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

/**
 * JUnit test fixture for {@code Statement}'s constructor and kernel methods.
 *
 * @author Wayne Heym
 * @author Krish Patel and Chloe Feller
 *
 */
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";

    /**
     * The name of a second file containing a sequence of BL statements.
     */
    private static final String FILE_NAME_2 = "data/statement-test1.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);
```

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

```
125         final int whilePos = 3;
126         Statement sourceTest = this.createFromFileTest(FILE_NAME_1);
127         Statement sourceRef = this.createFromFileRef(FILE_NAME_1);
128         Statement sTest = sourceTest.removeFromBlock(whilePos);
129         Statement sRef = sourceRef.removeFromBlock(whilePos);
130         Kind kRef = sRef.kind();
131
132         /*
133          * The call
134          */
135         Kind kTest = sTest.kind();
136
137         /*
138          * Evaluation
139          */
140         assertEquals(kRef, kTest);
141         assertEquals(sRef, sTest);
142     }
143
144     /**
145      * Test kind of a WHILE statement.
146      */
147     @Test
148     public final void testKindWhileTwo() {
149         /*
150          * Setup
151          */
152         final int whilePos = 3;
153         Statement sourceTest = this.createFromFileTest(FILE_NAME_2);
154         Statement sourceRef = this.createFromFileRef(FILE_NAME_2);
155         Statement sTest = sourceTest.removeFromBlock(whilePos);
156         Statement sRef = sourceRef.removeFromBlock(whilePos);
157         Kind kRef = sRef.kind();
158
159         /*
160          * The call
161          */
162         Kind kTest = sTest.kind();
163
164         /*
165          * Evaluation
166          */
167         assertEquals(kRef, kTest);
168         assertEquals(sRef, sTest);
169     }
170
171     /**
172      * Test addToBlock at an interior position.
173      */
174     @Test
175     public final void testAddToBlockInterior() {
176         /*
177          * Setup
178          */
179         Statement sTest = this.createFromFileTest(FILE_NAME_1);
180         Statement sRef = this.createFromFileRef(FILE_NAME_1);
181         Statement emptyBlock = sRef.newInstance();
```

```
182         Statement nestedTest = sTest.removeFromBlock(1);
183         Statement nestedRef = sRef.removeFromBlock(1);
184         sRef.addToBlock(2, nestedRef);
185 
186         /*
187          * The call
188          */
189         sTest.addToBlock(2, nestedTest);
190 
191         /*
192          * Evaluation
193          */
194         assertEquals(emptyBlock, nestedTest);
195         assertEquals(sRef, sTest);
196     }
197 
198     /**
199      * Test addToBlock at an interior position.
200      */
201     @Test
202     public final void testAddToBlockInteriorTwo() {
203         /*
204          * Setup
205          */
206         Statement sTest = this.createFromFileTest(FILE_NAME_2);
207         Statement sRef = this.createFromFileRef(FILE_NAME_2);
208         Statement emptyBlock = sRef.newInstance();
209         Statement nestedTest = sTest.removeFromBlock(1);
210         Statement nestedRef = sRef.removeFromBlock(1);
211         sRef.addToBlock(2, nestedRef);
212 
213         /*
214          * The call
215          */
216         sTest.addToBlock(2, nestedTest);
217 
218         /*
219          * Evaluation
220          */
221         assertEquals(emptyBlock, nestedTest);
222         assertEquals(sRef, sTest);
223     }
224 
225     /**
226      * Test removeFromBlock at the front leaving a non-empty block behind.
227      */
228     @Test
229     public final void testRemoveFromBlockFrontLeavingNonEmpty() {
230         /*
231          * Setup
232          */
233         Statement sTest = this.createFromFileTest(FILE_NAME_1);
234         Statement sRef = this.createFromFileRef(FILE_NAME_1);
235         Statement nestedRef = sRef.removeFromBlock(0);
236 
237         /*
238          * The call
```

```
239          */
240         Statement nestedTest = sTest.removeFromBlock(0);
241 
242         /*
243          * Evaluation
244          */
245         assertEquals(sRef, sTest);
246         assertEquals(nestedRef, nestedTest);
247     }
248 
249     /**
250      * Test removeFromBlock at the front leaving a non-empty block behind.
251      */
252     @Test
253     public final void testRemoveFromBlockFrontLeavingNonEmptyTwo() {
254         /*
255          * Setup
256          */
257         Statement sTest = this.createFromFileTest(FILE_NAME_2);
258         Statement sRef = this.createFromFileRef(FILE_NAME_2);
259         Statement nestedRef = sRef.removeFromBlock(0);
260 
261         /*
262          * The call
263          */
264         Statement nestedTest = sTest.removeFromBlock(0);
265 
266         /*
267          * Evaluation
268          */
269         assertEquals(sRef, sTest);
270         assertEquals(nestedRef, nestedTest);
271     }
272 
273     /**
274      * Test lengthOfBlock, greater than zero.
275      */
276     @Test
277     public final void testLengthOfBlockNonEmpty() {
278         /*
279          * Setup
280          */
281         Statement sTest = this.createFromFileTest(FILE_NAME_1);
282         Statement sRef = this.createFromFileRef(FILE_NAME_1);
283         int lengthRef = sRef.lengthOfBlock();
284 
285         /*
286          * The call
287          */
288         int lengthTest = sTest.lengthOfBlock();
289 
290         /*
291          * Evaluation
292          */
293         assertEquals(lengthRef, lengthTest);
294         assertEquals(sRef, sTest);
295     }
```

```
296
297     /**
298      * Test lengthOfBlock, greater than zero.
299      */
300     @Test
301     public final void testLengthOfBlockNonEmptyTwo() {
302         /*
303          * Setup
304          */
305         Statement sTest = this.createFromFileTest(FILE_NAME_2);
306         Statement sRef = this.createFromFileRef(FILE_NAME_2);
307         int lengthRef = sRef.lengthOfBlock();
308
309         /*
310          * The call
311          */
312         int lengthTest = sTest.lengthOfBlock();
313
314         /*
315          * Evaluation
316          */
317         assertEquals(lengthRef, lengthTest);
318         assertEquals(sRef, sTest);
319     }
320
321     /**
322      * Test assembleIf.
323      */
324     @Test
325     public final void testAssembleIf() {
326         /*
327          * Setup
328          */
329         Statement blockTest = this.createFromFileTest(FILE_NAME_1);
330         Statement blockRef = this.createFromFileRef(FILE_NAME_1);
331         Statement emptyBlock = blockRef.newInstance();
332         Statement sourceTest = blockTest.removeFromBlock(1);
333         Statement sRef = blockRef.removeFromBlock(1);
334         Statement nestedTest = sourceTest.newInstance();
335         Condition c = sourceTest.disassembleIf(nestedTest);
336         Statement sTest = sourceTest.newInstance();
337
338         /*
339          * The call
340          */
341         sTest.assembleIf(c, nestedTest);
342
343         /*
344          * Evaluation
345          */
346         assertEquals(emptyBlock, nestedTest);
347         assertEquals(sRef, sTest);
348     }
349
350     /**
351      * Test assembleIf.
352      */
```

```
353     @Test
354     public final void testAssembleIfTwo() {
355         /*
356          * Setup
357          */
358         Statement blockTest = this.createFromFileTest(FILE_NAME_2);
359         Statement blockRef = this.createFromFileRef(FILE_NAME_2);
360         Statement emptyBlock = blockRef.newInstance();
361         Statement sourceTest = blockTest.removeFromBlock(1);
362         Statement sRef = blockRef.removeFromBlock(1);
363         Statement nestedTest = sourceTest.newInstance();
364         Condition c = sourceTest.disassembleIf(nestedTest);
365         Statement sTest = sourceTest.newInstance();
366 
367         /*
368          * The call
369          */
370         sTest.assembleIf(c, nestedTest);
371 
372         /*
373          * Evaluation
374          */
375         assertEquals(emptyBlock, nestedTest);
376         assertEquals(sRef, sTest);
377     }
378 
379     /**
380      * Test disassembleIf.
381      */
382     @Test
383     public final void testDisassembleIf() {
384         /*
385          * Setup
386          */
387         Statement blockTest = this.createFromFileTest(FILE_NAME_1);
388         Statement blockRef = this.createFromFileRef(FILE_NAME_1);
389         Statement sTest = blockTest.removeFromBlock(1);
390         Statement sRef = blockRef.removeFromBlock(1);
391         Statement nestedTest = sTest.newInstance();
392         Statement nestedRef = sRef.newInstance();
393         Condition cRef = sRef.disassembleIf(nestedRef);
394 
395         /*
396          * The call
397          */
398         Condition cTest = sTest.disassembleIf(nestedTest);
399 
400         /*
401          * Evaluation
402          */
403         assertEquals(nestedRef, nestedTest);
404         assertEquals(sRef, sTest);
405         assertEquals(cRef, cTest);
406     }
407 
408     /**
409      * Test disassembleIf.
```

```
410     */
411    @Test
412    public final void testDisassembleIfTwo() {
413        /*
414         * Setup
415         */
416        Statement blockTest = this.createFromFileTest(FILE_NAME_2);
417        Statement blockRef = this.createFromFileRef(FILE_NAME_2);
418        Statement sTest = blockTest.removeFromBlock(1);
419        Statement sRef = blockRef.removeFromBlock(1);
420        Statement nestedTest = sTest.newInstance();
421        Statement nestedRef = sRef.newInstance();
422        Condition cRef = sRef.disassembleIf(nestedRef);
423
424        /*
425         * The call
426         */
427        Condition cTest = sTest.disassembleIf(nestedTest);
428
429        /*
430         * Evaluation
431         */
432        assertEquals(nestedRef, nestedTest);
433        assertEquals(sRef, sTest);
434        assertEquals(cRef, cTest);
435    }
436
437    /**
438     * Test assembleIfElse.
439     */
440    @Test
441    public final void testAssembleIfElse() {
442        /*
443         * Setup
444         */
445        final int ifElsePos = 2;
446        Statement blockTest = this.createFromFileTest(FILE_NAME_1);
447        Statement blockRef = this.createFromFileRef(FILE_NAME_1);
448        Statement emptyBlock = blockRef.newInstance();
449        Statement sourceTest = blockTest.removeFromBlock(ifElsePos);
450        Statement sRef = blockRef.removeFromBlock(ifElsePos);
451        Statement thenBlockTest = sourceTest.newInstance();
452        Statement elseBlockTest = sourceTest.newInstance();
453        Condition cTest = sourceTest.disassembleIfElse(thenBlockTest,
454                elseBlockTest);
455        Statement sTest = blockTest.newInstance();
456
457        /*
458         * The call
459         */
460        sTest.assembleIfElse(cTest, thenBlockTest, elseBlockTest);
461
462        /*
463         * Evaluation
464         */
465        assertEquals(emptyBlock, thenBlockTest);
466        assertEquals(emptyBlock, elseBlockTest);
```

```
467         assertEquals(sRef, sTest);
468     }
469
470     /**
471      * Test assembleIfElse.
472      */
473     @Test
474     public final void testAssembleIfElseTwo() {
475         /*
476          * Setup
477          */
478         final int ifElsePos = 2;
479         Statement blockTest = this.createFromFileTest(FILE_NAME_2);
480         Statement blockRef = this.createFromFileRef(FILE_NAME_2);
481         Statement emptyBlock = blockRef.newInstance();
482         Statement sourceTest = blockTest.removeFromBlock(ifElsePos);
483         Statement sRef = blockRef.removeFromBlock(ifElsePos);
484         Statement thenBlockTest = sourceTest.newInstance();
485         Statement elseBlockTest = sourceTest.newInstance();
486         Condition cTest = sourceTest.disassembleIfElse(thenBlockTest,
487                 elseBlockTest);
488         Statement sTest = blockTest.newInstance();
489
490         /*
491          * The call
492          */
493         sTest.assembleIfElse(cTest, thenBlockTest, elseBlockTest);
494
495         /*
496          * Evaluation
497          */
498         assertEquals(emptyBlock, thenBlockTest);
499         assertEquals(emptyBlock, elseBlockTest);
500         assertEquals(sRef, sTest);
501     }
502
503     /**
504      * Test disassembleIfElse.
505      */
506     @Test
507     public final void testDisassembleIfElse() {
508         /*
509          * Setup
510          */
511         final int ifElsePos = 2;
512         Statement blockTest = this.createFromFileTest(FILE_NAME_1);
513         Statement blockRef = this.createFromFileRef(FILE_NAME_1);
514         Statement sTest = blockTest.removeFromBlock(ifElsePos);
515         Statement sRef = blockRef.removeFromBlock(ifElsePos);
516         Statement thenBlockTest = sTest.newInstance();
517         Statement elseBlockTest = sTest.newInstance();
518         Statement thenBlockRef = sRef.newInstance();
519         Statement elseBlockRef = sRef.newInstance();
520         Condition cRef = sRef.disassembleIfElse(thenBlockRef, elseBlockRef);
521
522         /*
523          * The call
```

```
524          */
525         Condition cTest = sTest.disassembleIfElse(thenBlockTest, elseBlockTest);
526 
527         /*
528          * Evaluation
529          */
530         assertEquals(cRef, cTest);
531         assertEquals(thenBlockRef, thenBlockTest);
532         assertEquals(elseBlockRef, elseBlockTest);
533         assertEquals(sRef, sTest);
534     }
535 
536     /**
537      * Test disassembleIfElse.
538      */
539     @Test
540     public final void testDisassembleIfElseTwo() {
541         /*
542          * Setup
543          */
544         final int ifElsePos = 2;
545         Statement blockTest = this.createFromFileTest(FILE_NAME_2);
546         Statement blockRef = this.createFromFileRef(FILE_NAME_2);
547         Statement sTest = blockTest.removeFromBlock(ifElsePos);
548         Statement sRef = blockRef.removeFromBlock(ifElsePos);
549         Statement thenBlockTest = sTest.newInstance();
550         Statement elseBlockTest = sTest.newInstance();
551         Statement thenBlockRef = sRef.newInstance();
552         Statement elseBlockRef = sRef.newInstance();
553         Condition cRef = sRef.disassembleIfElse(thenBlockRef, elseBlockRef);
554 
555         /*
556          * The call
557          */
558         Condition cTest = sTest.disassembleIfElse(thenBlockTest, elseBlockTest);
559 
560         /*
561          * Evaluation
562          */
563         assertEquals(cRef, cTest);
564         assertEquals(thenBlockRef, thenBlockTest);
565         assertEquals(elseBlockRef, elseBlockTest);
566         assertEquals(sRef, sTest);
567     }
568 
569     /**
570      * Test assembleWhile.
571      */
572     @Test
573     public final void testAssembleWhile() {
574         /*
575          * Setup
576          */
577         Statement blockTest = this.createFromFileTest(FILE_NAME_1);
578         Statement blockRef = this.createFromFileRef(FILE_NAME_1);
579         Statement emptyBlock = blockRef.newInstance();
580         Statement sourceTest = blockTest.removeFromBlock(1);
```

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

```
638     @Test
639     public final void testDisassembleWhile() {
640         /*
641          * Setup
642          */
643         final int whilePos = 3;
644         Statement blockTest = this.createFromFileTest(FILE_NAME_1);
645         Statement blockRef = this.createFromFileRef(FILE_NAME_1);
646         Statement sTest = blockTest.removeFromBlock(whilePos);
647         Statement sRef = blockRef.removeFromBlock(whilePos);
648         Statement nestedTest = sTest.newInstance();
649         Statement nestedRef = sRef.newInstance();
650         Condition cRef = sRef.disassembleWhile(nestedRef);
651 
652         /*
653          * The call
654          */
655         Condition cTest = sTest.disassembleWhile(nestedTest);
656 
657         /*
658          * Evaluation
659          */
660         assertEquals(nestedRef, nestedTest);
661         assertEquals(sRef, sTest);
662         assertEquals(cRef, cTest);
663     }
664 
665     /**
666      * Test disassembleWhile.
667      */
668     @Test
669     public final void testDisassembleWhileTwo() {
670         /*
671          * Setup
672          */
673         final int whilePos = 3;
674         Statement blockTest = this.createFromFileTest(FILE_NAME_2);
675         Statement blockRef = this.createFromFileRef(FILE_NAME_2);
676         Statement sTest = blockTest.removeFromBlock(whilePos);
677         Statement sRef = blockRef.removeFromBlock(whilePos);
678         Statement nestedTest = sTest.newInstance();
679         Statement nestedRef = sRef.newInstance();
680         Condition cRef = sRef.disassembleWhile(nestedRef);
681 
682         /*
683          * The call
684          */
685         Condition cTest = sTest.disassembleWhile(nestedTest);
686 
687         /*
688          * Evaluation
689          */
690         assertEquals(nestedRef, nestedTest);
691         assertEquals(sRef, sTest);
692         assertEquals(cRef, cTest);
693     }
694 
```

```
695     /**
696      * Test assembleCall.
697      */
698     @Test
699     public final void testAssembleCall() {
700         /*
701          * Setup
702          */
703         Statement sRef = this.constructorRef().newInstance();
704         Statement sTest = this.constructorTest().newInstance();
705 
706         String name = "look-for-something";
707         sRef.assembleCall(name);
708 
709         /*
710          * The call
711          */
712         sTest.assembleCall(name);
713 
714         /*
715          * Evaluation
716          */
717         assertEquals(sRef, sTest);
718     }
719 
720     /**
721      * Test disassembleCall.
722      */
723     @Test
724     public final void testDisassembleCall() {
725         /*
726          * Setup
727          */
728         Statement blockTest = this.createFromFileTest(FILE_NAME_1);
729         Statement blockRef = this.createFromFileRef(FILE_NAME_1);
730         Statement sTest = blockTest.removeFromBlock(0);
731         Statement sRef = blockRef.removeFromBlock(0);
732         String nRef = sRef.disassembleCall();
733 
734         /*
735          * The call
736          */
737         String nTest = sTest.disassembleCall();
738 
739         /*
740          * Evaluation
741          */
742         assertEquals(sRef, sTest);
743         assertEquals(nRef, nTest);
744     }
745 
746     /**
747      * Test disassembleCall.
748      */
749     @Test
750     public final void testDisassembleCallTwo() {
751         /*
```

```
752          * Setup
753          */
754         Statement blockTest = this.createFromFileTest(FILE_NAME_2);
755         Statement blockRef = this.createFromFileRef(FILE_NAME_2);
756         Statement sTest = blockTest.removeFromBlock(0);
757         Statement sRef = blockRef.removeFromBlock(0);
758         String nRef = sRef.disassembleCall();
759 
760         /*
761          * The call
762          */
763         String nTest = sTest.disassembleCall();
764 
765         /*
766          * Evaluation
767          */
768         assertEquals(sRef, sTest);
769         assertEquals(nRef, nTest);
770     }
771 
772     // TODO - provide additional test cases to thoroughly test StatementKernel
773 
774 }
775 
```