JArrayInitializer.java

```java
// Copyright 2011 Bill Campbell, Swami Iyer and Bahar Akbal-Delibas

package jminusminus;

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

/**
 * The AST node for an array initializer. Basically a list of
 * initializing expressions.
 */

class JArrayInitializer
    extends JExpression {

    /** The initializations. */
    private ArrayList<JExpression> initials;

    /**
     * Construct an AST node for an array initializer given the
     * (expected) array type and initial values.
     *
     * @param line
     *                line in which this array initializer occurs
     *                in the source file.
     * @param expected
     *                the type of the array we're initializing.
     * @param initials
     *                initializations.
     */

    public JArrayInitializer(int line, Type expected,
        ArrayList<JExpression> initials) {
        super(line);
        type = expected;
        this.initials = initials;
    }

    /**
     * Analysis of array initializer involves making sure that
     * that the type of the initials is the same as the component
     * type.
     *
     * @param context
     *                context in which names are resolved.
     * @return the analyzed (and possibly rewritten) AST subtree.
     */

    public JExpression analyze(Context context) {
        type = type.resolve(context);
        if (!type.isArray()) {
            JAST.compilationUnit.reportSemanticError(line,
                "Cannot initialize a " + type.toString()
                    + " with an array sequence {...}");
            return this; // un-analyzed
        }
        Type componentType = type.componentType();
        for (int i = 0; i < initials.size(); i++) {
            JExpression component = initials.get(i);
            initials.set(i, component = component.analyze(context));
            if (!(component instanceof JArrayInitializer)) {
                component.type().mustMatchExpected(line,
                    componentType);
            }
        }
        return this;
    }

    /**
     * Perform code generation necessary to construct the
     * initializing array and leave it on top of the stack.
     *
     * @param output
     *                the code emitter (basically an abstraction
     *                for producing the .class file).
     */

    public void codegen(CLEmitter output) {
        Type componentType = type.componentType();

        // Code to push array length.
        new JLiteralInt(line, String.valueOf(initials.size()))
            .codegen(output);

        // Code to create the (empty) array
        output.addArrayInstruction(componentType.isReference()
            ? ANEWARRAY
            : NEWARRAY, componentType.jvmName());

        // Code to load initial values and store them as
        // elements in the newly created array.
        for (int i = 0; i < initials.size(); i++) {
            JExpression initExpr = initials.get(i);

            // Duplicate the array for each element store
            output.addNoArgInstruction(DUP);

            // Code to push index for store
            new JLiteralInt(line, String.valueOf(i)).codegen(output);

            // Code to compute the initial value.
            initExpr.codegen(output);

            // Code to store the initial value in the array
        if (componentType == Type.INT) {
        output.addNoArgInstruction(IASTORE);
        } else if (componentType == Type.BOOLEAN) {
        output.addNoArgInstruction(BASTORE);
        } else if (componentType == Type.CHAR) {
        output.addNoArgInstruction(CASTORE);
        } else if (!componentType.isPrimitive()) {
        output.addNoArgInstruction(AASTORE);
        }
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.println("<JArrayInitializer>");
        if (initials != null) {
            for (JAST initial : initials) {
                p.indentRight();
                initial.writeToStdOut(p);
                p.indentLeft();
            }
        }
        p.println("</JArrayInitializer>");
    }
}
```