

JMessageExpression.java

```
1  // Copyright 2013 Bill Campbell, Swami Iyer and Bahar Akbal-Delibas
2
3  package jminusminus;
4
5  import java.util.ArrayList;
6
7  import static jminusminus.CLConstants.*;
8
9  /**
10   * The AST node for a message expression that has a target, optionally an
11   * ambiguous part, a message name, and zero or more actual arguments.
12   */
13
14  class JMessageExpression extends JExpression {
15
16      /** The target expression. */
17      private JExpression target;
18
19      /** The ambiguous part that is reclassified in analyze(). */
20      private AmbiguousName ambiguousPart;
21
22      /** The message name. */
23      private String messageName;
24
25      /** Message arguments. */
26      private ArrayList<JExpression> arguments;
27
28      /** Types of arguments. */
29      private Type[] argTypes;
30
31      /** The Method representing this message. */
32      private Method method;
33
34      /**
35       * Construct an AST node for a message expression without an ambiguous part.
36       *
37       * @param line
38       *     line in which the expression occurs in the source file.
39       * @param target
40       *     the target expression.
41       * @param messageName
42       *     the message name.
43       * @param arguments
44       *     the ambiguousPart arguments.
45       */
46
47      protected JMessageExpression(int line, JExpression target,
48          String messageName, ArrayList<JExpression> arguments) {
49          this(line, target, null, messageName, arguments);
50      }
51
52      /**
53       * Construct an AST node for a message expression having an ambiguous part.
54       *
55       * @param line
56       *     line in which the expression occurs in the source file.
57       * @param target
58       *     the target expression.
59       * @param ambiguousPart
60       *     the ambiguous part.
61       * @param messageName
62       *     the message name.
63       * @param arguments
64       *     the arguments.
65       */
66  }
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

67     protected JMessageExpression(int line, JExpression target,
68         AmbiguousName ambiguousPart, String messageName,
69         ArrayList<JExpression> arguments) {
70         super(line);
71         this.target = target;
72         this.ambiguousPart = ambiguousPart;
73         this.messageName = messageName;
74         this.arguments = arguments;
75     }
76
77     /**
78      * Analysis of a message expression involves: (1) reclassifying any
79      * ambiguous part, (2) analyzing and computing the types for the actual
80      * arguments, (3) determining the type we are currently in (for checking
81      * access), (4) analyzing the target and determining its type, (5) finding
82      * the appropriate Method, (6) checking accessibility, and (7) determining
83      * the result type.
84      *
85      * @param context
86      *      context in which names are resolved.
87      * @return the analyzed (and possibly rewritten) AST subtree.
88      */
89
90     public JExpression analyze(Context context) {
91         // Reclassify the ambiguous part
92         if (ambiguousPart != null) {
93             JExpression expr = ambiguousPart.reclassify(context);
94             if (expr != null) {
95                 if (target == null) {
96                     target = expr;
97                 } else {
98                     // Can't even happen syntactically
99                     JAST.compilationUnit.reportSemanticError(line(),
100                         "Badly formed suffix");
101                 }
102             }
103         }
104
105         // Then analyze the arguments, collect and
106         // their types (in Class form) as argTypes
107         argTypes = new Type[arguments.size()];
108         for (int i = 0; i < arguments.size(); i++) {
109             arguments.set(i, (JExpression) arguments.get(i).analyze(context));
110             argTypes[i] = arguments.get(i).type();
111         }
112
113         // Where are we now? (For access)
114         Type thisType = ((JTypeDecl) context.classContext.definition())
115             .thisType();
116
117         // Then analyze the target
118         if (target == null) {
119             // Implied this (or, implied type for statics)
120             if (!context.methodContext().isStatic()) {
121                 target = new JThis(line()).analyze(context);
122             } else {
123                 target = new JVariable(line(), context.definingType()
124                     .toString()).analyze(context);
125             }
126         } else {
127             target = (JExpression) target.analyze(context);
128             if (target.type().isPrimitive()) {
129                 JAST.compilationUnit.reportSemanticError(line(),
130                     "cannot invoke a message on a primitive type:"
131                     + target.type());
132             }
133         }
134
135         // Find appropriate Method for this message expression

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

136     method = target.type().methodFor(messageName, argTypes);
137     if (method == null) {
138         JAST.compilationUnit.reportSemanticError(line(),
139             "Cannot find method for: "
140             + Type.signatureFor(messageName, argTypes));
141         type = Type.ANY;
142     } else {
143         context.definingType().checkAccess(line, (Member) method);
144         type = method.returnType();
145
146         // Non-static method cannot be referenced from a static context.
147         if (!method.isStatic()) {
148             if (target instanceof JVariable
149                 && ((JVariable) target).iDefn() instanceof TypeNameDefn)
150             {
151                 JAST.compilationUnit
152                     .reportSemanticError(
153                         line(),
154                         "Non-static method "
155                         + Type.signatureFor(messageName,
156                             argTypes)
157                         + "cannot be referenced from a static
context");
158             }
159         }
160         return this;
161     }
162 }
163
164 /**
165  * Code generation for a message expression involves generating code for
166  * loading the target onto the stack, generating code to load the actual
167  * arguments onto the stack, and then invoking the named Method. Notice that
168  * if this is a statement expression, as marked by a parent
169  * JStatementExpression, then we also generate code for popping the stacked
170  * value for any non-void invocation.
171  *
172  * @param output the code emitter (basically an abstraction for producing the
173  *                .class file).
174  */
175
176 public void codegen(CLEmitter output) {
177     if (!method.isStatic()) {
178         target.codegen(output);
179     }
180     for (JExpression argument : arguments) {
181         argument.codegen(output);
182     }
183     int mnemonic = method.isStatic() ? INVOKESTATIC : target.type()
184         .isInterface() ? INVOKEINTERFACE : INVOKEVIRTUAL;
185     output.addMemberAccessInstruction(mnemonic, target.type().jvmName(),
186         messageName, method.toDescriptor());
187     if (isStatementExpression && type != Type.VOID) {
188         // Pop any value left on the stack
189         output.addNoArgInstruction(POP);
190     }
191 }
192
193 /**
194  * The semantics of j-- require that we implement short-circuiting branching
195  * in implementing message expressions.
196  *
197  * @param output the code emitter (basically an abstraction for producing the
198  *                .class file).
199  * @param targetLabel the label to which we should branch.
200  * @param onTrue
201  */

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

203     *           do we branch on true?
204     */
205
206 public void codegen(CLEmitter output, String targetLabel, boolean onTrue) {
207     // Push the value
208     codegen(output);
209
210     if (onTrue) {
211         // Branch on true
212         output.addBranchInstruction(IFNE, targetLabel);
213     } else {
214         // Branch on false
215         output.addBranchInstruction(IFEQ, targetLabel);
216     }
217 }
218
219 /**
220  * @inheritDoc
221  */
222
223 public void writeToStdOut(PrettyPrinter p) {
224     p.printf("<JMessageExpression line=\"%d\" name=\"%s\">\n", line(),
225             messageName);
226     p.indentRight();
227     if (target != null) {
228         p.println("<Target>");
229         p.indentRight();
230         target.writeToStdOut(p);
231         p.indentLeft();
232         p.println("</Target>");
233     }
234     if (arguments != null) {
235         p.println("<Arguments>");
236         for (JExpression argument : arguments) {
237             p.indentRight();
238             p.println("<Argument>");
239             p.indentRight();
240             argument.writeToStdOut(p);
241             p.indentLeft();
242             p.println("</Argument>");
243             p.indentLeft();
244         }
245         p.println("</Arguments>");
246     }
247     p.indentLeft();
248     p.println("</JMessageExpression>");
249 }
250
251 }
252

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder