```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 super(...) constructor.
 */

class JSuperConstruction 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 an AST node for a super(...) 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 JSuperConstruction(int line, ArrayList<JExpression> arguments) {
        super(line);
        this.arguments = arguments;
    }

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

    public void markProperUseOfConstructor() {
        properUseOfConstructor = true;
    }

    /**
     * Analyzing a super constructor statement involves (1) setting the type,
     * (2) analyzing the actual arguments, and (3) checking that this
     * construction statement is properly invoked (as the first statement in
     * another 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;

```

```
        // 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();
        }

        if (!properUseOfConstructor) {
            JAST.compilationUnit.reportSemanticError(line(), "super"
                    + Type.argTypesAsString(argTypes)
                    + " must be first statement in the constructor's body.");
            return this;
        }

        // Get the Constructor super(...) refers to
        Type superClass = ((JTypeDecl) context.classContext.definition())
                .thisType().superClass();
        if (superClass == null) {
            JAST.compilationUnit.reportSemanticError(line,
                    ((JTypeDecl) context.classContext.definition()).thisType()
                            + " has no super class.");
        }
        constructor = superClass.constructorFor(argTypes);

        if (constructor == null) {
            JAST.compilationUnit.reportSemanticError(line(),
                    "No such constructor: super"
                            + Type.argTypesAsString(argTypes));
        }
        return this;
    }

    /**
     * Code generation involves generating code to load the actual arguments
     * onto the stack, and then the code for invoking the constructor.
     *
     * @param output
     *            the code emitter (basically an abstraction for producing the
     *            .class file).
     */

    public void codegen(CLEmitter output) {
        output.addNoArgInstruction(ALOAD_0); // this
        for (JExpression argument : arguments) {
            argument.codegen(output);
        }
        output.addMemberAccessInstruction(INVOKESPECIAL, constructor
                .declaringType().jvmName(), "<init>", constructor
                .toDescriptor());
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("<JSuperConstruction line=\"%d\"/>\n", line());
        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>");
```

```
136                 p.indentLeft();
137             }
138             p.println("</Arguments>");
139         }
140         p.indentLeft();
141         p.println("</JSuperConstruction>");
142     }
143 
144 }
145 
```