;
 6 import components.map.Map.Pair;
 7 import components.program.Program;
 8 import components.simplereader.SimpleReader;
 9 import components.simplereader.SimpleReader1L;
10 import components.statement.Statement;
11
12 /**
13  * JUnit test fixture for {@code Program}'s constructor and
   kernel methods.
14  *
15  * @author Ansh Pachauri
16  * @author Daniil Gofman
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      * The name of a file containing a BL program.
27      */
28     private static final String FILE_NAME_2 = "data/program-
   sample2.bl";
29     /**
30      * The name of a file containing a BL program.
31      */
32     private static final String FILE_NAME_3 = "data/program-
   sample3.bl";
33
34     /**
35      * Invokes the {@code Program} constructor for the
```

```
   implementation under test
36      * and returns the result.
37      *
38      * @return the new program
39      * @ensures constructor = ("Unnamed", {},
   compose((BLOCK, ?, ?), <>))
40      */
41     protected abstract Program constructorTest();
42
43     /**
44      * Invokes the {@code Program} constructor for the
   reference implementation
45      * and returns the result.
46      *
47      * @return the new program
48      * @ensures constructor = ("Unnamed", {},
   compose((BLOCK, ?, ?), <>))
49      */
50     protected abstract Program constructorRef();
51
52     /**
53      *
54      * Creates and returns a {@code Program}, of the type of
   the implementation
55      * under test, from the file with the given name.
56      *
57      * @param filename
58      *            the name of the file to be parsed to create
   the program
59      * @return the constructed program
60      * @ensures createFromFile = [the program as parsed from
   the file]
61      */
62     private Program createFromFileTest(String filename) {
63         Program p = this.constructorTest();
64         SimpleReader file = new SimpleReader1L(filename);
65         p.parse(file);
66         file.close();
67         return p;
```

```
 68     }
 69
 70     /**
 71      *
 72      * Creates and returns a {@code Program}, of the reference
    implementation
 73      * type, from the file with the given name.
 74      *
 75      * @param filename
 76      *            the name of the file to be parsed to create
    the program
 77      * @return the constructed program
 78      * @ensures createFromFile = [the program as parsed from
    the file]
 79      */
 80     private Program createFromFileRef(String filename) {
 81         Program p = this.constructorRef();
 82         SimpleReader file = new SimpleReader1L(filename);
 83         p.parse(file);
 84         file.close();
 85         return p;
 86     }
 87
 88     /**
 89      * Test constructor.
 90      */
 91     @Test
 92     public final void testConstructor() {
 93         /*
 94          * Setup
 95          */
 96         Program pRef = this.constructorRef();
 97
 98         /*
 99          * The call
100          */
101         Program pTest = this.constructorTest();
102
103         /*
```

```
104          * Evaluation
105          */
106         assertEquals(pRef, pTest);
107     }
108 
109     /**
110      * Test name.
111      */
112     @Test
113     public final void testName() {
114         /*
115          * Setup
116          */
117         Program pTest = this.createFromFileTest(FILE_NAME_1);
118         Program pRef = this.createFromFileRef(FILE_NAME_1);
119 
120         /*
121          * The call
122          */
123         String result = pTest.name();
124 
125         /*
126          * Evaluation
127          */
128         assertEquals(pRef, pTest);
129         assertEquals("Test", result);
130     }
131 
132     /**
133      * Test setName.
134      */
135     @Test
136     public final void testSetName() {
137         /*
138          * Setup
139          */
140         Program pTest = this.createFromFileTest(FILE_NAME_1);
141         Program pRef = this.createFromFileRef(FILE_NAME_1);
142         String newName = "Replacement";
```

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

```
182     */
183    @Test
184    public final void testSwapContext() {
185        /*
186         * Setup
187         */
188        Program pTest = this.createFromFileTest(FILE_NAME_1);
189        Program pRef = this.createFromFileRef(FILE_NAME_1);
190        Map<String, Statement> contextRef = pRef.newContext();
191        Map<String, Statement> contextTest =
   pTest.newContext();
192        String oneName = "one";
193        pRef.swapContext(contextRef);
194        Pair<String, Statement> oneRef =
   contextRef.remove(oneName);
195        /* contextRef now has just "two" */
196        pRef.swapContext(contextRef);
197        /* pRef's context now has just "two" */
198        contextRef.add(oneRef.key(), oneRef.value());
199        /* contextRef now has just "one" */
200
201        /* Make the reference call, replacing, in pRef, "one"
   with "two": */
202        pRef.swapContext(contextRef);
203
204        pTest.swapContext(contextTest);
205        Pair<String, Statement> oneTest =
   contextTest.remove(oneName);
206        /* contextTest now has just "two" */
207        pTest.swapContext(contextTest);
208        /* pTest's context now has just "two" */
209        contextTest.add(oneTest.key(), oneTest.value());
210        /* contextTest now has just "one" */
211
212        /*
213         * The call
214         */
215        pTest.swapContext(contextTest);
216
```

```
217         /*
218          * Evaluation
219          */
220         assertEquals(pRef, pTest);
221         assertEquals(contextRef, contextTest);
222     }
223 
224     /**
225      * Test newBody.
226      */
227     @Test
228     public final void testNewBody() {
229         /*
230          * Setup
231          */
232         Program pTest = this.createFromFileTest(FILE_NAME_1);
233         Program pRef = this.createFromFileRef(FILE_NAME_1);
234         Statement bRef = pRef.newBody();
235 
236         /*
237          * The call
238          */
239         Statement bTest = pTest.newBody();
240 
241         /*
242          * Evaluation
243          */
244         assertEquals(pRef, pTest);
245         assertEquals(bRef, bTest);
246     }
247 
248     /**
249      * Test swapBody.
250      */
251     @Test
252     public final void testSwapBody() {
253         /*
254          * Setup
255          */
```

```
256         Program pTest = this.createFromFileTest(FILE_NAME_1);
257         Program pRef = this.createFromFileRef(FILE_NAME_1);
258         Statement bodyRef = pRef.newBody();
259         Statement bodyTest = pTest.newBody();
260         pRef.swapBody(bodyRef);
261         Statement firstRef = bodyRef.removeFromBlock(0);
262         /* bodyRef now lacks the first statement */
263         pRef.swapBody(bodyRef);
264         /* pRef's body now lacks the first statement */
265         bodyRef.addToBlock(0, firstRef);
266         /* bodyRef now has just the first statement */
267
268         /* Make the reference call, replacing, in pRef,
    remaining with first: */
269         pRef.swapBody(bodyRef);
270
271         pTest.swapBody(bodyTest);
272         Statement firstTest = bodyTest.removeFromBlock(0);
273         /* bodyTest now lacks the first statement */
274         pTest.swapBody(bodyTest);
275         /* pTest's body now lacks the first statement */
276         bodyTest.addToBlock(0, firstTest);
277         /* bodyTest now has just the first statement */
278
279         /*
280          * The call
281          */
282         pTest.swapBody(bodyTest);
283
284         /*
285          * Evaluation
286          */
287         assertEquals(pRef, pTest);
288         assertEquals(bodyRef, bodyTest);
289     }
290
291     // TODO - provide additional test cases to thoroughly test
    ProgramKernel
292
```

```
293     /**
294      * Test name.
295      */
296     @Test
297     public final void testNameFileName2() {
298         /*
299          * Setup
300          */
301         Program pTest = this.createFromFileTest(FILE_NAME_2);
302         Program pRef = this.createFromFileRef(FILE_NAME_2);
303 
304         /*
305          * The call
306          */
307         String result = pTest.name();
308 
309         /*
310          * Evaluation
311          */
312         assertEquals(pRef, pTest);
313         assertEquals("BugTerminator", result);
314     }
315 
316     /**
317      * Test setName.
318      */
319     @Test
320     public final void testSetNameFileName2() {
321         /*
322          * Setup
323          */
324         Program pTest = this.createFromFileTest(FILE_NAME_2);
325         Program pRef = this.createFromFileRef(FILE_NAME_2);
326         String newName = "Replacement";
327         pRef.setName(newName);
328 
329         /*
330          * The call
331          */
```

```
332         pTest.setName(newName);
333 
334         /*
335          * Evaluation
336          */
337         assertEquals(pRef, pTest);
338     }
339 
340     /**
341      * Test newContext.
342      */
343     @Test
344     public final void testNewContextFileName2() {
345         /*
346          * Setup
347          */
348         Program pTest = this.createFromFileTest(FILE_NAME_2);
349         Program pRef = this.createFromFileRef(FILE_NAME_2);
350         Map<String, Statement> cRef = pRef.newContext();
351 
352         /*
353          * The call
354          */
355         Map<String, Statement> cTest = pTest.newContext();
356 
357         /*
358          * Evaluation
359          */
360         assertEquals(pRef, pTest);
361         assertEquals(cRef, cTest);
362     }
363 
364     /**
365      * Test swapContext.
366      */
367     @Test
368     public final void testSwapContextFileName2() {
369         /*
370          * Setup
```

```
371         */
372        Program pTest = this.createFromFileTest(FILE_NAME_2);
373        Program pRef = this.createFromFileRef(FILE_NAME_2);
374        Map<String, Statement> contextRef = pRef.newContext();
375        Map<String, Statement> contextTest =
   pTest.newContext();
376        String oneName = "kill";
377        pRef.swapContext(contextRef);
378        Pair<String, Statement> oneRef =
   contextRef.remove(oneName);
379        pRef.swapContext(contextRef);
380        contextRef.add(oneRef.key(), oneRef.value());
381        pRef.swapContext(contextRef);
382
383        pTest.swapContext(contextTest);
384        Pair<String, Statement> oneTest =
   contextTest.remove(oneName);
385        pTest.swapContext(contextTest);
386        contextTest.add(oneTest.key(), oneTest.value());
387
388        /*
389         * The call
390         */
391        pTest.swapContext(contextTest);
392
393        /*
394         * Evaluation
395         */
396        assertEquals(pRef, pTest);
397        assertEquals(contextRef, contextTest);
398    }
399
400    /**
401     * Test newBody.
402     */
403    @Test
404    public final void testNewBodyFileName2() {
405        /*
406         * Setup
```

```
407          */
408         Program pTest = this.createFromFileTest(FILE_NAME_2);
409         Program pRef = this.createFromFileRef(FILE_NAME_2);
410         Statement bRef = pRef.newBody();
411 
412         /*
413          * The call
414          */
415         Statement bTest = pTest.newBody();
416 
417         /*
418          * Evaluation
419          */
420         assertEquals(pRef, pTest);
421         assertEquals(bRef, bTest);
422     }
423 
424     /**
425      * Test swapBody.
426      */
427     @Test
428     public final void testSwapBodyFileName2() {
429         /*
430          * Setup
431          */
432         Program pTest = this.createFromFileTest(FILE_NAME_2);
433         Program pRef = this.createFromFileRef(FILE_NAME_2);
434         Statement bodyRef = pRef.newBody();
435         Statement bodyTest = pTest.newBody();
436         pRef.swapBody(bodyRef);
437         Statement firstRef = bodyRef.removeFromBlock(0);
438         /* bodyRef now lacks the first statement */
439         pRef.swapBody(bodyRef);
440         /* pRef's body now lacks the first statement */
441         bodyRef.addToBlock(0, firstRef);
442         /* bodyRef now has just the first statement */
443 
444         /* Make the reference call, replacing, in pRef,
    remaining with first: */
```

```
445         pRef.swapBody(bodyRef);
446
447         pTest.swapBody(bodyTest);
448         Statement firstTest = bodyTest.removeFromBlock(0);
449         /* bodyTest now lacks the first statement */
450         pTest.swapBody(bodyTest);
451         /* pTest's body now lacks the first statement */
452         bodyTest.addToBlock(0, firstTest);
453         /* bodyTest now has just the first statement */
454
455         /*
456          * The call
457          */
458         pTest.swapBody(bodyTest);
459
460         /*
461          * Evaluation
462          */
463         assertEquals(pRef, pTest);
464         assertEquals(bodyRef, bodyTest);
465     }
466
467     /**
468      * Test name.
469      */
470     @Test
471     public final void testNameFileName3() {
472         /*
473          * Setup
474          */
475         Program pTest = this.createFromFileTest(FILE_NAME_3);
476         Program pRef = this.createFromFileRef(FILE_NAME_3);
477
478         /*
479          * The call
480          */
481         String result = pTest.name();
482
483         /*
```

```
484         * Evaluation
485         */
486        assertEquals(pRef, pTest);
487        assertEquals("WeAreBorg", result);
488    }
489
490    /**
491     * Test setName.
492     */
493    @Test
494    public final void testSetNameFileName3() {
495        /*
496         * Setup
497         */
498        Program pTest = this.createFromFileTest(FILE_NAME_3);
499        Program pRef = this.createFromFileRef(FILE_NAME_3);
500        String newName = "Replacement";
501        pRef.setName(newName);
502
503        /*
504         * The call
505         */
506        pTest.setName(newName);
507
508        /*
509         * Evaluation
510         */
511        assertEquals(pRef, pTest);
512    }
513
514    /**
515     * Test newContext.
516     */
517    @Test
518    public final void testNewContextFileName3() {
519        /*
520         * Setup
521         */
522        Program pTest = this.createFromFileTest(FILE_NAME_3);
```

```
523         Program pRef = this.createFromFileRef(FILE_NAME_3);
524         Map<String, Statement> cRef = pRef.newContext();
525 
526         /*
527          * The call
528          */
529         Map<String, Statement> cTest = pTest.newContext();
530 
531         /*
532          * Evaluation
533          */
534         assertEquals(pRef, pTest);
535         assertEquals(cRef, cTest);
536     }
537 
538     /**
539      * Test swapContext.
540      */
541     @Test
542     public final void testSwapContextFileName3() {
543         /*
544          * Setup
545          */
546         Program pTest = this.createFromFileTest(FILE_NAME_3);
547         Program pRef = this.createFromFileRef(FILE_NAME_3);
548         Map<String, Statement> contextRef = pRef.newContext();
549         Map<String, Statement> contextTest =
    pTest.newContext();
550         String oneName = "FindSpecies8472";
551         pRef.swapContext(contextRef);
552         Pair<String, Statement> oneRef =
    contextRef.remove(oneName);
553         pRef.swapContext(contextRef);
554         contextRef.add(oneRef.key(), oneRef.value());
555 
556         pRef.swapContext(contextRef);
557 
558         pTest.swapContext(contextTest);
559         Pair<String, Statement> oneTest =
```

```
   contextTest.remove(oneName);
560         pTest.swapContext(contextTest);
561         contextTest.add(oneTest.key(), oneTest.value());
562 
563         /*
564          * The call
565          */
566         pTest.swapContext(contextTest);
567 
568         /*
569          * Evaluation
570          */
571         assertEquals(pRef, pTest);
572         assertEquals(contextRef, contextTest);
573     }
574 
575     /**
576      * Test newBody.
577      */
578     @Test
579     public final void testNewBodyFileName3() {
580         /*
581          * Setup
582          */
583         Program pTest = this.createFromFileTest(FILE_NAME_3);
584         Program pRef = this.createFromFileRef(FILE_NAME_3);
585         Statement bRef = pRef.newBody();
586 
587         /*
588          * The call
589          */
590         Statement bTest = pTest.newBody();
591 
592         /*
593          * Evaluation
594          */
595         assertEquals(pRef, pTest);
596         assertEquals(bRef, bTest);
597     }
```

```
598
599     /**
600      * Test swapBody.
601      */
602     @Test
603     public final void testSwapBodyFileName3() {
604         /*
605          * Setup
606          */
607         Program pTest = this.createFromFileTest(FILE_NAME_3);
608         Program pRef = this.createFromFileRef(FILE_NAME_3);
609         Statement bodyRef = pRef.newBody();
610         Statement bodyTest = pTest.newBody();
611         pRef.swapBody(bodyRef);
612         Statement firstRef = bodyRef.removeFromBlock(0);
613         /* bodyRef now lacks the first statement */
614         pRef.swapBody(bodyRef);
615         /* pRef's body now lacks the first statement */
616         bodyRef.addToBlock(0, firstRef);
617         /* bodyRef now has just the first statement */
618
619         /* Make the reference call, replacing, in pRef,
    remaining with first: */
620         pRef.swapBody(bodyRef);
621
622         pTest.swapBody(bodyTest);
623         Statement firstTest = bodyTest.removeFromBlock(0);
624         /* bodyTest now lacks the first statement */
625         pTest.swapBody(bodyTest);
626         /* pTest's body now lacks the first statement */
627         bodyTest.addToBlock(0, firstTest);
628         /* bodyTest now has just the first statement */
629
630         /*
631          * The call
632          */
633         pTest.swapBody(bodyTest);
634
635         /*
```

```
636          * Evaluation
637          */
638         assertEquals(pRef, pTest);
639         assertEquals(bodyRef, bodyTest);
640     }
641
642 }
643
```

```
 1 import components.statement.Statement;
 2 import components.statement.Statement1;
 3
 4 /**
 5  * Customized JUnit test fixture for {@code Statement2}.
 6  */
 7 public class Statement2Test extends StatementTest {
 8
 9     @Override
10     protected final Statement constructorTest() {
11         return new Statement2();
12     }
13
14     @Override
15     protected final Statement constructorRef() {
16         return new Statement1();
17     }
18
19 }
20
```

```
 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.statement.StatementKernel.Condition;
10 import components.statement.StatementKernel.Kind;
11 import components.utilities.Tokenizer;
12
13 /**
14  * JUnit test fixture for {@code Statement}'s constructor and
   kernel methods.
15  *
16  * @author Ansh Pachauri
17  * @author Daniil Gofman
18  *
19  */
20 public abstract class StatementTest {
21
22     /**
23      * The name of a file containing a sequence of BL
   statements.
24      */
25     private static final String FILE_NAME_1 = "data/statement-
   sample.bl";
26
27     /**
28      * The name of a file containing a sequence of BL
   statements.
29      */
30     private static final String FILE_NAME_2 = "data/statement-
   sample2.bl";
31
32     /**
33      * The name of a file containing a sequence of BL
   statements.
```

```
 34     */
 35    private static final String FILE_NAME_3 = "data/statement-
    sample3.bl";
 36
 37    /**
 38     * Invokes the {@code Statement} constructor for the
    implementation under
 39     * test and returns the result.
 40     *
 41     * @return the new statement
 42     * @ensures constructor = compose((BLOCK, ?, ?), <>)
 43     */
 44    protected abstract Statement constructorTest();
 45
 46    /**
 47     * Invokes the {@code Statement} constructor for the
    reference
 48     * implementation and returns the result.
 49     *
 50     * @return the new statement
 51     * @ensures constructor = compose((BLOCK, ?, ?), <>)
 52     */
 53    protected abstract Statement constructorRef();
 54
 55    /**
 56     *
 57     * Creates and returns a block {@code Statement}, of the
    type of the
 58     * implementation under test, from the file with the given
    name.
 59     *
 60     * @param filename
 61     *            the name of the file to be parsed for the
    sequence of
 62     *            statements to go in the block statement
 63     * @return the constructed block statement
 64     * @ensures <pre>
 65     * createFromFile = [the block statement containing the
    statements
```

```
 66      * parsed from the file]
 67      * </pre>
 68      */
 69     private Statement createFromFileTest(String filename) {
 70         Statement s = this.constructorTest();
 71         SimpleReader file = new SimpleReader1L(filename);
 72         Queue<String> tokens = Tokenizer.tokens(file);
 73         s.parseBlock(tokens);
 74         file.close();
 75         return s;
 76     }
 77
 78     /**
 79      *
 80      * Creates and returns a block {@code Statement}, of the
    reference
 81      * implementation type, from the file with the given name.
 82      *
 83      * @param filename
 84      *            the name of the file to be parsed for the
    sequence of
 85      *            statements to go in the block statement
 86      * @return the constructed block statement
 87      * @ensures <pre>
 88      * createFromFile = [the block statement containing the
    statements
 89      * parsed from the file]
 90      * </pre>
 91      */
 92     private Statement createFromFileRef(String filename) {
 93         Statement s = this.constructorRef();
 94         SimpleReader file = new SimpleReader1L(filename);
 95         Queue<String> tokens = Tokenizer.tokens(file);
 96         s.parseBlock(tokens);
 97         file.close();
 98         return s;
 99     }
100
101     /**
```

```
102     * Test constructor.
103     */
104    @Test
105    public final void testConstructor() {
106        /*
107         * Setup
108         */
109        Statement sRef = this.constructorRef();
110
111        /*
112         * The call
113         */
114        Statement sTest = this.constructorTest();
115
116        /*
117         * Evaluation
118         */
119        assertEquals(sRef, sTest);
120    }
121
122    /**
123     * Test kind of a WHILE statement.
124     */
125    @Test
126    public final void testKindWhile1() {
127        /*
128         * Setup
129         */
130        final int whilePos = 3;
131        Statement sourceTest =
    this.createFromFileTest(FILE_NAME_1);
132        Statement sourceRef =
    this.createFromFileRef(FILE_NAME_1);
133        Statement sTest =
    sourceTest.removeFromBlock(whilePos);
134        Statement sRef = sourceRef.removeFromBlock(whilePos);
135        Kind kRef = sRef.kind();
136
137        /*
```

```
138         * The call
139         */
140        Kind kTest = sTest.kind();
141
142        /*
143         * Evaluation
144         */
145        assertEquals(kRef, kTest);
146        assertEquals(sRef, sTest);
147    }
148
149    /**
150     * Test kind of a WHILE statement.
151     */
152    @Test
153    public final void testKindWhile2() {
154        /*
155         * Setup
156         */
157        final int whilePos = 3;
158        Statement sourceTest =
    this.createFromFileTest(FILE_NAME_2);
159        Statement sourceRef =
    this.createFromFileRef(FILE_NAME_2);
160        Statement sTest =
    sourceTest.removeFromBlock(whilePos);
161        Statement sRef = sourceRef.removeFromBlock(whilePos);
162        Kind kRef = sRef.kind();
163
164        /*
165         * The call
166         */
167        Kind kTest = sTest.kind();
168
169        /*
170         * Evaluation
171         */
172        assertEquals(kRef, kTest);
173        assertEquals(sRef, sTest);
```

```
174     }
175
176     /**
177      * Test kind of a WHILE statement.
178      */
179     @Test
180     public final void testKindWhile3() {
181         /*
182          * Setup
183          */
184         final int whilePos = 3;
185         Statement sourceTest =
    this.createFromFileTest(FILE_NAME_3);
186         Statement sourceRef =
    this.createFromFileRef(FILE_NAME_3);
187         Statement sTest =
    sourceTest.removeFromBlock(whilePos);
188         Statement sRef = sourceRef.removeFromBlock(whilePos);
189         Kind kRef = sRef.kind();
190
191         /*
192          * The call
193          */
194         Kind kTest = sTest.kind();
195
196         /*
197          * Evaluation
198          */
199         assertEquals(kRef, kTest);
200         assertEquals(sRef, sTest);
201     }
202
203     /**
204      * Test addToBlock at an interior position.
205      */
206     @Test
207     public final void testAddToBlockInterior1() {
208         /*
209          * Setup
```

```
210         */
211        Statement sTest =
    this.createFromFileTest(FILE_NAME_1);
212        Statement sRef = this.createFromFileRef(FILE_NAME_1);
213        Statement emptyBlock = sRef.newInstance();
214        Statement nestedTest = sTest.removeFromBlock(1);
215        Statement nestedRef = sRef.removeFromBlock(1);
216        sRef.addToBlock(2, nestedRef);
217
218        /*
219         * The call
220         */
221        sTest.addToBlock(2, nestedTest);
222
223        /*
224         * Evaluation
225         */
226        assertEquals(emptyBlock, nestedTest);
227        assertEquals(sRef, sTest);
228    }
229
230    /**
231     * Test addToBlock at an interior position.
232     */
233    @Test
234    public final void testAddToBlockInterior2() {
235        /*
236         * Setup
237         */
238        Statement sTest =
    this.createFromFileTest(FILE_NAME_2);
239        Statement sRef = this.createFromFileRef(FILE_NAME_2);
240        Statement emptyBlock = sRef.newInstance();
241        Statement nestedTest = sTest.removeFromBlock(1);
242        Statement nestedRef = sRef.removeFromBlock(1);
243        sRef.addToBlock(2, nestedRef);
244
245        /*
246         * The call
```

```
247          */
248         sTest.addToBlock(2, nestedTest);
249
250         /*
251          * Evaluation
252          */
253         assertEquals(emptyBlock, nestedTest);
254         assertEquals(sRef, sTest);
255     }
256
257     /**
258      * Test addToBlock at an interior position.
259      */
260     @Test
261     public final void testAddToBlockInterior3() {
262         /*
263          * Setup
264          */
265         Statement sTest =
    this.createFromFileTest(FILE_NAME_3);
266         Statement sRef = this.createFromFileRef(FILE_NAME_3);
267         Statement emptyBlock = sRef.newInstance();
268         Statement nestedTest = sTest.removeFromBlock(1);
269         Statement nestedRef = sRef.removeFromBlock(1);
270         sRef.addToBlock(2, nestedRef);
271
272         /*
273          * The call
274          */
275         sTest.addToBlock(2, nestedTest);
276
277         /*
278          * Evaluation
279          */
280         assertEquals(emptyBlock, nestedTest);
281         assertEquals(sRef, sTest);
282     }
283
284     /**
```

```
285     * Test removeFromBlock at the front leaving a non-empty
    block behind.
286     */
287    @Test
288    public final void
    testRemoveFromBlockFrontLeavingNonEmpty1() {
289        /*
290         * Setup
291         */
292        Statement sTest =
    this.createFromFileTest(FILE_NAME_1);
293        Statement sRef = this.createFromFileRef(FILE_NAME_1);
294        Statement nestedRef = sRef.removeFromBlock(0);
295
296        /*
297         * The call
298         */
299        Statement nestedTest = sTest.removeFromBlock(0);
300
301        /*
302         * Evaluation
303         */
304        assertEquals(sRef, sTest);
305        assertEquals(nestedRef, nestedTest);
306    }
307
308    /**
309     * Test removeFromBlock at the front leaving a non-empty
    block behind.
310     */
311    @Test
312    public final void
    testRemoveFromBlockFrontLeavingNonEmpty2() {
313        /*
314         * Setup
315         */
316        Statement sTest =
    this.createFromFileTest(FILE_NAME_2);
317        Statement sRef = this.createFromFileRef(FILE_NAME_2);
```

```
318         Statement nestedRef = sRef.removeFromBlock(0);
319 
320         /*
321          * The call
322          */
323         Statement nestedTest = sTest.removeFromBlock(0);
324 
325         /*
326          * Evaluation
327          */
328         assertEquals(sRef, sTest);
329         assertEquals(nestedRef, nestedTest);
330     }
331 
332     /**
333      * Test removeFromBlock at the front leaving a non-empty
    block behind.
334      */
335     @Test
336     public final void
    testRemoveFromBlockFrontLeavingNonEmpty3() {
337         /*
338          * Setup
339          */
340         Statement sTest =
    this.createFromFileTest(FILE_NAME_3);
341         Statement sRef = this.createFromFileRef(FILE_NAME_3);
342         Statement nestedRef = sRef.removeFromBlock(0);
343 
344         /*
345          * The call
346          */
347         Statement nestedTest = sTest.removeFromBlock(0);
348 
349         /*
350          * Evaluation
351          */
352         assertEquals(sRef, sTest);
353         assertEquals(nestedRef, nestedTest);
```

```
354     }
355
356     /**
357      * Test lengthOfBlock, greater than zero.
358      */
359     @Test
360     public final void testLengthOfBlockNonEmpty1() {
361         /*
362          * Setup
363          */
364         Statement sTest =
    this.createFromFileTest(FILE_NAME_1);
365         Statement sRef = this.createFromFileRef(FILE_NAME_1);
366         int lengthRef = sRef.lengthOfBlock();
367
368         /*
369          * The call
370          */
371         int lengthTest = sTest.lengthOfBlock();
372
373         /*
374          * Evaluation
375          */
376         assertEquals(lengthRef, lengthTest);
377         assertEquals(sRef, sTest);
378     }
379
380     /**
381      * Test lengthOfBlock, greater than zero.
382      */
383     @Test
384     public final void testLengthOfBlockNonEmpty2() {
385         /*
386          * Setup
387          */
388         Statement sTest =
    this.createFromFileTest(FILE_NAME_2);
389         Statement sRef = this.createFromFileRef(FILE_NAME_2);
390         int lengthRef = sRef.lengthOfBlock();
```

```
391
392         /*
393          * The call
394          */
395         int lengthTest = sTest.lengthOfBlock();
396
397         /*
398          * Evaluation
399          */
400         assertEquals(lengthRef, lengthTest);
401         assertEquals(sRef, sTest);
402     }
403
404     /**
405      * Test lengthOfBlock, greater than zero.
406      */
407     @Test
408     public final void testLengthOfBlockNonEmpty3() {
409         /*
410          * Setup
411          */
412         Statement sTest =
    this.createFromFileTest(FILE_NAME_3);
413         Statement sRef = this.createFromFileRef(FILE_NAME_3);
414         int lengthRef = sRef.lengthOfBlock();
415
416         /*
417          * The call
418          */
419         int lengthTest = sTest.lengthOfBlock();
420
421         /*
422          * Evaluation
423          */
424         assertEquals(lengthRef, lengthTest);
425         assertEquals(sRef, sTest);
426     }
427
428     /**
```

```
429     * Test assembleIf.
430     */
431    @Test
432    public final void testAssembleIf1() {
433        /*
434         * Setup
435         */
436        Statement blockTest =
   this.createFromFileTest(FILE_NAME_1);
437        Statement blockRef =
   this.createFromFileRef(FILE_NAME_1);
438        Statement emptyBlock = blockRef.newInstance();
439        Statement sourceTest = blockTest.removeFromBlock(1);
440        Statement sRef = blockRef.removeFromBlock(1);
441        Statement nestedTest = sourceTest.newInstance();
442        Condition c = sourceTest.disassembleIf(nestedTest);
443        Statement sTest = sourceTest.newInstance();
444
445        /*
446         * The call
447         */
448        sTest.assembleIf(c, nestedTest);
449
450        /*
451         * Evaluation
452         */
453        assertEquals(emptyBlock, nestedTest);
454        assertEquals(sRef, sTest);
455    }
456
457    /**
458     * Test assembleIf.
459     */
460    @Test
461    public final void testAssembleIf2() {
462        /*
463         * Setup
464         */
465        Statement blockTest =
```

```
    this.createFromFileTest(FILE_NAME_2);
466         Statement blockRef =
    this.createFromFileRef(FILE_NAME_2);
467         Statement emptyBlock = blockRef.newInstance();
468         Statement sourceTest = blockTest.removeFromBlock(8);
469         Statement sRef = blockRef.removeFromBlock(8);
470         Statement nestedTest = sourceTest.newInstance();
471         Condition c = sourceTest.disassembleIf(nestedTest);
472         Statement sTest = sourceTest.newInstance();
473
474         /*
475          * The call
476          */
477         sTest.assembleIf(c, nestedTest);
478
479         /*
480          * Evaluation
481          */
482         assertEquals(emptyBlock, nestedTest);
483         assertEquals(sRef, sTest);
484     }
485
486     /**
487      * Test assembleIf.
488      */
489     @Test
490     public final void testAssembleIf3() {
491         /*
492          * Setup
493          */
494         Statement blockTest =
    this.createFromFileTest(FILE_NAME_3);
495         Statement blockRef =
    this.createFromFileRef(FILE_NAME_3);
496         Statement emptyBlock = blockRef.newInstance();
497         Statement sourceTest = blockTest.removeFromBlock(4);
498         Statement sRef = blockRef.removeFromBlock(4);
499         Statement nestedTest = sourceTest.newInstance();
500         Condition c = sourceTest.disassembleIf(nestedTest);
```

```
501         Statement sTest = sourceTest.newInstance();
502 
503         /*
504          * The call
505          */
506         sTest.assembleIf(c, nestedTest);
507 
508         /*
509          * Evaluation
510          */
511         assertEquals(emptyBlock, nestedTest);
512         assertEquals(sRef, sTest);
513     }
514 
515     /**
516      * Test disassembleIf.
517      */
518     @Test
519     public final void testDisassembleIf1() {
520         /*
521          * Setup
522          */
523         Statement blockTest =
    this.createFromFileTest(FILE_NAME_1);
524         Statement blockRef =
    this.createFromFileRef(FILE_NAME_1);
525         Statement sTest = blockTest.removeFromBlock(1);
526         Statement sRef = blockRef.removeFromBlock(1);
527         Statement nestedTest = sTest.newInstance();
528         Statement nestedRef = sRef.newInstance();
529         Condition cRef = sRef.disassembleIf(nestedRef);
530 
531         /*
532          * The call
533          */
534         Condition cTest = sTest.disassembleIf(nestedTest);
535 
536         /*
537          * Evaluation
```

```
538          */
539         assertEquals(nestedRef, nestedTest);
540         assertEquals(sRef, sTest);
541         assertEquals(cRef, cTest);
542     }
543 
544     /**
545      * Test disassembleIf.
546      */
547     @Test
548     public final void testDisassembleIf2() {
549         /*
550          * Setup
551          */
552         Statement blockTest =
    this.createFromFileTest(FILE_NAME_2);
553         Statement blockRef =
    this.createFromFileRef(FILE_NAME_2);
554         Statement sTest = blockTest.removeFromBlock(0);
555         Statement sRef = blockRef.removeFromBlock(0);
556         Statement nestedTest = sTest.newInstance();
557         Statement nestedRef = sRef.newInstance();
558         Condition cRef = sRef.disassembleIf(nestedRef);
559 
560         /*
561          * The call
562          */
563         Condition cTest = sTest.disassembleIf(nestedTest);
564 
565         /*
566          * Evaluation
567          */
568         assertEquals(nestedRef, nestedTest);
569         assertEquals(sRef, sTest);
570         assertEquals(cRef, cTest);
571     }
572 
573     /**
574      * Test disassembleIf.
```

```
575     */
576    @Test
577    public final void testDisassembleIf3() {
578        /*
579         * Setup
580         */
581        Statement blockTest =
    this.createFromFileTest(FILE_NAME_3);
582        Statement blockRef =
    this.createFromFileRef(FILE_NAME_3);
583        Statement sTest = blockTest.removeFromBlock(4);
584        Statement sRef = blockRef.removeFromBlock(4);
585        Statement nestedTest = sTest.newInstance();
586        Statement nestedRef = sRef.newInstance();
587        Condition cRef = sRef.disassembleIf(nestedRef);
588
589        /*
590         * The call
591         */
592        Condition cTest = sTest.disassembleIf(nestedTest);
593
594        /*
595         * Evaluation
596         */
597        assertEquals(nestedRef, nestedTest);
598        assertEquals(sRef, sTest);
599        assertEquals(cRef, cTest);
600    }
601
602    /**
603     * Test assembleIfElse.
604     */
605    @Test
606    public final void testAssembleIfElse1() {
607        /*
608         * Setup
609         */
610        final int ifElsePos = 2;
611        Statement blockTest =
```

```
    this.createFromFileTest(FILE_NAME_1);
612         Statement blockRef =
    this.createFromFileRef(FILE_NAME_1);
613         Statement emptyBlock = blockRef.newInstance();
614         Statement sourceTest =
    blockTest.removeFromBlock(ifElsePos);
615         Statement sRef = blockRef.removeFromBlock(ifElsePos);
616         Statement thenBlockTest = sourceTest.newInstance();
617         Statement elseBlockTest = sourceTest.newInstance();
618         Condition cTest =
    sourceTest.disassembleIfElse(thenBlockTest,
619                 elseBlockTest);
620         Statement sTest = blockTest.newInstance();
621
622         /*
623          * The call
624          */
625         sTest.assembleIfElse(cTest, thenBlockTest,
    elseBlockTest);
626
627         /*
628          * Evaluation
629          */
630         assertEquals(emptyBlock, thenBlockTest);
631         assertEquals(emptyBlock, elseBlockTest);
632         assertEquals(sRef, sTest);
633     }
634
635     /**
636      * Test assembleIfElse.
637      */
638     @Test
639     public final void testAssembleIfElse2() {
640         /*
641          * Setup
642          */
643         final int ifElsePos = 6;
644         Statement blockTest =
    this.createFromFileTest(FILE_NAME_2);
```

```
645         Statement blockRef =
    this.createFromFileRef(FILE_NAME_2);
646         Statement emptyBlock = blockRef.newInstance();
647         Statement sourceTest =
    blockTest.removeFromBlock(ifElsePos);
648         Statement sRef = blockRef.removeFromBlock(ifElsePos);
649         Statement thenBlockTest = sourceTest.newInstance();
650         Statement elseBlockTest = sourceTest.newInstance();
651         Condition cTest =
    sourceTest.disassembleIfElse(thenBlockTest,
652                 elseBlockTest);
653         Statement sTest = blockTest.newInstance();
654
655         /*
656          * The call
657          */
658         sTest.assembleIfElse(cTest, thenBlockTest,
    elseBlockTest);
659
660         /*
661          * Evaluation
662          */
663         assertEquals(emptyBlock, thenBlockTest);
664         assertEquals(emptyBlock, elseBlockTest);
665         assertEquals(sRef, sTest);
666     }
667
668     /**
669      * Test assembleIfElse.
670      */
671     @Test
672     public final void testAssembleIfElse3() {
673         /*
674          * Setup
675          */
676         final int ifElsePos = 0;
677         Statement blockTest =
    this.createFromFileTest(FILE_NAME_3);
678         Statement blockRef =
```

```
    this.createFromFileRef(FILE_NAME_3);
679         Statement emptyBlock = blockRef.newInstance();
680         Statement sourceTest =
    blockTest.removeFromBlock(ifElsePos);
681         Statement sRef = blockRef.removeFromBlock(ifElsePos);
682         Statement thenBlockTest = sourceTest.newInstance();
683         Statement elseBlockTest = sourceTest.newInstance();
684         Condition cTest =
    sourceTest.disassembleIfElse(thenBlockTest,
685                 elseBlockTest);
686         Statement sTest = blockTest.newInstance();
687
688         /*
689          * The call
690          */
691         sTest.assembleIfElse(cTest, thenBlockTest,
    elseBlockTest);
692
693         /*
694          * Evaluation
695          */
696         assertEquals(emptyBlock, thenBlockTest);
697         assertEquals(emptyBlock, elseBlockTest);
698         assertEquals(sRef, sTest);
699     }
700
701     /**
702      * Test disassembleIfElse.
703      */
704     @Test
705     public final void testDisassembleIfElse1() {
706         /*
707          * Setup
708          */
709         final int ifElsePos = 2;
710         Statement blockTest =
    this.createFromFileTest(FILE_NAME_1);
711         Statement blockRef =
    this.createFromFileRef(FILE_NAME_1);
```

```
712         Statement sTest =
    blockTest.removeFromBlock(ifElsePos);
713         Statement sRef = blockRef.removeFromBlock(ifElsePos);
714         Statement thenBlockTest = sTest.newInstance();
715         Statement elseBlockTest = sTest.newInstance();
716         Statement thenBlockRef = sRef.newInstance();
717         Statement elseBlockRef = sRef.newInstance();
718         Condition cRef = sRef.disassembleIfElse(thenBlockRef,
    elseBlockRef);
719
720         /*
721          * The call
722          */
723         Condition cTest =
    sTest.disassembleIfElse(thenBlockTest, elseBlockTest);
724
725         /*
726          * Evaluation
727          */
728         assertEquals(cRef, cTest);
729         assertEquals(thenBlockRef, thenBlockTest);
730         assertEquals(elseBlockRef, elseBlockTest);
731         assertEquals(sRef, sTest);
732     }
733
734     /**
735      * Test disassembleIfElse.
736      */
737     @Test
738     public final void testDisassembleIfElse2() {
739         /*
740          * Setup
741          */
742         final int ifElsePos = 6;
743         Statement blockTest =
    this.createFromFileTest(FILE_NAME_2);
744         Statement blockRef =
    this.createFromFileRef(FILE_NAME_2);
745         Statement sTest =
```

```
    blockTest.removeFromBlock(ifElsePos);
746         Statement sRef = blockRef.removeFromBlock(ifElsePos);
747         Statement thenBlockTest = sTest.newInstance();
748         Statement elseBlockTest = sTest.newInstance();
749         Statement thenBlockRef = sRef.newInstance();
750         Statement elseBlockRef = sRef.newInstance();
751         Condition cRef = sRef.disassembleIfElse(thenBlockRef,
    elseBlockRef);
752
753         /*
754          * The call
755          */
756         Condition cTest =
    sTest.disassembleIfElse(thenBlockTest, elseBlockTest);
757
758         /*
759          * Evaluation
760          */
761         assertEquals(cRef, cTest);
762         assertEquals(thenBlockRef, thenBlockTest);
763         assertEquals(elseBlockRef, elseBlockTest);
764         assertEquals(sRef, sTest);
765     }
766
767     /**
768      * Test disassembleIfElse.
769      */
770     @Test
771     public final void testDisassembleIfElse3() {
772         /*
773          * Setup
774          */
775         final int ifElsePos = 0;
776         Statement blockTest =
    this.createFromFileTest(FILE_NAME_3);
777         Statement blockRef =
    this.createFromFileRef(FILE_NAME_3);
778         Statement sTest =
    blockTest.removeFromBlock(ifElsePos);
```

```
779         Statement sRef = blockRef.removeFromBlock(ifElsePos);
780         Statement thenBlockTest = sTest.newInstance();
781         Statement elseBlockTest = sTest.newInstance();
782         Statement thenBlockRef = sRef.newInstance();
783         Statement elseBlockRef = sRef.newInstance();
784         Condition cRef = sRef.disassembleIfElse(thenBlockRef,
    elseBlockRef);
785
786         /*
787          * The call
788          */
789         Condition cTest =
    sTest.disassembleIfElse(thenBlockTest, elseBlockTest);
790
791         /*
792          * Evaluation
793          */
794         assertEquals(cRef, cTest);
795         assertEquals(thenBlockRef, thenBlockTest);
796         assertEquals(elseBlockRef, elseBlockTest);
797         assertEquals(sRef, sTest);
798     }
799
800     /**
801      * Test assembleWhile.
802      */
803     @Test
804     public final void testAssembleWhile1() {
805         /*
806          * Setup
807          */
808         Statement blockTest =
    this.createFromFileTest(FILE_NAME_1);
809         Statement blockRef =
    this.createFromFileRef(FILE_NAME_1);
810         Statement emptyBlock = blockRef.newInstance();
811         Statement sourceTest = blockTest.removeFromBlock(1);
812         Statement sourceRef = blockRef.removeFromBlock(1);
813         Statement nestedTest = sourceTest.newInstance();
```

```
814         Statement nestedRef = sourceRef.newInstance();
815         Condition cTest =
    sourceTest.disassembleIf(nestedTest);
816         Condition cRef = sourceRef.disassembleIf(nestedRef);
817         Statement sRef = sourceRef.newInstance();
818         sRef.assembleWhile(cRef, nestedRef);
819         Statement sTest = sourceTest.newInstance();
820
821         /*
822          * The call
823          */
824         sTest.assembleWhile(cTest, nestedTest);
825
826         /*
827          * Evaluation
828          */
829         assertEquals(emptyBlock, nestedTest);
830         assertEquals(sRef, sTest);
831     }
832
833     /**
834      * Test assembleWhile.
835      */
836     @Test
837     public final void testAssembleWhile2() {
838         /*
839          * Setup
840          */
841         Statement blockTest =
    this.createFromFileTest(FILE_NAME_2);
842         Statement blockRef =
    this.createFromFileRef(FILE_NAME_2);
843         Statement emptyBlock = blockRef.newInstance();
844         Statement sourceTest = blockTest.removeFromBlock(8);
845         Statement sourceRef = blockRef.removeFromBlock(8);
846         Statement nestedTest = sourceTest.newInstance();
847         Statement nestedRef = sourceRef.newInstance();
848         Condition cTest =
    sourceTest.disassembleIf(nestedTest);
```

```
849         Condition cRef = sourceRef.disassembleIf(nestedRef);
850         Statement sRef = sourceRef.newInstance();
851         sRef.assembleWhile(cRef, nestedRef);
852         Statement sTest = sourceTest.newInstance();
853
854         /*
855          * The call
856          */
857         sTest.assembleWhile(cTest, nestedTest);
858
859         /*
860          * Evaluation
861          */
862         assertEquals(emptyBlock, nestedTest);
863         assertEquals(sRef, sTest);
864     }
865
866     /**
867      * Test assembleWhile.
868      */
869     @Test
870     public final void testAssembleWhile3() {
871         /*
872          * Setup
873          */
874         Statement blockTest =
    this.createFromFileTest(FILE_NAME_3);
875         Statement blockRef =
    this.createFromFileRef(FILE_NAME_3);
876         Statement emptyBlock = blockRef.newInstance();
877         Statement sourceTest = blockTest.removeFromBlock(4);
878         Statement sourceRef = blockRef.removeFromBlock(4);
879         Statement nestedTest = sourceTest.newInstance();
880         Statement nestedRef = sourceRef.newInstance();
881         Condition cTest =
    sourceTest.disassembleIf(nestedTest);
882         Condition cRef = sourceRef.disassembleIf(nestedRef);
883         Statement sRef = sourceRef.newInstance();
884         sRef.assembleWhile(cRef, nestedRef);
```

```
885         Statement sTest = sourceTest.newInstance();
886
887         /*
888          * The call
889          */
890         sTest.assembleWhile(cTest, nestedTest);
891
892         /*
893          * Evaluation
894          */
895         assertEquals(emptyBlock, nestedTest);
896         assertEquals(sRef, sTest);
897     }
898
899     /**
900      * Test disassembleWhile.
901      */
902     @Test
903     public final void testDisassembleWhile1() {
904         /*
905          * Setup
906          */
907         final int whilePos = 3;
908         Statement blockTest =
    this.createFromFileTest(FILE_NAME_1);
909         Statement blockRef =
    this.createFromFileRef(FILE_NAME_1);
910         Statement sTest = blockTest.removeFromBlock(whilePos);
911         Statement sRef = blockRef.removeFromBlock(whilePos);
912         Statement nestedTest = sTest.newInstance();
913         Statement nestedRef = sRef.newInstance();
914         Condition cRef = sRef.disassembleWhile(nestedRef);
915
916         /*
917          * The call
918          */
919         Condition cTest = sTest.disassembleWhile(nestedTest);
920
921         /*
```

```
922         * Evaluation
923         */
924        assertEquals(nestedRef, nestedTest);
925        assertEquals(sRef, sTest);
926        assertEquals(cRef, cTest);
927    }
928
929    /**
930     * Test disassembleWhile.
931     */
932    @Test
933    public final void testDisassembleWhile2() {
934        /*
935         * Setup
936         */
937        final int whilePos = 5;
938        Statement blockTest =
    this.createFromFileTest(FILE_NAME_2);
939        Statement blockRef =
    this.createFromFileRef(FILE_NAME_2);
940        Statement sTest = blockTest.removeFromBlock(whilePos);
941        Statement sRef = blockRef.removeFromBlock(whilePos);
942        Statement nestedTest = sTest.newInstance();
943        Statement nestedRef = sRef.newInstance();
944        Condition cRef = sRef.disassembleWhile(nestedRef);
945
946        /*
947         * The call
948         */
949        Condition cTest = sTest.disassembleWhile(nestedTest);
950
951        /*
952         * Evaluation
953         */
954        assertEquals(nestedRef, nestedTest);
955        assertEquals(sRef, sTest);
956        assertEquals(cRef, cTest);
957    }
958
```

```
959     /**
960      * Test disassembleWhile.
961      */
962     @Test
963     public final void testDisassembleWhile3() {
964         /*
965          * Setup
966          */
967         final int whilePos = 2;
968         Statement blockTest =
    this.createFromFileTest(FILE_NAME_3);
969         Statement blockRef =
    this.createFromFileRef(FILE_NAME_3);
970         Statement sTest = blockTest.removeFromBlock(whilePos);
971         Statement sRef = blockRef.removeFromBlock(whilePos);
972         Statement nestedTest = sTest.newInstance();
973         Statement nestedRef = sRef.newInstance();
974         Condition cRef = sRef.disassembleWhile(nestedRef);
975
976         /*
977          * The call
978          */
979         Condition cTest = sTest.disassembleWhile(nestedTest);
980
981         /*
982          * Evaluation
983          */
984         assertEquals(nestedRef, nestedTest);
985         assertEquals(sRef, sTest);
986         assertEquals(cRef, cTest);
987     }
988
989     /**
990      * Test assembleCall.
991      */
992     @Test
993     public final void testAssembleCall() {
994         /*
995          * Setup
```

```
 996         */
 997        Statement sRef = this.constructorRef().newInstance();
 998        Statement sTest =
    this.constructorTest().newInstance();
 999
1000        String name = "look-for-something";
1001        sRef.assembleCall(name);
1002
1003        /*
1004         * The call
1005         */
1006        sTest.assembleCall(name);
1007
1008        /*
1009         * Evaluation
1010         */
1011        assertEquals(sRef, sTest);
1012    }
1013
1014    /**
1015     * Test disassembleCall.
1016     */
1017    @Test
1018    public final void testDisassembleCall1() {
1019        /*
1020         * Setup
1021         */
1022        Statement blockTest =
    this.createFromFileTest(FILE_NAME_1);
1023        Statement blockRef =
    this.createFromFileRef(FILE_NAME_1);
1024        Statement sTest = blockTest.removeFromBlock(0);
1025        Statement sRef = blockRef.removeFromBlock(0);
1026        String nRef = sRef.disassembleCall();
1027
1028        /*
1029         * The call
1030         */
1031        String nTest = sTest.disassembleCall();
```

```
1032
1033        /*
1034         * Evaluation
1035         */
1036        assertEquals(sRef, sTest);
1037        assertEquals(nRef, nTest);
1038    }
1039
1040    /**
1041     * Test disassembleCall.
1042     */
1043    @Test
1044    public final void testDisassembleCall2() {
1045        /*
1046         * Setup
1047         */
1048        Statement blockTest =
    this.createFromFileTest(FILE_NAME_2);
1049        Statement blockRef =
    this.createFromFileRef(FILE_NAME_2);
1050        Statement sTest = blockTest.removeFromBlock(1);
1051        Statement sRef = blockRef.removeFromBlock(1);
1052        String nRef = sRef.disassembleCall();
1053
1054        /*
1055         * The call
1056         */
1057        String nTest = sTest.disassembleCall();
1058
1059        /*
1060         * Evaluation
1061         */
1062        assertEquals(sRef, sTest);
1063        assertEquals(nRef, nTest);
1064    }
1065
1066    /**
1067     * Test disassembleCall.
1068     */
```

```
1069    @Test
1070    public final void testDisassembleCall3() {
1071        /*
1072         * Setup
1073         */
1074        Statement blockTest =
    this.createFromFileTest(FILE_NAME_3);
1075        Statement blockRef =
    this.createFromFileRef(FILE_NAME_3);
1076        Statement sTest = blockTest.removeFromBlock(3);
1077        Statement sRef = blockRef.removeFromBlock(3);
1078        String nRef = sRef.disassembleCall();
1079
1080        /*
1081         * The call
1082         */
1083        String nTest = sTest.disassembleCall();
1084
1085        /*
1086         * Evaluation
1087         */
1088        assertEquals(sRef, sTest);
1089        assertEquals(nRef, nTest);
1090    }
1091
1092}
1093
```