JNewOp.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" expression. It keeps track of its type, the
 * Constructor representing the expression, its arguments and their types.
 */

class JNewOp extends JExpression {

    /** The constructor representing this "new" expression. */
    private Constructor constructor;

    /** The arguments to the constructor. */
    private ArrayList<JExpression> arguments;

    /** Types of the arguments. */
    private Type[] argTypes;

    /**
     * Construct an AST node for a "new" expression.
     * 
     * @param line
     *            the line in which the "new" expression occurs in the source
     *            file.
     * @param type
     *            the type being constructed.
     * @param arguments
     *            arguments to the constructor.
     */

    public JNewOp(int line, Type type, ArrayList<JExpression> arguments) {
        super(line);
        this.type = type;
        this.arguments = arguments;
    }

    /**
     * To analyze the new operation, we (1) resolve the type, (2) analyze its
     * arguments, (3) check accessibility of the type, (3) find the appropriate
     * Constructor.
     * 
     * @param context
     *            context in which names are resolved.
     * @return the analyzed (and possibly rewritten) AST subtree.
     */

    public JExpression analyze(Context context) {
        // First resolve the type
        type = type.resolve(context);

        // Analyze the arguments, collecting
        // their types (in Class form) as argTypes.
        argTypes = new Type[arguments.size()];
        for (int i = 0; i < arguments.size(); i++) {
            arguments.set(i, (JExpression) arguments.get(i).analyze(context));
            argTypes[i] = arguments.get(i).type();
        }

        // Can't instantiate an abstract type
        if (type.isAbstract()) {
            JAST.compilationUnit.reportSemanticError(line(),
                    "Cannot instantiate an abstract type:" + type.toString());
        }

        // Where are we now? Check accessability of type
        // resolve() checks accessibility, so the following two
        // is commented
        // Type thisType = context.definingType();
        // thisType.checkAccess( line, type );

        // Then get the proper constructor, given the arguments
        constructor = type.constructorFor(argTypes);

        if (constructor == null) {
            JAST.compilationUnit.reportSemanticError(line(),
                    "Cannot find constructor: "
                            + Type.signatureFor(type.toString(), argTypes));
        }
        return this;
    }

    /**
     * Generating code for a new operation involves generating the NEW
     * instruction for creating the object on the stack, then gnerating the code
     * for the actual arguments, and then code for invoking the constructor (the
     * initialization method).
     *
     * @param output
     *            the code emitter (basically an abstraction for producing the
     *            .class file).
     */

    public void codegen(CLEmitter output) {
        output.addReferenceInstruction(NEW, type.jvmName());
        output.addNoArgInstruction(DUP);
        for (JExpression argument : arguments) {
            argument.codegen(output);
        }
        output.addMemberAccessInstruction(INVOKESPECIAL, type.jvmName(),
                "<init>", constructor.toDescriptor());
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("<JNewOp line=\"%d\" type=\"%s\"/>\n", line(),
                ((type == null) ? "" : type.toString()));
        p.indentRight();
        if (arguments != null) {
            p.println("<Arguments>");
            for (JExpression argument : arguments) {
                p.indentRight();
                p.println("<Argument>");
                p.indentRight();
                argument.writeToStdOut(p);
                p.indentLeft();
                p.println("</Argument>");
                p.indentLeft();
            }
            p.println("</Arguments>");
        }
        p.indentLeft();
        p.println("</JNewOp>");
    }

}
```