```java
import components.map.Map;

/**
 * Layered implementation of secondary method {@code prettyPrint} for
 * {@code Program}.
 */
public final class Program1PrettyPrint1 extends Program1 {

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

    /**
     * Constructs into the given {@code Program} the program read from the given
     * input file.
     *
     * @param fileName
     *            the name of the file containing the program
     * @param p
     *            the constructed program
     * @replaces p
     * @requires [fileName is the name of a file containing a valid BL program]
     * @ensures p = [program from file fileName]
     */
    private static void loadProgram(String fileName, Program p) {
        SimpleReader in = new SimpleReader1L(fileName);
        p.parse(in);
        in.close();
    }

    /**
     * Prints the given number of spaces to the given output stream.
     *
     * @param out
     *            the output stream
     * @param numSpaces
     *            the number of spaces to print
     * @updates out.content
     * @requires out.is_open and spaces >= 0
     * @ensures out.content = #out.content * [numSpaces spaces]
     */
    private static void printSpaces(SimpleWriter out, int numSpaces) {
        for (int i = 0; i < numSpaces; i++) {
            out.print(' ');
        }
    }

    /*
     * Constructors -----------------------------------------------------------
     */

    /**
     * No-argument constructor.
     */
    public Program1PrettyPrint1() {
        super();
    }

    /*
```

```
 67      * Secondary methods ------------------------------------------------------
 68      */
 69
 70     @Override
 71     public void prettyPrint(SimpleWriter out) {
 72         assert out != null : "Violation of: out is not null";
 73         assert out.isOpen() : "Violation of: out.is_open";
 74
 75         out.println("PROGRAM " + this.name() + " IS\n");
 76         Map<String, Statement> map = this.newContext();
 77         this.swapContext(map);
 78
 79         for (Map.Pair<String, Statement> i : map) {
 80             printSpaces(out, INDENT_SIZE);
 81             i.key();
 82             out.println("INSTRUCTION " + i.key() + " IS");
 83
 84             i.value().prettyPrint(out, INDENT_SIZE * 2);
 85
 86             printSpaces(out, INDENT_SIZE);
 87             out.println("END " + i.key() + "\n");
 88
 89         }
 90
 91         this.swapContext(map);
 92
 93         out.println("BEGIN");
 94
 95         Statement stmt = this.newBody();
 96         this.swapBody(stmt);
 97
 98         stmt.prettyPrint(out, INDENT_SIZE);
 99
100         this.swapBody(stmt);
101
102         out.println("END " + this.name());
103
104     }
105
106     /*
107      * Main test method -------------------------------------------------------
108      */
109
110     /**
111      * Main method.
112      *
113      * @param args
114      *            the command line arguments
115      */
116     public static void main(String[] args) {
117         SimpleReader in = new SimpleReader1L();
118         SimpleWriter out = new SimpleWriter1L();
119         /*
120          * Get input file name
121          */
122         out.print("Enter valid BL program file name: ");
123         String fileName = in.nextLine();
124         /*
125          * Generate expected output in file "data/expected-output.txt"
```

```
126          */
127         out.println("*** Generating expected output ***");
128         Program p1 = new Program1();
129         loadProgram(fileName, p1);
130         SimpleWriter ppOut = new SimpleWriter1L("data/expected-output.txt");
131         p1.prettyPrint(ppOut);
132         ppOut.close();
133         /*
134          * Generate actual output in file "data/actual-output.txt"
135          */
136         out.println("*** Generating actual output ***");
137         Program p2 = new Program1PrettyPrint1();
138         loadProgram(fileName, p2);
139         ppOut = new SimpleWriter1L("data/actual-output.txt");
140         p2.prettyPrint(ppOut);
141         ppOut.close();
142         /*
143          * Check that prettyPrint restored the value of the program
144          */
145         if (p2.equals(p1)) {
146             out.println("Program value restored correctly.");
147         } else {
148             out.println("Error: program value was not restored.");
149         }
150
151         in.close();
152         out.close();
153     }
154
155 }
156
```