```java
// Copyright 2013 Bill Campbell, Swami Iyer and Bahar Akbal-Delibas

package jminusminus;

import static jminusminus.CLConstants.*;

/**
 * The AST node for an if-statement.
 */

class JIfStatement extends JStatement {

    /** Test expression. */
    private JExpression condition;

    /** Then clause. */
    private JStatement thenPart;

    /** Else clause. */
    private JStatement elsePart;

    /**
     * Construct an AST node for an if-statement given its line number, the test
     * expression, the consequent, and the alternate.
     * 
     * @param line
     *            line in which the if-statement occurs in the source file.
     * @param condition
     *            test expression.
     * @param thenPart
     *            then clause.
     * @param elsePart
     *            else clause.
     */

    public JIfStatement(int line, JExpression condition, JStatement thenPart,
            JStatement elsePart) {
        super(line);
        this.condition = condition;
        this.thenPart = thenPart;
        this.elsePart = elsePart;
    }

    /**
     * Analyzing the if-statement means analyzing its components and checking
     * that the test is boolean.
     * 
     * @param context
     *            context in which names are resolved.
     * @return the analyzed (and possibly rewritten) AST subtree.
     */

    public JStatement analyze(Context context) {
        condition = (JExpression) condition.analyze(context);
        condition.type().mustMatchExpected(line(), Type.BOOLEAN);
        thenPart = (JStatement) thenPart.analyze(context);
        if (elsePart != null) {
            elsePart = (JStatement) elsePart.analyze(context);
        }
        return this;
    }

    /**
     * Code generation for an if-statement. We generate code to branch over the
     * consequent if !test; the consequent is followed by an unconditonal branch
     * over (any) alternate.
     *
     * @param output
     *            the code emitter (basically an abstraction for producing the
     *            .class file).
     */

    public void codegen(CLEmitter output) {
        String elseLabel = output.createLabel();
        String endLabel = output.createLabel();
        condition.codegen(output, elseLabel, false);
        thenPart.codegen(output);
        if (elsePart != null) {
            output.addBranchInstruction(GOTO, endLabel);
        }
        output.addLabel(elseLabel);
        if (elsePart != null) {
            elsePart.codegen(output);
            output.addLabel(endLabel);
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("<JIfStatement line=\"%d\">\n", line());
        p.indentRight();
        p.printf("<TestExpression>\n");
        p.indentRight();
        condition.writeToStdOut(p);
        p.indentLeft();
        p.printf("</TestExpression>\n");
        p.printf("<ThenClause>\n");
        p.indentRight();
        thenPart.writeToStdOut(p);
        p.indentLeft();
        p.printf("</ThenClause>\n");
        if (elsePart != null) {
            p.printf("<ElseClause>\n");
            p.indentRight();
            elsePart.writeToStdOut(p);
            p.indentLeft();
            p.printf("</ElseClause>\n");
        }
        p.indentLeft();
        p.printf("</JIfStatement>\n");
    }

}
```