JLiteralInt.java

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

package jminusminus;

import static jminusminus.CLConstants.*;

/**
 * The AST node for an int literal.
 */

class JLiteralInt extends JExpression {

    /** String representation of the int. */
    private String text;

    /**
     * Construct an AST node for an int literal given its line number and string
     * representation.
     * 
     * @param line
     *            line in which the literal occurs in the source file.
     * @param text
     *            string representation of the literal.
     */

    public JLiteralInt(int line, String text) {
        super(line);
        this.text = text;
    }

    /**
     * Analyzing an int literal is trivial.
     * 
     * @param context
     *            context in which names are resolved (ignored here).
     * @return the analyzed (and possibly rewritten) AST subtree.
     */

    public JExpression analyze(Context context) {
        type = Type.INT;
        return this;
    }

    /**
     * Generating code for an int literal means generating code to push it onto
     * the stack.
     * 
     * @param output
     *            the code emitter (basically an abstraction for producing the
     *            .class file).
     */

    public void codegen(CLEmitter output) {
        int i = Integer.parseInt(text);
        switch (i) {
        case 0:
            output.addNoArgInstruction(ICONST_0);
            break;
        case 1:
            output.addNoArgInstruction(ICONST_1);
            break;
        case 2:
            output.addNoArgInstruction(ICONST_2);
            break;
        case 3:
            output.addNoArgInstruction(ICONST_3);
```

```
                break;
            case 4:
                output.addNoArgInstruction(ICONST_4);
                break;
            case 5:
                output.addNoArgInstruction(ICONST_5);
                break;
            default:
                if (i >= 6 && i <= 127) {
                    output.addOneArgInstruction(BIPUSH, i);
                } else if (i >= 128 && i <= 32767) {
                    output.addOneArgInstruction(SIPUSH, i);
                } else {
                    output.addLDCInstruction(i);
                }
            }
        }

        /**
         * @inheritDoc
         */

        public void writeToStdOut(PrettyPrinter p) {
            p.printf("<JLiteralInt line=\"%d\" type=\"%s\" " + "value=\"%s\"/>\n",
                    line(), ((type == null) ? "" : type.toString()), text);
        }

    }

```