JBlock.java

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

package jminusminus;

import java.util.ArrayList;

/**
 * The AST node for a block, which delimits a nested level of scope.
 */

class JBlock extends JStatement {

    /** List of statements forming the block body. */
    private ArrayList<JStatement> statements;

    /**
     * The new context (built in analyze()) represented by this block.
     */
    private LocalContext context;

    /**
     * Construct an AST node for a block given its line number, and the list of
     * statements forming the block body.
     *
     * @param line
     *            line in which the block occurs in the source file.
     * @param statements
     *            list of statements forming the block body.
     */

    public JBlock(int line, ArrayList<JStatement> statements) {
        super(line);
        this.statements = statements;
    }

    /**
     * Return the list of statements comprising the block.
     *
     * @return list of statements.
     */

    public ArrayList<JStatement> statements() {
        return statements;
    }

    /**
     * Analyzing a block consists of creating a new nested context for that
     * block and analyzing each of its statements within that context.
     *
     * @param context
     *            context in which names are resolved.
     * @return the analyzed (and possibly rewritten) AST subtree.
     */

    public JBlock analyze(Context context) {
        // { ... } defines a new level of scope.
        this.context = new LocalContext(context);

        for (int i = 0; i < statements.size(); i++) {
            statements.set(i, (JStatement) statements.get(i).analyze(
                    this.context));
        }
        return this;
    }

    /**
```

```
     * Generating code for a block consists of generating code for each of its
     * statements.
     *
     * @param output
     *            the code emitter (basically an abstraction for producing the
     *            .class file).
     */

    public void codegen(CLEmitter output) {
        for (JStatement statement : statements) {
            statement.codegen(output);
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("<JBlock line=\"%d\">\n", line());
        if (context != null) {
            p.indentRight();
            context.writeToStdOut(p);
            p.indentLeft();
        }
        for (JStatement statement : statements) {
            p.indentRight();
            statement.writeToStdOut(p);
            p.indentLeft();
        }
        p.printf("</JBlock>\n");
    }

}
```