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

/**
 * JUnit test fixture for {@code Statement}'s constructor and kernel methods.
 *
 * @author Robert Frenken
 * @author Bennett Palmer
 *
 */
public abstract class StatementTest {

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

    private static final String FILE_NAME_2 = "data/statement-sample2.bl";

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

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

    /**
     *
     * Creates and returns a block {@code Statement}, of the type of the
     * implementation under test, from the file with the given name.
     *
     * @param filename
     *            the name of the file to be parsed for the sequence of
     *            statements to go in the block statement
     * @return the constructed block statement
     * @ensures <pre>
     * createFromFile = [the block statement containing the statements
     * parsed from the file]
     * </pre>
     */
    private Statement createFromFileTest(String filename) {
        Statement s = this.constructorTest();
        SimpleReader file = new SimpleReader1L(filename);
        Queue<String> tokens = Tokenizer.tokens(file);
        s.parseBlock(tokens);
        file.close();
        return s;
```

```
68      }
69
70      /**
71       *
72       * Creates and returns a block {@code Statement}, of the reference
73       * implementation type, from the file with the given name.
74       *
75       * @param filename
76       *            the name of the file to be parsed for the sequence of
77       *            statements to go in the block statement
78       * @return the constructed block statement
79       * @ensures <pre>
80       * createFromFile = [the block statement containing the statements
81       * parsed from the file]
82       * </pre>
83       */
84      private Statement createFromFileRef(String filename) {
85          Statement s = this.constructorRef();
86          SimpleReader file = new SimpleReader1L(filename);
87          Queue<String> tokens = Tokenizer.tokens(file);
88          s.parseBlock(tokens);
89          file.close();
90          return s;
91      }
92
93      /**
94       * Test constructor.
95       */
96      @Test
97      public final void testConstructor() {
98          /*
99           * Setup
100          */
101         Statement sRef = this.constructorRef();
102
103         /*
104          * The call
105          */
106         Statement sTest = this.constructorTest();
107
108         /*
109          * Evaluation
110          */
111         assertEquals(sRef, sTest);
112     }
113
114     /**
115      * Test kind of a WHILE statement.
116      */
117     @Test
118     public final void testKindWhile() {
119         /*
120          * Setup
121          */
122         final int whilePos = 3;
123         Statement sourceTest = this.createFromFileTest(FILE_NAME_1);
124         Statement sourceRef = this.createFromFileRef(FILE_NAME_1);
```

```
125         Statement sTest = sourceTest.removeFromBlock(whilePos);
126         Statement sRef = sourceRef.removeFromBlock(whilePos);
127         Kind kRef = sRef.kind();
128
129         /*
130          * The call
131          */
132         Kind kTest = sTest.kind();
133
134         /*
135          * Evaluation
136          */
137         assertEquals(kRef, kTest);
138         assertEquals(sRef, sTest);
139     }
140
141     /**
142      * Test addToBlock at an interior position.
143      */
144     @Test
145     public final void testAddToBlockInterior() {
146         /*
147          * Setup
148          */
149         Statement sTest = this.createFromFileTest(FILE_NAME_1);
150         Statement sRef = this.createFromFileRef(FILE_NAME_1);
151         Statement emptyBlock = sRef.newInstance();
152         Statement nestedTest = sTest.removeFromBlock(1);
153         Statement nestedRef = sRef.removeFromBlock(1);
154         sRef.addToBlock(2, nestedRef);
155
156         /*
157          * The call
158          */
159         sTest.addToBlock(2, nestedTest);
160
161         /*
162          * Evaluation
163          */
164         assertEquals(emptyBlock, nestedTest);
165         assertEquals(sRef, sTest);
166     }
167
168     /**
169      * Test removeFromBlock at the front leaving a non-empty block behind.
170      */
171     @Test
172     public final void testRemoveFromBlockFrontLeavingNonEmpty() {
173         /*
174          * Setup
175          */
176         Statement sTest = this.createFromFileTest(FILE_NAME_1);
177         Statement sRef = this.createFromFileRef(FILE_NAME_1);
178         Statement nestedRef = sRef.removeFromBlock(0);
179
180         /*
181          * The call
```

```
182          */
183         Statement nestedTest = sTest.removeFromBlock(0);
184 
185         /*
186          * Evaluation
187          */
188         assertEquals(sRef, sTest);
189         assertEquals(nestedRef, nestedTest);
190     }
191 
192     /**
193      * Test lengthOfBlock, greater than zero.
194      */
195     @Test
196     public final void testLengthOfBlockNonEmpty() {
197         /*
198          * Setup
199          */
200         Statement sTest = this.createFromFileTest(FILE_NAME_1);
201         Statement sRef = this.createFromFileRef(FILE_NAME_1);
202         int lengthRef = sRef.lengthOfBlock();
203 
204         /*
205          * The call
206          */
207         int lengthTest = sTest.lengthOfBlock();
208 
209         /*
210          * Evaluation
211          */
212         assertEquals(lengthRef, lengthTest);
213         assertEquals(sRef, sTest);
214     }
215 
216     /**
217      * Test assembleIf.
218      */
219     @Test
220     public final void testAssembleIf() {
221         /*
222          * Setup
223          */
224         Statement blockTest = this.createFromFileTest(FILE_NAME_1);
225         Statement blockRef = this.createFromFileRef(FILE_NAME_1);
226         Statement emptyBlock = blockRef.newInstance();
227         Statement sourceTest = blockTest.removeFromBlock(1);
228         Statement sRef = blockRef.removeFromBlock(1);
229         Statement nestedTest = sourceTest.newInstance();
230         Condition c = sourceTest.disassembleIf(nestedTest);
231         Statement sTest = sourceTest.newInstance();
232 
233         /*
234          * The call
235          */
236         sTest.assembleIf(c, nestedTest);
237 
238         /*
```

```
239         * Evaluation
240         */
241        assertEquals(emptyBlock, nestedTest);
242        assertEquals(sRef, sTest);
243    }
244
245    /**
246     * Test disassembleIf.
247     */
248    @Test
249    public final void testDisassembleIf() {
250        /*
251         * Setup
252         */
253        Statement blockTest = this.createFromFileTest(FILE_NAME_1);
254        Statement blockRef = this.createFromFileRef(FILE_NAME_1);
255        Statement sTest = blockTest.removeFromBlock(1);
256        Statement sRef = blockRef.removeFromBlock(1);
257        Statement nestedTest = sTest.newInstance();
258        Statement nestedRef = sRef.newInstance();
259        Condition cRef = sRef.disassembleIf(nestedRef);
260
261        /*
262         * The call
263         */
264        Condition cTest = sTest.disassembleIf(nestedTest);
265
266        /*
267         * Evaluation
268         */
269        assertEquals(nestedRef, nestedTest);
270        assertEquals(sRef, sTest);
271        assertEquals(cRef, cTest);
272    }
273
274    /**
275     * Test assembleIfElse.
276     */
277    @Test
278    public final void testAssembleIfElse() {
279        /*
280         * Setup
281         */
282        final int ifElsePos = 2;
283        Statement blockTest = this.createFromFileTest(FILE_NAME_1);
284        Statement blockRef = this.createFromFileRef(FILE_NAME_1);
285        Statement emptyBlock = blockRef.newInstance();
286        Statement sourceTest = blockTest.removeFromBlock(ifElsePos);
287        Statement sRef = blockRef.removeFromBlock(ifElsePos);
288        Statement thenBlockTest = sourceTest.newInstance();
289        Statement elseBlockTest = sourceTest.newInstance();
290        Condition cTest = sourceTest.disassembleIfElse(thenBlockTest,
291                elseBlockTest);
292        Statement sTest = blockTest.newInstance();
293
294        /*
295         * The call
```

```
296          */
297         sTest.assembleIfElse(cTest, thenBlockTest, elseBlockTest);
298 
299         /*
300          * Evaluation
301          */
302         assertEquals(emptyBlock, thenBlockTest);
303         assertEquals(emptyBlock, elseBlockTest);
304         assertEquals(sRef, sTest);
305     }
306 
307     /**
308      * Test disassembleIfElse.
309      */
310     @Test
311     public final void testDisassembleIfElse() {
312         /*
313          * Setup
314          */
315         final int ifElsePos = 2;
316         Statement blockTest = this.createFromFileTest(FILE_NAME_1);
317         Statement blockRef = this.createFromFileRef(FILE_NAME_1);
318         Statement sTest = blockTest.removeFromBlock(ifElsePos);
319         Statement sRef = blockRef.removeFromBlock(ifElsePos);
320         Statement thenBlockTest = sTest.newInstance();
321         Statement elseBlockTest = sTest.newInstance();
322         Statement thenBlockRef = sRef.newInstance();
323         Statement elseBlockRef = sRef.newInstance();
324         Condition cRef = sRef.disassembleIfElse(thenBlockRef, elseBlockRef);
325 
326         /*
327          * The call
328          */
329         Condition cTest = sTest.disassembleIfElse(thenBlockTest, elseBlockTest);
330 
331         /*
332          * Evaluation
333          */
334         assertEquals(cRef, cTest);
335         assertEquals(thenBlockRef, thenBlockTest);
336         assertEquals(elseBlockRef, elseBlockTest);
337         assertEquals(sRef, sTest);
338     }
339 
340     /**
341      * Test assembleWhile.
342      */
343     @Test
344     public final void testAssembleWhile() {
345         /*
346          * Setup
347          */
348         Statement blockTest = this.createFromFileTest(FILE_NAME_1);
349         Statement blockRef = this.createFromFileRef(FILE_NAME_1);
350         Statement emptyBlock = blockRef.newInstance();
351         Statement sourceTest = blockTest.removeFromBlock(1);
352         Statement sourceRef = blockRef.removeFromBlock(1);
```

```
353         Statement nestedTest = sourceTest.newInstance();
354         Statement nestedRef = sourceRef.newInstance();
355         Condition cTest = sourceTest.disassembleIf(nestedTest);
356         Condition cRef = sourceRef.disassembleIf(nestedRef);
357         Statement sRef = sourceRef.newInstance();
358         sRef.assembleWhile(cRef, nestedRef);
359         Statement sTest = sourceTest.newInstance();
360 
361         /*
362          * The call
363          */
364         sTest.assembleWhile(cTest, nestedTest);
365 
366         /*
367          * Evaluation
368          */
369         assertEquals(emptyBlock, nestedTest);
370         assertEquals(sRef, sTest);
371     }
372 
373     /**
374      * Test disassembleWhile.
375      */
376     @Test
377     public final void testDisassembleWhile() {
378         /*
379          * Setup
380          */
381         final int whilePos = 3;
382         Statement blockTest = this.createFromFileTest(FILE_NAME_1);
383         Statement blockRef = this.createFromFileRef(FILE_NAME_1);
384         Statement sTest = blockTest.removeFromBlock(whilePos);
385         Statement sRef = blockRef.removeFromBlock(whilePos);
386         Statement nestedTest = sTest.newInstance();
387         Statement nestedRef = sRef.newInstance();
388         Condition cRef = sRef.disassembleWhile(nestedRef);
389 
390         /*
391          * The call
392          */
393         Condition cTest = sTest.disassembleWhile(nestedTest);
394 
395         /*
396          * Evaluation
397          */
398         assertEquals(nestedRef, nestedTest);
399         assertEquals(sRef, sTest);
400         assertEquals(cRef, cTest);
401     }
402 
403     /**
404      * Test assembleCall.
405      */
406     @Test
407     public final void testAssembleCall() {
408         /*
409          * Setup
```

```
410         */
411        Statement sRef = this.constructorRef().newInstance();
412        Statement sTest = this.constructorTest().newInstance();
413
414        String name = "look-for-something";
415        sRef.assembleCall(name);
416
417        /*
418         * The call
419         */
420        sTest.assembleCall(name);
421
422        /*
423         * Evaluation
424         */
425        assertEquals(sRef, sTest);
426    }
427
428    /**
429     * Test disassembleCall.
430     */
431    @Test
432    public final void testDisassembleCall() {
433        /*
434         * Setup
435         */
436        Statement blockTest = this.createFromFileTest(FILE_NAME_1);
437        Statement blockRef = this.createFromFileRef(FILE_NAME_1);
438        Statement sTest = blockTest.removeFromBlock(0);
439        Statement sRef = blockRef.removeFromBlock(0);
440        String nRef = sRef.disassembleCall();
441
442        /*
443         * The call
444         */
445        String nTest = sTest.disassembleCall();
446
447        /*
448         * Evaluation
449         */
450        assertEquals(sRef, sTest);
451        assertEquals(nRef, nTest);
452    }
453
454    /**
455     * Test kind of a WHILE statement.
456     */
457    @Test
458    public final void testKindWhileFile2() {
459        /*
460         * Setup
461         */
462        final int whilePos = 2;
463        Statement sourceTest = this.createFromFileTest(FILE_NAME_2);
464        Statement sourceRef = this.createFromFileRef(FILE_NAME_2);
465        Statement sTest = sourceTest.removeFromBlock(whilePos);
466        Statement sRef = sourceRef.removeFromBlock(whilePos);
```

```
467         Kind kRef = sRef.kind();
468 
469         /*
470          * The call
471          */
472         Kind kTest = sTest.kind();
473 
474         /*
475          * Evaluation
476          */
477         assertEquals(kRef, kTest);
478         assertEquals(sRef, sTest);
479     }
480 
481     /**
482      * Test addToBlock at an interior position.
483      */
484     @Test
485     public final void testAddToBlockInteriorFile2() {
486         /*
487          * Setup
488          */
489         Statement sTest = this.createFromFileTest(FILE_NAME_2);
490         Statement sRef = this.createFromFileRef(FILE_NAME_2);
491         Statement emptyBlock = sRef.newInstance();
492         Statement nestedTest = sTest.removeFromBlock(1);
493         Statement nestedRef = sRef.removeFromBlock(1);
494         sRef.addToBlock(2, nestedRef);
495 
496         /*
497          * The call
498          */
499         sTest.addToBlock(2, nestedTest);
500 
501         /*
502          * Evaluation
503          */
504         assertEquals(emptyBlock, nestedTest);
505         assertEquals(sRef, sTest);
506     }
507 
508     /**
509      * Test removeFromBlock at the front leaving a non-empty block behind.
510      */
511     @Test
512     public final void testRemoveFromBlockFrontLeavingNonEmptyFile2() {
513         /*
514          * Setup
515          */
516         Statement sTest = this.createFromFileTest(FILE_NAME_2);
517         Statement sRef = this.createFromFileRef(FILE_NAME_2);
518         Statement nestedRef = sRef.removeFromBlock(0);
519 
520         /*
521          * The call
522          */
523         Statement nestedTest = sTest.removeFromBlock(0);
```

```
524
525         /*
526          * Evaluation
527          */
528         assertEquals(sRef, sTest);
529         assertEquals(nestedRef, nestedTest);
530     }
531
532     /**
533      * Test lengthOfBlock, greater than zero.
534      */
535     @Test
536     public final void testLengthOfBlockNonEmptyFile2() {
537         /*
538          * Setup
539          */
540         Statement sTest = this.createFromFileTest(FILE_NAME_2);
541         Statement sRef = this.createFromFileRef(FILE_NAME_2);
542         int lengthRef = sRef.lengthOfBlock();
543
544         /*
545          * The call
546          */
547         int lengthTest = sTest.lengthOfBlock();
548
549         /*
550          * Evaluation
551          */
552         assertEquals(lengthRef, lengthTest);
553         assertEquals(sRef, sTest);
554     }
555
556     /**
557      * Test assembleIf.
558      */
559     @Test
560     public final void testAssembleIfFile2() {
561         /*
562          * Setup
563          */
564         Statement blockTest = this.createFromFileTest(FILE_NAME_2);
565         Statement blockRef = this.createFromFileRef(FILE_NAME_2);
566         Statement emptyBlock = blockRef.newInstance();
567         Statement sourceTest = blockTest.removeFromBlock(0);
568         Statement sRef = blockRef.removeFromBlock(0);
569         Statement nestedTest = sourceTest.newInstance();
570         Condition c = sourceTest.disassembleIf(nestedTest);
571         Statement sTest = sourceTest.newInstance();
572
573         /*
574          * The call
575          */
576         sTest.assembleIf(c, nestedTest);
577
578         /*
579          * Evaluation
580          */
```

```
581         assertEquals(emptyBlock, nestedTest);
582         assertEquals(sRef, sTest);
583     }
584 
585     /**
586      * Test disassembleIf.
587      */
588     @Test
589     public final void testDisassembleIfFile2() {
590         /*
591          * Setup
592          */
593         Statement blockTest = this.createFromFileTest(FILE_NAME_2);
594         Statement blockRef = this.createFromFileRef(FILE_NAME_2);
595         Statement sTest = blockTest.removeFromBlock(0);
596         Statement sRef = blockRef.removeFromBlock(0);
597         Statement nestedTest = sTest.newInstance();
598         Statement nestedRef = sRef.newInstance();
599         Condition cRef = sRef.disassembleIf(nestedRef);
600 
601         /*
602          * The call
603          */
604         Condition cTest = sTest.disassembleIf(nestedTest);
605 
606         /*
607          * Evaluation
608          */
609         assertEquals(nestedRef, nestedTest);
610         assertEquals(sRef, sTest);
611         assertEquals(cRef, cTest);
612     }
613 
614     /**
615      * Test assembleIfElse.
616      */
617     @Test
618     public final void testAssembleIfElseFile2() {
619         /*
620          * Setup
621          */
622         final int ifElsePos = 1;
623         Statement blockTest = this.createFromFileTest(FILE_NAME_2);
624         Statement blockRef = this.createFromFileRef(FILE_NAME_2);
625         Statement emptyBlock = blockRef.newInstance();
626         Statement sourceTest = blockTest.removeFromBlock(ifElsePos);
627         Statement sRef = blockRef.removeFromBlock(ifElsePos);
628         Statement thenBlockTest = sourceTest.newInstance();
629         Statement elseBlockTest = sourceTest.newInstance();
630         Condition cTest = sourceTest.disassembleIfElse(thenBlockTest,
631                 elseBlockTest);
632         Statement sTest = blockTest.newInstance();
633 
634         /*
635          * The call
636          */
637         sTest.assembleIfElse(cTest, thenBlockTest, elseBlockTest);
```

```
638
639         /*
640          * Evaluation
641          */
642         assertEquals(emptyBlock, thenBlockTest);
643         assertEquals(emptyBlock, elseBlockTest);
644         assertEquals(sRef, sTest);
645     }
646
647     /**
648      * Test disassembleIfElse.
649      */
650     @Test
651     public final void testDisassembleIfElseFile2() {
652         /*
653          * Setup
654          */
655         final int ifElsePos = 1;
656         Statement blockTest = this.createFromFileTest(FILE_NAME_2);
657         Statement blockRef = this.createFromFileRef(FILE_NAME_2);
658         Statement sTest = blockTest.removeFromBlock(ifElsePos);
659         Statement sRef = blockRef.removeFromBlock(ifElsePos);
660         Statement thenBlockTest = sTest.newInstance();
661         Statement elseBlockTest = sTest.newInstance();
662         Statement thenBlockRef = sRef.newInstance();
663         Statement elseBlockRef = sRef.newInstance();
664         Condition cRef = sRef.disassembleIfElse(thenBlockRef, elseBlockRef);
665
666         /*
667          * The call
668          */
669         Condition cTest = sTest.disassembleIfElse(thenBlockTest, elseBlockTest);
670
671         /*
672          * Evaluation
673          */
674         assertEquals(cRef, cTest);
675         assertEquals(thenBlockRef, thenBlockTest);
676         assertEquals(elseBlockRef, elseBlockTest);
677         assertEquals(sRef, sTest);
678     }
679
680     /**
681      * Test assembleWhile.
682      */
683     @Test
684     public final void testAssembleWhileFile2() {
685         /*
686          * Setup
687          */
688         Statement blockTest = this.createFromFileTest(FILE_NAME_2);
689         Statement blockRef = this.createFromFileRef(FILE_NAME_2);
690         Statement emptyBlock = blockRef.newInstance();
691         Statement sourceTest = blockTest.removeFromBlock(0);
692         Statement sourceRef = blockRef.removeFromBlock(0);
693         Statement nestedTest = sourceTest.newInstance();
694         Statement nestedRef = sourceRef.newInstance();
```

```
695         Condition cTest = sourceTest.disassembleIf(nestedTest);
696         Condition cRef = sourceRef.disassembleIf(nestedRef);
697         Statement sRef = sourceRef.newInstance();
698         sRef.assembleWhile(cRef, nestedRef);
699         Statement sTest = sourceTest.newInstance();
700 
701         /*
702          * The call
703          */
704         sTest.assembleWhile(cTest, nestedTest);
705 
706         /*
707          * Evaluation
708          */
709         assertEquals(emptyBlock, nestedTest);
710         assertEquals(sRef, sTest);
711     }
712 
713     /**
714      * Test disassembleWhile.
715      */
716     @Test
717     public final void testDisassembleWhileFile2() {
718         /*
719          * Setup
720          */
721         final int whilePos = 2;
722         Statement blockTest = this.createFromFileTest(FILE_NAME_2);
723         Statement blockRef = this.createFromFileRef(FILE_NAME_2);
724         Statement sTest = blockTest.removeFromBlock(whilePos);
725         Statement sRef = blockRef.removeFromBlock(whilePos);
726         Statement nestedTest = sTest.newInstance();
727         Statement nestedRef = sRef.newInstance();
728         Condition cRef = sRef.disassembleWhile(nestedRef);
729 
730         /*
731          * The call
732          */
733         Condition cTest = sTest.disassembleWhile(nestedTest);
734 
735         /*
736          * Evaluation
737          */
738         assertEquals(nestedRef, nestedTest);
739         assertEquals(sRef, sTest);
740         assertEquals(cRef, cTest);
741     }
742 
743 }
744 
```