```java
import components.program.Program;
import components.simplewriter.SimpleWriter;
import components.statement.Statement;

/**
 * {@code Queue} represented as a {@code Sequence} of entries, with
 * implementations of primary methods.
 *
 * @param <T>
 *            type of {@code Queue} entries
 * @correspondence this = $this.entries
 */
public class HelloWorld {

    public static void main(String[] args) {

    }

    /**
     * Pretty prints {@code this} to the given stream {@code out} {@code offset}
     * spaces from the left margin using
     * {@link components.program.Program#INDENT_SIZE Program.INDENT_SIZE} spaces
     * for each indentation level.
     *
     * @param out
     *            the output stream
     * @param offset
     *            the number of spaces to be placed before every nonempty line
     *            of output; nonempty lines of output that are indented further
     *            will, of course, continue with even more spaces
     * @updates out.content
     * @requires out.is_open and 0 <= offset
     * @ensures <pre>
     * out.content =
     *   #out.content * [this pretty printed offset spaces from the left margin
     *                   using Program.INDENT_SIZE spaces for indentation]
     * </pre>
```

```
38      */
39     public void prettyPrint(SimpleWriter out, int offset) {
40         int indent = Program.INDENT_SIZE;
41         switch (this.kind()) {
42             case BLOCK: {
43 
44                 // TODO - fill in case
45                 int length = this.lengthOfBlock();
46                 for (int i = 0; i < length; i++) {
47                     Statement  = this.removeFromBlock(i);
48                     .this.prettyPrint(out, offset);
49                     this.addToBlock(i, );
50                 }
51                 break;
52             }
53             case IF: {
54 
55                 Statement  = this.newInstance();
56                 Condition ifCondition = this.disassembleIf();
57                 printSpaces(out, offset);
58 
59                 out.println("IF " +
   toStringCondition(ifCondition));
60                 .this.prettyPrint(out, offset + indent);
61 
62                 for (int i = 0; i < offset; i++) {
63                     out.print(" ");
64                 }
65                 out.println("END IF");
66                 this.assembleIf(ifCondition, );
67                 break;
68             }
69             case IF_ELSE: {
70 
71                 Statement If = this.newInstance();
72                 Statement Else = this.newInstance();
73                 Condition ifElseCondition =
   this.disassembleIfElse(If,
74                         Else);
75                 printSpaces(out, offset);
76                 out.println(
77                         "IF " + toStringCondition(ifElseCondition)
   + " THEN");
78                 If.prettyPrint(out, offset + indent);
```

```
 79
 80                 printSpaces(out, offset);
 81                 out.println("ELSE");
 82                 Else.prettyPrint(out, offset + indent);
 83
 84                 printSpaces(out, offset);
 85                 out.println("END IF");
 86                 this.assembleIfElse(ifElseCondition, If,
    childBlockElse);
 87
 88                 break;
 89             }
 90             case WHILE: {
 91
 92                 Statement  = this.newInstance();
 93                 Condition whileCondition =
    this.disassembleWhile();
 94                 printSpaces(out, offset);
 95                 out.println(
 96                         "WHILE " +
    toStringCondition(whileCondition) + " DO");
 97                 .this.prettyPrint(out, offset + indent);
 98
 99                 printSpaces(out, offset);
100                 out.println("END WHILE");
101                 this.assembleWhile(whileCondition, );
102
103                 break;
104             }
105             case CALL: {
106
107                 String call = this.disassembleCall();
108                 printSpaces(out, offset);
109                 out.println(call);
110                 this.assembleCall(call);
111
112                 break;
113             }
114             default: {
115                 // this will never happen...
116                 break;
117             }
118         }
119     }
```

```
120 }
```