

Statement2.java

```

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 Robert Frenken
11 * @author Bennett Palmer
12 */
13
14 public class Statement2 extends StatementSecondary {
15
16     /**
17      * Private members -----
18      */
19
20     /**
21      * Label class for the tree representation.
22      */
23     private static final class StatementLabel {
24
25         /**
26          * Statement kind.
27          */
28         private Kind kind;
29
30         /**
31          * IF/IF_ELSE/WHILE statement condition.
32          */
33         private Condition condition;
34
35         /**
36          * CALL instruction name.
37          */
38         private String instruction;
39
40         /**
41          * Constructor for BLOCK.
42          *
43          * @param k
44          *         the kind of statement
45          */
46         private StatementLabel(Kind k) {
47             assert k == Kind.BLOCK : "Violation of: k = BLOCK";
48             this.kind = k;
49         }
50
51         /**
52          * Constructor for IF, IF_ELSE, WHILE.
53          *
54          * @param k
55          *         the kind of statement
56          * @param c
57          *         the statement condition
58          */
59     }
60 }

```

Statement2.java

```

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

```

Statement2.java

```

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      // Fix this line to return the result.
172      return this.rep.root().kind;
173  }
174
175  @Override
176  public final void addToBlock(int pos, Statement s) {

```

Statement2.java

```

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

```

Statement2.java

```

234     Statement2 localS = (Statement2) s;
235     StatementLabel label = new StatementLabel(Kind.IF, c);
236     Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
237     children.add(0, localS.rep);
238     this.rep.assemble(label, children);
239     localS.createNewRep(); // clears s
240 }
241
242 @Override
243 public final Condition disassembleIf(Statement s) {
244     assert s != null : "Violation of: s is not null";
245     assert s != this : "Violation of: s is not this";
246     assert s instanceof Statement2 : "Violation of: s is a Statement2";
247     assert this.kind() == Kind.IF : ""
248         + "Violation of: [this is an IF statement]";
249     Statement2 localS = (Statement2) s;
250     Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
251     StatementLabel label = this.rep.disassemble(children);
252     localS.rep = children.remove(0);
253     this.createNewRep(); // clears this
254     return label.condition;
255 }
256
257 @Override
258 public final void assembleIfElse(Condition c, Statement s1, Statement s2) {
259     assert c != null : "Violation of: c is not null";
260     assert s1 != null : "Violation of: s1 is not null";
261     assert s2 != null : "Violation of: s2 is not null";
262     assert s1 != this : "Violation of: s1 is not this";
263     assert s2 != this : "Violation of: s2 is not this";
264     assert s1 != s2 : "Violation of: s1 is not s2";
265     assert s1 instanceof Statement2 : "Violation of: s1 is a Statement2";
266     assert s2 instanceof Statement2 : "Violation of: s2 is a Statement2";
267     assert s1
268         .kind() == Kind.BLOCK : "Violation of: [s1 is a BLOCK statement]";
269     assert s2
270         .kind() == Kind.BLOCK : "Violation of: [s2 is a BLOCK statement]";
271
272     Statement2 localS1 = (Statement2) s1;
273     Statement2 localS2 = (Statement2) s2;
274     StatementLabel label = new StatementLabel(Kind.IF_ELSE, c);
275     Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
276     children.add(0, localS2.rep);
277     children.add(0, localS1.rep);
278     this.rep.assemble(label, children);
279     localS1.createNewRep();
280     localS2.createNewRep();
281 }
282
283 @Override
284 public final Condition disassembleIfElse(Statement s1, Statement s2) {
285     assert s1 != null : "Violation of: s1 is not null";
286     assert s2 != null : "Violation of: s1 is not null";
287     assert s1 != this : "Violation of: s1 is not this";
288     assert s2 != this : "Violation of: s2 is not this";
289     assert s1 != s2 : "Violation of: s1 is not s2";
290     assert s1 instanceof Statement2 : "Violation of: s1 is a Statement2";

```

Statement2.java

```

291     assert s2 instanceof Statement2 : "Violation of: s2 is a Statement2";
292     assert this.kind() == Kind.IF_ELSE : ""
293         + "Violation of: [this is an IF_ELSE statement]";
294
295     Statement2 localS1 = (Statement2) s1;
296     Statement2 localS2 = (Statement2) s2;
297     Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
298     StatementLabel label = this.rep.disassemble(children);
299     localS1.rep = children.remove(0);
300     localS2.rep = children.remove(0);
301     this.createNewRep();
302     return label.condition;
303
304 }
305
306 @Override
307 public final void assembleWhile(Condition c, Statement s) {
308     assert c != null : "Violation of: c is not null";
309     assert s != null : "Violation of: s is not null";
310     assert s != this : "Violation of: s is not this";
311     assert s instanceof Statement2 : "Violation of: s is a Statement2";
312     assert s.kind() == Kind.BLOCK : "Violation of: [s is a BLOCK statement]";
313
314     Statement2 localS = (Statement2) s;
315     StatementLabel label = new StatementLabel(Kind.WHILE, c);
316     Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
317     children.add(0, localS.rep);
318     this.rep.assemble(label, children);
319     localS.createNewRep(); // clears s
320
321 }
322
323 @Override
324 public final Condition disassembleWhile(Statement s) {
325     assert s != null : "Violation of: s is not null";
326     assert s != this : "Violation of: s is not this";
327     assert s instanceof Statement2 : "Violation of: s is a Statement2";
328     assert this.kind() == Kind.WHILE : ""
329         + "Violation of: [this is a WHILE statement]";
330
331     Statement2 localS = (Statement2) s;
332     Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
333     StatementLabel label = this.rep.disassemble(children);
334     localS.rep = children.remove(0);
335     this.createNewRep();
336     return label.condition;
337 }
338
339 @Override
340 public final void assembleCall(String inst) {
341     assert inst != null : "Violation of: inst is not null";
342     assert Tokenizer.isIdentifier(inst) : ""
343         + "Violation of: inst is a valid IDENTIFIER";
344
345     StatementLabel label = new StatementLabel(Kind.CALL, inst);
346     Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
347

```

Statement2.java

```
348     this.rep.assemble(label, children);
349
350 }
351
352 @Override
353 public final String disassembleCall() {
354     assert this.kind() == Kind.CALL : ""
355         + "Violation of: [this is a CALL statement]";
356
357     Sequence<Tree<StatementLabel>> children = this.rep.newSequenceOfTree();
358     StatementLabel label = this.rep.disassemble(children);
359     this.createNewRep();
360
361     // Fix this line to return the result.
362     return label.instruction;
363 }
364
365 }
366
```