JVariableDeclaration.java

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

package jminusminus;

import java.util.ArrayList;

/**
 * The AST node for a local variable declaration. Local variables are declared
 * by analyze(), which also re-writes any initializations as assignment
 * statements, in turn generated by codegen().
 */

class JVariableDeclaration extends JStatement {

    /** Modifiers. */
    private ArrayList<String> mods;

    /** Variable declarators. */
    private ArrayList<JVariableDeclarator> decls;

    /** Variable initializers. */
    private ArrayList<JStatement> initializations;

    /**
     * Construct an AST node for a variable declaration given the line number,
     * modifiers, and the variable declarators.
     *
     * @param line
     *            line in which the variable declaration occurs in the source
     *            file.
     * @param mods
     *            modifiers.
     * @param decls
     *            variable declarators.
     */

    public JVariableDeclaration(int line, ArrayList<String> mods,
            ArrayList<JVariableDeclarator> decls) {
        super(line);
        this.mods = mods;
        this.decls = decls;
        initializations = new ArrayList<JStatement>();
    }

    /**
     * Return the list of modifiers.
     *
     * @return list of modifiers.
     */

    public ArrayList<String> mods() {
        return mods;
    }

    /**
     * We declare the variable(s). Initializations are rewritten as assignment
     * statements.
     *
     * @param context
     *            context in which names are resolved.
     * @return the analyzed (and possibly rewritten) AST subtree.
     */

    public JStatement analyze(Context context) {
        for (JVariableDeclarator decl : decls) {
            // Local variables are declared here (fields are
```

```
            // declared
            // in preAnalyze())
            int offset = ((LocalContext) context).nextOffset();
            LocalVariableDefn defn = new LocalVariableDefn(decl.type().resolve(
                    context), offset);

            // First, check for shadowing
            IDefn previousDefn = context.lookup(decl.name());
            if (previousDefn != null
                    && previousDefn instanceof LocalVariableDefn) {
                JAST.compilationUnit.reportSemanticError(decl.line(),
                        "The name " + decl.name()
                                + " overshadows another local variable.");
            }

            // Then declare it in the local context
            context.addEntry(decl.line(), decl.name(), defn);

            // All initializations must be turned into assignment
            // statements and analyzed
            if (decl.initializer() != null) {
                defn.initialize();
                JAssignOp assignOp = new JAssignOp(decl.line(), new JVariable(
                        decl.line(), decl.name()), decl.initializer());
                assignOp.isStatementExpression = true;
                initializations.add(new JStatementExpression(decl.line(),
                        assignOp).analyze(context));
            }
        }
        return this;
    }

    /**
     * Local variable initializations (rewritten as assignments in analyze())
     * are generated here.
     *
     * @param output
     *            the code emitter (basically an abstraction for producing the
     *            .class file)
     */

    public void codegen(CLEmitter output) {
        for (JStatement initialization : initializations) {
            initialization.codegen(output);
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.println("<JVariableDeclaration>");
        p.indentRight();
        if (mods != null) {
            p.println("<Modifiers>");
            p.indentRight();
            for (String mod : mods) {
                p.printf("<Modifier name=\"%s\"/>\n", mod);
            }
            p.indentLeft();
            p.println("</Modifiers>");
        }
        if (decls != null) {
            p.println("<VariableDeclarators>");
            for (JVariableDeclarator decl : decls) {
                p.indentRight();
                decl.writeToStdOut(p);
                p.indentLeft();
```

```
136             }
137             p.println("</VariableDeclarators>");
138         }
139         p.indentLeft();
140         p.println("</JVariableDeclaration>");
141     }
142 
143 }
144 
```