

```

1 import components.sequence.Sequence;
2
3 /**
4  * {@code Statement} represented as a {@code Tree<StatementLabel>} with
5  * implementations of primary methods.
6  *
7  * @convention [$this.rep is a valid representation of a Statement]
8  * @correspondence this = $this.rep
9  *
10 * @author Qinuo Shi & Yiming Cheng
11 */
12
13 public class Statement2 extends StatementSecondary {
14
15     /**
16      * Private members -----
17      */
18
19     /**
20      * Label class for the tree representation.
21      */
22     private static final class StatementLabel {
23
24         /**
25          * Statement kind.
26          */
27         private Kind kind;
28
29         /**
30          * IF/IF_ELSE/WHILE statement condition.
31          */
32         private Condition condition;
33
34         /**
35          * CALL instruction name.
36          */
37         private String instruction;
38
39         /**
40          * Constructor for BLOCK.
41          *
42          * @param k
43          *         the kind of statement
44          */
45         private StatementLabel(Kind k) {
46             assert k == Kind.BLOCK : "Violation of: k = BLOCK";
47             this.kind = k;
48         }
49
50         /**
51          * Constructor for IF, IF_ELSE, WHILE.
52          *
53          * @param k
54          *         the kind of statement
55          * @param c
56          *         the statement condition
57          */
58     }
59
60 }
61
62

```

```

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

```

```

120     * Constructors -----
121     */
122
123     /**
124     * No-argument constructor.
125     */
126     public Statement2() {
127         this.createNewRep();
128     }
129
130     /*
131     * Standard methods -----
132     */
133
134     @Override
135     public final Statement2 newInstance() {
136         try {
137             return this.getClass().getConstructor().newInstance();
138         } catch (ReflectiveOperationException e) {
139             throw new AssertionError(
140                 "Cannot construct object of type " + this.getClass());
141         }
142     }
143
144     @Override
145     public final void clear() {
146         this.createNewRep();
147     }
148
149     @Override
150     public final void transferFrom(Statement source) {
151         assert source != null : "Violation of: source is not null";
152         assert source != this : "Violation of: source is not this";
153         assert source instanceof Statement2 : ""
154             + "Violation of: source is of dynamic type Statement2";
155         /*
156         * This cast cannot fail since the assert above would have stopped
157         * execution in that case: source must be of dynamic type Statement2.
158         */
159         Statement2 localSource = (Statement2) source;
160         this.rep = localSource.rep;
161         localSource.createNewRep();
162     }
163
164     /*
165     * Kernel methods -----
166     */
167
168     @Override
169     public final Kind kind() {
170
171         // TODO - fill in body
172
173         // Fix this line to return the result.
174         return this.rep.root().kind;
175     }
176

```

```

177     @Override
178     public final void addToBlock(int pos, Statement s) {
179         assert s != null : "Violation of: s is not null";
180         assert s != this : "Violation of: s is not this";
181         assert s instanceof Statement2 : "Violation of: s is a Statement2";
182         assert this.kind() == Kind.BLOCK : ""
183             + "Violation of: [this is a BLOCK statement]";
184         assert 0 <= pos : "Violation of: 0 <= pos";
185         assert pos <= this.lengthOfBlock() : ""
186             + "Violation of: pos <= [length of this BLOCK]";
187         assert s.kind() != Kind.BLOCK : "Violation of: [s is not a BLOCK statement]";
188
189         // TODO - fill in body
190         Sequence<Tree<StatementLabel>> child = this.rep.newSequenceOfTree();
191         StatementLabel label = this.rep.disassemble(child);
192
193         Statement2 l = (Statement2) s;
194         child.add(pos, l.rep);
195
196         this.rep.assemble(label, child);
197
198         l.createNewRep();
199     }
200 }
201
202     @Override
203     public final Statement removeFromBlock(int pos) {
204         assert 0 <= pos : "Violation of: 0 <= pos";
205         assert pos < this.lengthOfBlock() : ""
206             + "Violation of: pos < [length of this BLOCK]";
207         assert this.kind() == Kind.BLOCK : ""
208             + "Violation of: [this is a BLOCK statement]";
209         /*
210          * The following call to Statement newInstance method is a violation of
211          * the kernel purity rule. However, there is no way to avoid it and it
212          * is safe because the convention clearly holds at this point in the
213          * code.
214          */
215         Statement2 s = this.newInstance();
216
217         // TODO - fill in body
218
219         Sequence<Tree<StatementLabel>> child = this.rep.newSequenceOfTree();
220         StatementLabel thisStatementLabel = this.rep.disassemble(child);
221         Tree<StatementLabel> tree = child.remove(pos);
222
223         this.rep.assemble(thisStatementLabel, child);
224         s.rep = tree;
225         return s;
226     }
227
228     @Override
229     public final int lengthOfBlock() {
230         assert this.kind() == Kind.BLOCK : ""
231             + "Violation of: [this is a BLOCK statement]";
232
233         // TODO - fill in body

```

```

234     Sequence<Tree<StatementLabel>> child = this.rep.newSequenceOfTree();
235     StatementLabel root = this.rep.disassemble(child);
236     int len = child.length();
237     this.rep.assemble(root, child);
238
239     // Fix this line to return the result.
240     return len;
241 }
242
243 @Override
244 public final void assembleIf(Condition c, Statement s) {
245     assert c != null : "Violation of: c is not null";
246     assert s != null : "Violation of: s is not null";
247     assert s != this : "Violation of: s is not this";
248     assert s instanceof Statement2 : "Violation of: s is a Statement2";
249     assert s.kind() == Kind.BLOCK : ""
250         + "Violation of: [s is a BLOCK statement]";
251     Statement2 localS = (Statement2) s;
252     StatementLabel label = new StatementLabel(Kind.IF, c);
253     Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
254     children.add(0, localS.rep);
255     this.rep.assemble(label, children);
256     localS.createNewRep(); // clears s
257 }
258
259 @Override
260 public final Condition disassembleIf(Statement s) {
261     assert s != null : "Violation of: s is not null";
262     assert s != this : "Violation of: s is not this";
263     assert s instanceof Statement2 : "Violation of: s is a Statement2";
264     assert this.kind() == Kind.IF : ""
265         + "Violation of: [this is an IF statement]";
266     Statement2 localS = (Statement2) s;
267     Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
268     StatementLabel label = this.rep.disassemble(children);
269     localS.rep = children.remove(0);
270     this.createNewRep(); // clears this
271     return label.condition;
272 }
273
274 @Override
275 public final void assembleIfElse(Condition c, Statement s1, Statement s2) {
276     assert c != null : "Violation of: c is not null";
277     assert s1 != null : "Violation of: s1 is not null";
278     assert s2 != null : "Violation of: s2 is not null";
279     assert s1 != this : "Violation of: s1 is not this";
280     assert s2 != this : "Violation of: s2 is not this";
281     assert s1 != s2 : "Violation of: s1 is not s2";
282     assert s1 instanceof Statement2 : "Violation of: s1 is a Statement2";
283     assert s2 instanceof Statement2 : "Violation of: s2 is a Statement2";
284     assert s1
285         .kind() == Kind.BLOCK : "Violation of: [s1 is a BLOCK statement]";
286     assert s2
287         .kind() == Kind.BLOCK : "Violation of: [s2 is a BLOCK statement]";
288
289     // TODO - fill in body
290     StatementLabel root = new StatementLabel(Kind.IF_ELSE, c);

```

```

291
292     Sequence<Tree<StatementLabel>> child = this.rep.newSequenceOfTree();
293
294     Statement2 s1l = (Statement2) s1;
295     Statement2 s2l = (Statement2) s2;
296     Tree<StatementLabel> s1t = s1l.rep;
297     Tree<StatementLabel> s2t = s2l.rep;
298
299     child.add(0, s1t);
300     child.add(1, s2t);
301
302     this.rep.assemble(root, child);
303     s1l.createNewRep();
304     s2l.createNewRep();
305
306 }
307
308 @Override
309 public final Condition disassembleIfElse(Statement s1, Statement s2) {
310     assert s1 != null : "Violation of: s1 is not null";
311     assert s2 != null : "Violation of: s2 is not null";
312     assert s1 != this : "Violation of: s1 is not this";
313     assert s2 != this : "Violation of: s2 is not this";
314     assert s1 != s2 : "Violation of: s1 is not s2";
315     assert s1 instanceof Statement2 : "Violation of: s1 is a Statement2";
316     assert s2 instanceof Statement2 : "Violation of: s2 is a Statement2";
317     assert this.kind() == Kind.IF_ELSE : ""
318         + "Violation of: [this is an IF_ELSE statement]";
319
320     // TODO - fill in body
321     Statement2 s1l = (Statement2) s1;
322     Statement2 s2l = (Statement2) s2;
323
324     Sequence<Tree<StatementLabel>> child = this.rep.newSequenceOfTree();
325     StatementLabel root = this.rep.disassemble(child);
326
327     s1l.rep = child.remove(0);
328     s2l.rep = child.remove(0);
329
330     this.createNewRep();
331
332     // Fix this line to return the result.
333     return root.condition;
334 }
335
336 @Override
337 public final void assembleWhile(Condition c, Statement s) {
338     assert c != null : "Violation of: c is not null";
339     assert s != null : "Violation of: s is not null";
340     assert s != this : "Violation of: s is not this";
341     assert s instanceof Statement2 : "Violation of: s is a Statement2";
342     assert s.kind() == Kind.BLOCK : "Violation of: [s is a BLOCK statement]";
343
344     // TODO - fill in body
345     StatementLabel root = new StatementLabel(Kind.WHILE, c);
346
347     Sequence<Tree<StatementLabel>> child = this.rep.newSequenceOfTree();

```

```

348
349     Statement2 s1l = (Statement2) s;
350     Tree<StatementLabel> s1t = s1l.rep;
351
352     child.add(0, s1t);
353
354     this.rep.assemble(root, child);
355     s1l.createNewRep();
356 }
357
358 @Override
359 public final Condition disassembleWhile(Statement s) {
360     assert s != null : "Violation of: s is not null";
361     assert s != this : "Violation of: s is not this";
362     assert s instanceof Statement2 : "Violation of: s is a Statement2";
363     assert this.kind() == Kind.WHILE : ""
364         + "Violation of: [this is a WHILE statement]";
365
366     // TODO - fill in body
367     Statement2 s1l = (Statement2) s;
368
369     Sequence<Tree<StatementLabel>> child = this.rep.newSequenceOfTree();
370     StatementLabel root = this.rep.disassemble(child);
371
372     s1l.rep = child.remove(0);
373
374     this.createNewRep();
375
376     // Fix this line to return the result.
377     return root.condition;
378 }
379
380 @Override
381 public final void assembleCall(String inst) {
382     assert inst != null : "Violation of: inst is not null";
383     assert Tokenizer.isIdentifier(inst) : ""
384         + "Violation of: inst is a valid IDENTIFIER";
385
386     // TODO - fill in body
387     StatementLabel root = new StatementLabel(Kind.CALL, inst);
388
389     Sequence<Tree<StatementLabel>> child = this.rep.newSequenceOfTree();
390
391     this.rep.assemble(root, child);
392 }
393
394 @Override
395 public final String disassembleCall() {
396     assert this.kind() == Kind.CALL : ""
397         + "Violation of: [this is a CALL statement]";
398
399     // TODO - fill in body
400     Sequence<Tree<StatementLabel>> child = this.rep.newSequenceOfTree();
401     StatementLabel root = this.rep.disassemble(child);
402
403     this.createNewRep();
404

```

```
405         // Fix this line to return the result.
406         return root.instruction;
407     }
408
409 }
410
```