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

package jminusminus;

import static jminusminus.CLConstants.*;

/**
 * The AST node for a char literal.
 */

class JLiteralChar extends JExpression {

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

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

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

    /**
     * Analyzing a char 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.CHAR;
        return this;
    }

    /**
     * Generating code for a char 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) {
        // Unescape the escaped escapes
        String s = Util.unescape(text);

        // The string representation is padded (by hand-written
        // and JavaCC scanner) with single quotes, so we extract
        // the char at 1
        char c = s.charAt(1);
        int i = (int) c;
        switch (i) {
        case 0:
            output.addNoArgInstruction(ICONST_0);
            break;
        case 1:
```

```
67              output.addNoArgInstruction(ICONST_1);
68              break;
69          case 2:
70              output.addNoArgInstruction(ICONST_2);
71              break;
72          case 3:
73              output.addNoArgInstruction(ICONST_3);
74              break;
75          case 4:
76              output.addNoArgInstruction(ICONST_4);
77              break;
78          case 5:
79              output.addNoArgInstruction(ICONST_5);
80              break;
81          default:
82              if (i >= 6 && i <= 127) {
83                  output.addOneArgInstruction(BIPUSH, i);
84              } else if (i >= 128 && i <= 32767) {
85                  output.addOneArgInstruction(SIPUSH, i);
86              } else {
87                  output.addLDCInstruction(i);
88              }
89          }
90      }
91
92      /**
93       * @inheritDoc
94       */
95
96      public void writePostOut(PrettyPrinter p) {
97          p.println("<JLiteralChar line=\"%d\" type=\"%s\" " + "value=\"%s\"/>\n",
98                  line(), ((type == null) ? "" : type.toString()), Util
99                          .escapeSpecialXMLChars(text));
100     }
101
102 }
103
```