```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 this(...) constructor.
 */

class JThisConstruction extends JExpression {

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

    /** Constructor representation of the constructor. */
    private Constructor constructor;

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

    /**
     * Whether this constructor is used properly, ie, as the first statement
     * within a constructor.
     */
    private boolean properUseOfConstructor = false;

    /**
     * Construct the AST node for a this(...) constructor given its line number
     * and arguments.
     *
     * @param line
     *            line in which the constructor occurs in the source file.
     * @param arguments
     *            the constructor's arguments.
     */

    protected JThisConstruction(int line, ArrayList<JExpression> arguments) {
        super(line);
        this.arguments = arguments;
    }

    /**
     * Used in JConstructorDeclaration to mark this(...) as being properly
     * placed, ie, as the first statement in its body.
     */

    public void markProperUseOfConstructor() {
        properUseOfConstructor = true;
    }

    /**
     * Analyzing a this constructor statement involves (1) setting the type, (2)
     * analyzing the actual arguments, (3) checking that this construction
     * statement is properly invoked (as the first statement in another
     * constructor), and (4) finding the appropriate Constructor
     *
     * @param context
     *            context in which names are resolved.
     * @return the analyzed (and possibly rewritten) AST subtree.
     */

    public JExpression analyze(Context context) {
        type = Type.VOID;

```

```
67          // Analyze the arguments, collecting
68          // their types (in Class form) as argTypes.
69          argTypes = new Type[arguments.size()];
70          for (int i = 0; i < arguments.size(); i++) {
71              arguments.set(i, (JExpression) arguments.get(i).analyze(context));
72              argTypes[i] = arguments.get(i).type();
73          }
74
75          if (!properUseOfConstructor) {
76              JAST.compilationUnit.reportSemanticError(line(), "this"
77                      + Type.argTypesAsString(argTypes)
78                      + " must be first statement in the constructor's body.");
79              return this;
80          }
81
82          // Get the constructor this(...) refers to.
83          constructor = ((JTypeDecl) context.classContext.definition())
84                  .thisType().constructorFor(argTypes);
85
86          if (constructor == null) {
87              JAST.compilationUnit.reportSemanticError(line(),
88                      "No such constructor: this"
89                              + Type.argTypesAsString(argTypes));
90
91          }
92          return this;
93      }
94
95      /**
96       * Code generation involves generating the code for loading the actual
97       * arguments onto the stack, and then for invoking this constructor.
98       *
99       * @param output
100      *             the code emitter (basically an abstraction for producing the
101      *             class file).
102      */
103
104     public void codegen(CLEmitter output) {
105         output.addNoArgInstruction(ALOAD_0); // this
106         for (JExpression argument : arguments) {
107             argument.codegen(output);
108         }
109         output.addMemberAccessInstruction(INVOKESPECIAL, constructor
110                 .declaringType().jvmName(), "<init>", constructor
111                 .toDescriptor());
112     }
113
114     /**
115      * @inheritDoc
116      */
117
118     public void writeToStdOut(PrettyPrinter p) {
119         p.printf("<JThisConstruction line=\"%d\"/>\n", line());
120         p.indentRight();
121         if (arguments != null) {
122             p.println("<Arguments>");
123             for (JExpression argument : arguments) {
124                 p.indentRight();
125                 p.println("<Argument>");
126                 p.indentRight();
127                 argument.writeToStdOut(p);
128                 p.indentLeft();
129                 p.println("</Argument>");
130                 p.indentLeft();
131             }
132             p.println("</Arguments>");
133         }
134         p.indentLeft();
135         p.println("</JThisConstruction>");
```

```
136     }
137
138 }
139
```