JNewArrayOp.java

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

package jminusminus;

import java.util.ArrayList;
import static jminusminus.CLConstants.*;

/**
 * The AST node for a "new" array operation. It keeps track of its base type and
 * a list of its dimensions.
 */

class JNewArrayOp extends JExpression {

    /** The base (component) type of the array. */
    private Type typeSpec;

    /** Dimensions of the array. */
    private ArrayList<JExpression> dimExprs;

    /**
     * Construct an AST node for a "new" array operation.
     *
     * @param line
     *            the line in which the operation occurs in the source file.
     * @param typeSpec
     *            the type of the array being created.
     * @param dimExprs
     *            a list of dimension expressions.
     */

    public JNewArrayOp(int line, Type typeSpec, ArrayList<JExpression> dimExprs)
{
        super(line);
        this.typeSpec = typeSpec;
        this.dimExprs = dimExprs;
    }

    /**
     * Analysis of a new array operation involves resolving its type and
     * analyzing the array bounds and checking their types.
     *
     * @param context
     *            context in which names are resolved.
     * @return the analyzed (and possibly rewritten) AST subtree.
     */

    public JExpression analyze(Context context) {
        type = typeSpec.resolve(context);
        for (int i = 0; i < dimExprs.size(); i++) {
            dimExprs.set(i, dimExprs.get(i).analyze(context));
            dimExprs.get(i).type().mustMatchExpected(line, Type.INT);
        }
        return this;
    }

    /**
     * Generate code to push the bounds on the stack and then generate the
     * appropriate array creation instruction.
     *
     * @param output
     *            the code emitter (basically an abstraction for producing the
     *            .class file).
     */

    public void codegen(CLEmitter output) {
        // Code to push diemension exprs on to the stack
        for (JExpression dimExpr : dimExprs) {
            dimExpr.codegen(output);
        }

        // Generate the appropriate array creation instruction
        if (dimExprs.size() == 1) {
            output.addArrayInstruction(
                    type.componentType().isReference() ? ANEWARRAY : NEWARRAY,
                    type.componentType().jvmName());
        } else {
            output.addMULTIANEWARRAYInstruction(type.toDescriptor(), dimExprs
                    .size());
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("<JNewArrayOp line=\"%d\" type=\"%s\"/>\n", line(),
                ((type == null) ? "" : type.toString()));
        p.indentRight();
        p.println("<Dimensions>");
        if (dimExprs != null) {
            p.indentRight();
            for (JExpression dimExpr : dimExprs) {
                p.println("<Dimension>");
                p.indentRight();
                dimExpr.writeToStdOut(p);
                p.indentLeft();
                p.println("</Dimension>");
            }
            p.indentLeft();
        }
        p.println("</Dimensions>");
        p.indentLeft();
        p.println("</JNewArrayOp>");
    }

}
```