

```
1 import components.statement.Statement;
2
3 /**
4  * Utility class with method to count the number of calls to primitive
5  * instructions (move, turnleft, turnright, infect, skip) in a given
6  * {@code Statement}.
7  *
8  * @author Put your name here
9  *
10 */
11 public final class CountPrimitiveCalls {
12
13     /**
14      * Private constructor so this utility class cannot be instantiated.
15      */
16     private CountPrimitiveCalls() {
17     }
18
19     /**
20      * Reports the number of calls to primitive instructions (move, turnleft,
21      * turnright, infect, skip) in a given {@code Statement}.
22      *
23      * @param s
24      *         the {@code Statement}
25      * @return the number of calls to primitive instructions in {@code s}
26      * @ensures <pre>
27      *   countOfPrimitiveCalls =
28      *     [number of calls to primitive instructions in s]
29      * </pre>
30      */
31     public static int countOfPrimitiveCalls(Statement s) {
32         int count = 0;
33         switch (s.kind()) {
34             case BLOCK: {
35                 /*
36                  * Add up the number of calls to primitive instructions
37                  * in each nested statement in the BLOCK.
38                  */
39
40                 for (int i=0; i < s.lengthOfBlock(); i++) {
41                     count += countOfPrimitiveCalls(s.removeFromBlock(i));
42                 }
43
44                 break;
45             }
46             case IF: {
47                 /*
48                  * Find the number of calls to primitive instructions in
49                  * the body of the IF.
50                  */
51
52                 for (int i=0; i < s.lengthOfBlock(); i++) {
53                     count += countOfPrimitiveCalls(s.removeFromBlock(i));
54                 }
55                 break;
56             }
57             case IF_ELSE: {
```

```

58         /*
59         * Add up the number of calls to primitive instructions in
60         * the "then" and "else" bodies of the IF_ELSE.
61         */
62
63         for (int i=0; i < s.lengthOfBlock(); i++) {
64             count += countOfPrimitiveCalls(s.removeFromBlock(i));
65         }
66
67         break;
68     }
69     case WHILE: {
70         /*
71         * Find the number of calls to primitive instructions in
72         * the body of the WHILE.
73         */
74
75         for (int i=0; i < s.lengthOfBlock(); i++) {
76             count += countOfPrimitiveCalls(s.removeFromBlock(i));
77         }
78
79         break;
80     }
81     case CALL: {
82         /*
83         * This is a leaf: the count can only be 1 or 0. Determine
84         * whether this is a call to a primitive instruction or not.
85         */
86
87
88         // For fucks sake i've felt too sick to go to classes or labs for ANY of my
courses
89         // so i have no idea what I'm meant to be doing and I'm drowning in work that
I
90         // don't even understand
91
92         if (s.getClass().equals()) {
93             count++;
94         }
95
96         break;
97     }
98     default: {
99         // this will never happen...can you explain why?
100
101         // because all possible results are already addressed
102
103         break;
104     }
105 }
106 return count;
107 }
108
109 }
110

```