```java
import components.sequence.Sequence;

/**
 * {@code Statement} represented as a {@code Tree<StatementLabel>} with
 * implementations of primary methods.
 *
 * @convention [$this.rep is a valid representation of a Statement]
 * @correspondence this = $this.rep
 *
 * @author Gabe Azzarita and Ty Fredrick
 *
 */
public class Statement2 extends StatementSecondary {

    /*
     * Private members --------------------------------------------------------
     */

    /**
     * Label class for the tree representation.
     */
    private static final class StatementLabel {

        /**
         * Statement kind.
         */
        private Kind kind;

        /**
         * IF/IF_ELSE/WHILE statement condition.
         */
        private Condition condition;

        /**
         * CALL instruction name.
         */
        private String instruction;

        /**
         * Constructor for BLOCK.
         *
         * @param k
         *            the kind of statement
         *
         * @requires k = BLOCK
         * @ensures this = (BLOCK, ?, ?)
         */
        private StatementLabel(Kind k) {
            assert k == Kind.BLOCK : "Violation of: k = BLOCK";
            this.kind = k;
        }

        /**
         * Constructor for IF, IF_ELSE, WHILE.
         *
         * @param k
         *            the kind of statement
         * @param c
         *            the statement condition
```

```
65          *
66          * @requires k = IF or k = IF_ELSE or k = WHILE
67          * @ensures this = (k, c, ?)
68          */
69         private StatementLabel(Kind k, Condition c) {
70             assert k == Kind.IF || k == Kind.IF_ELSE || k == Kind.WHILE : ""
71                     + "Violation of: k = IF or k = IF_ELSE or k = WHILE";
72             this.kind = k;
73             this.condition = c;
74         }
75
76         /**
77          * Constructor for CALL.
78          *
79          * @param k
80          *            the kind of statement
81          * @param i
82          *            the instruction name
83          *
84          * @requires k = CALL and [i is an IDENTIFIER]
85          * @ensures this = (CALL, ?, i)
86          */
87         private StatementLabel(Kind k, String i) {
88             assert k == Kind.CALL : "Violation of: k = CALL";
89             assert i != null : "Violation of: i is not null";
90             assert Tokenizer
91                     .isIdentifier(i) : "Violation of: i is an IDENTIFIER";
92             this.kind = k;
93             this.instruction = i;
94         }
95
96         @Override
97         public String toString() {
98             String condition = "?", instruction = "?";
99             if ((this.kind == Kind.IF) || (this.kind == Kind.IF_ELSE)
100                    || (this.kind == Kind.WHILE)) {
101                condition = this.condition.toString();
102            } else if (this.kind == Kind.CALL) {
103                instruction = this.instruction;
104            }
105            return "(" + this.kind + "," + condition + "," + instruction + ")";
106        }
107
108    }
109
110    /**
111     * The tree representation field.
112     */
113    private Tree<StatementLabel> rep;
114
115    /**
116     * Creator of initial representation.
117     */
118    private void createNewRep() {
119
120        this.rep = new Tree1<StatementLabel>();
121        Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
122        StatementLabel root = new StatementLabel(Kind.BLOCK);
123
```

```
124         this.rep.assemble(root, children);
125     }
126
127     /*
128      * Constructors ------------------------------------------------------------
129      */
130
131     /**
132      * No-argument constructor.
133      */
134     public Statement2() {
135         this.createNewRep();
136     }
137
138     /*
139      * Standard methods --------------------------------------------------------
140      */
141
142     @Override
143     public final Statement2 newInstance() {
144         try {
145             return this.getClass().getConstructor().newInstance();
146         } catch (ReflectiveOperationException e) {
147             throw new AssertionError(
148                     "Cannot construct object of type " + this.getClass());
149         }
150     }
151
152     @Override
153     public final void clear() {
154         this.createNewRep();
155     }
156
157     @Override
158     public final void transferFrom(Statement source) {
159         assert source != null : "Violation of: source is not null";
160         assert source != this : "Violation of: source is not this";
161         assert source instanceof Statement2 : ""
162                 + "Violation of: source is of dynamic type Statement2";
163         /*
164          * This cast cannot fail since the assert above would have stopped
165          * execution in that case: source must be of dynamic type Statement2.
166          */
167         Statement2 localSource = (Statement2) source;
168         this.rep = localSource.rep;
169         localSource.createNewRep();
170     }
171
172     /*
173      * Kernel methods ----------------------------------------------------------
174      */
175
176     @Override
177     public final Kind kind() {
178
179         return this.rep.root().kind;
180     }
181
182     @Override
```

```
183     public final void addToBlock(int pos, Statement s) {
184         assert s != null : "Violation of: s is not null";
185         assert s != this : "Violation of: s is not this";
186         assert s instanceof Statement2 : "Violation of: s is a Statement2";
187         assert this.kind() == Kind.BLOCK : ""
188                 + "Violation of: [this is a BLOCK statement]";
189         assert 0 <= pos : "Violation of: 0 <= pos";
190         assert pos <= this.lengthOfBlock() : ""
191                 + "Violation of: pos <= [length of this BLOCK]";
192         assert s.kind() != Kind.BLOCK : "Violation of: [s is not a BLOCK statement]";
193
194         Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
195         StatementLabel root = this.rep.disassemble(children);
196         Statement2 localS = (Statement2) s;
197
198         // add statement to desired position, clear localS, then assemble tree
199         children.add(pos, localS.rep);
200         localS.createNewRep(); // clear localS
201         this.rep.assemble(root, children);
202
203     }
204
205     @Override
206     public final Statement removeFromBlock(int pos) {
207         assert 0 <= pos : "Violation of: 0 <= pos";
208         assert pos < this.lengthOfBlock() : ""
209                 + "Violation of: pos < [length of this BLOCK]";
210         assert this.kind() == Kind.BLOCK : ""
211                 + "Violation of: [this is a BLOCK statement]";
212         /*
213          * The following call to Statement newInstance method is a violation of
214          * the kernel purity rule. However, there is no way to avoid it and it
215          * is safe because the convention clearly holds at this point in the
216          * code.
217          */
218         Statement2 s = this.newInstance();
219         Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
220         StatementLabel root = this.rep.disassemble(children);
221
222         // remove desired position then reassemble tree
223         s.rep = children.remove(pos);
224         this.rep.assemble(root, children);
225
226         return s;
227     }
228
229     @Override
230     public final int lengthOfBlock() {
231         assert this.kind() == Kind.BLOCK : ""
232                 + "Violation of: [this is a BLOCK statement]";
233
234         return this.rep.numberOfSubtrees();
235     }
236
237     @Override
238     public final void assembleIf(Condition c, Statement s) {
239         assert c != null : "Violation of: c is not null";
240         assert s != null : "Violation of: s is not null";
241         assert s != this : "Violation of: s is not this";
```

```
242         assert s instanceof Statement2 : "Violation of: s is a Statement2";
243         assert s.kind() == Kind.BLOCK : ""
244                 + "Violation of: [s is a BLOCK statement]";
245         Statement2 localS = (Statement2) s;
246         StatementLabel label = new StatementLabel(Kind.IF, c);
247         Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
248         children.add(0, localS.rep);
249         this.rep.assemble(label, children);
250         localS.createNewRep(); // clears s
251     }
252
253     @Override
254     public final Condition disassembleIf(Statement s) {
255         assert s != null : "Violation of: s is not null";
256         assert s != this : "Violation of: s is not this";
257         assert s instanceof Statement2 : "Violation of: s is a Statement2";
258         assert this.kind() == Kind.IF : ""
259                 + "Violation of: [this is an IF statement]";
260         Statement2 localS = (Statement2) s;
261         Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
262         StatementLabel label = this.rep.disassemble(children);
263         localS.rep = children.remove(0);
264         this.createNewRep(); // clears this
265         return label.condition;
266     }
267
268     @Override
269     public final void assembleIfElse(Condition c, Statement s1, Statement s2) {
270         assert c != null : "Violation of: c is not null";
271         assert s1 != null : "Violation of: s1 is not null";
272         assert s2 != null : "Violation of: s2 is not null";
273         assert s1 != this : "Violation of: s1 is not this";
274         assert s2 != this : "Violation of: s2 is not this";
275         assert s1 != s2 : "Violation of: s1 is not s2";
276         assert s1 instanceof Statement2 : "Violation of: s1 is a Statement2";
277         assert s2 instanceof Statement2 : "Violation of: s2 is a Statement2";
278         assert s1
279                 .kind() == Kind.BLOCK : "Violation of: [s1 is a BLOCK statement]";
280         assert s2
281                 .kind() == Kind.BLOCK : "Violation of: [s2 is a BLOCK statement]";
282
283         Statement2 ifBlock = (Statement2) s1;
284         Statement2 elseBlock = (Statement2) s2;
285         StatementLabel label = new StatementLabel(Kind.IF_ELSE, c);
286         Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
287
288         // add ifBlock and elseBlock to sequence, then assemble tree
289         children.add(0, ifBlock.rep);
290         children.add(1, elseBlock.rep);
291         this.rep.assemble(label, children);
292
293         ifBlock.createNewRep(); // clears s1
294         elseBlock.createNewRep(); // clears s2
295
296     }
297
298     @Override
299     public final Condition disassembleIfElse(Statement s1, Statement s2) {
300         assert s1 != null : "Violation of: s1 is not null";
```

```
301         assert s2 != null : "Violation of: s1 is not null";
302         assert s1 != this : "Violation of: s1 is not this";
303         assert s2 != this : "Violation of: s2 is not this";
304         assert s1 != s2 : "Violation of: s1 is not s2";
305         assert s1 instanceof Statement2 : "Violation of: s1 is a Statement2";
306         assert s2 instanceof Statement2 : "Violation of: s2 is a Statement2";
307         assert this.kind() == Kind.IF_ELSE : ""
308                 + "Violation of: [this is an IF_ELSE statement]";
309
310         Statement2 ifBlock = (Statement2) s1;
311         Statement2 elseBlock = (Statement2) s2;
312         Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
313         StatementLabel label = this.rep.disassemble(children);
314
315         // remove ifBlock and elseBlock from sequence
316         ifBlock.rep = children.remove(0);
317         elseBlock.rep = children.remove(0);
318
319         this.createNewRep(); // clears this
320         return label.condition;
321
322     }
323
324     @Override
325     public final void assembleWhile(Condition c, Statement s) {
326         assert c != null : "Violation of: c is not null";
327         assert s != null : "Violation of: s is not null";
328         assert s != this : "Violation of: s is not this";
329         assert s instanceof Statement2 : "Violation of: s is a Statement2";
330         assert s.kind() == Kind.BLOCK : "Violation of: [s is a BLOCK statement]";
331
332         Statement2 localS = (Statement2) s;
333         StatementLabel label = new StatementLabel(Kind.WHILE, c);
334         Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
335
336         // add statement s to children and then assemble tree
337         children.add(0, localS.rep);
338         this.rep.assemble(label, children);
339
340         localS.createNewRep(); // clears s
341
342     }
343
344     @Override
345     public final Condition disassembleWhile(Statement s) {
346         assert s != null : "Violation of: s is not null";
347         assert s != this : "Violation of: s is not this";
348         assert s instanceof Statement2 : "Violation of: s is a Statement2";
349         assert this.kind() == Kind.WHILE : ""
350                 + "Violation of: [this is a WHILE statement]";
351
352         Statement2 localS = (Statement2) s;
353         Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
354         StatementLabel label = this.rep.disassemble(children);
355
356         // remove block from sequence
357         localS.rep = children.remove(0);
358
359         this.createNewRep(); // clears this
```

```
360         return label.condition;
361
362     }
363
364     @Override
365     public final void assembleCall(String inst) {
366         assert inst != null : "Violation of: inst is not null";
367         assert Tokenizer.isIdentifier(inst) : ""
368                 + "Violation of: inst is a valid IDENTIFIER";
369
370         StatementLabel label = new StatementLabel(Kind.CALL, inst);
371         Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
372
373         this.rep.assemble(label, children);
374
375     }
376
377     @Override
378     public final String disassembleCall() {
379         assert this.kind() == Kind.CALL : ""
380                 + "Violation of: [this is a CALL statement]";
381
382         Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
383         StatementLabel label = this.rep.disassemble(children);
384
385         // clear this
386         this.createNewRep();
387         return label.instruction;
388
389     }
390
391 }
392
```