

JVariable.java

```
1  // Copyright 2013 Bill Campbell, Swami Iyer and Bahar Akbal-Delibas
2
3  package jminusminus;
4
5  import static jminusminus.CLConstants.*;
6
7  /**
8   * The AST node for an identifier used as a primary expression.
9   */
10
11 class JVariable extends JExpression implements JLhs {
12
13     /** The variable's name. */
14     private String name;
15
16     /** The variable's definition. */
17     private IDefn iDefn;
18
19     /** Was analyzeLhs() done? */
20     private boolean analyzeLhs;
21
22     /**
23      * Construct the AST node for a variable given its line number and name.
24      *
25      * @param line
26      *      line in which the variable occurs in the source file.
27      * @param name
28      *      the name.
29      */
30
31     public JVariable(int line, String name) {
32         super(line);
33         this.name = name;
34     }
35
36     /**
37      * Return the identifier name.
38      *
39      * @return the identifier name.
40      */
41
42     public String name() {
43         return name;
44     }
45
46     /**
47      * Return the identifier's definition.
48      *
49      * @return the identifier's definition.
50      */
51
52     public IDefn iDefn() {
53         return iDefn;
54     }
55
56     /**
57      * Analyzing identifiers involves resolving them in the context. Identifiers
58      * denoting fields (with implicit targets) are rewritten as explicit field
59      * selection operations.
60      *
61      * @param context
62      *      context in which names are resolved.
63      * @return the analyzed (and possibly rewritten) AST subtree.
64      */
65
66     public JExpression analyze(Context context) {
```

```

67         iDefn = context.lookup(name);
68         if (iDefn == null) {
69             // Not a local, but is it a field?
70             Type definingType = context.definingType();
71             Field field = definingType.fieldFor(name);
72             if (field == null) {
73                 type = Type.ANY;
74                 JAST.compilationUnit.reportSemanticError(line,
75                     "Cannot find name: " + name);
76             } else {
77                 // Rewrite a variable denoting a field as an
78                 // explicit field selection
79                 type = field.type();
80                 JExpression newTree = new JFieldSelection(line(), field
81                     .isStatic()
82                     || (context.methodContext() != null && context
83                         .methodContext().isStatic()) ? new JVariable(
84                     line(), definingType.toString()) : new JThis(line(),
85                     name);
86                 return (JExpression) newTree.analyze(context);
87             }
88         } else {
89             if (!analyzeLhs && iDefn instanceof LocalVariableDefn
90                 && !((LocalVariableDefn) iDefn).isInitialized()) {
91                 JAST.compilationUnit.reportSemanticError(line, "Variable "
92                     + name + " might not have been initialized");
93             }
94             type = iDefn.type();
95         }
96     }
97 }

98
99 /**
100  * Analyze the identifier as used on the lhs of an assignment.
101  *
102  * @param context
103  *     context in which names are resolved.
104  * @return the analyzed (and possibly rewritten) AST subtree.
105  */
106
107 public JExpression analyzeLhs(Context context) {
108     analyzeLhs = true;
109     JExpression newTree = analyze(context);
110     if (newTree instanceof JVariable) {
111         // Could (now) be a JFieldSelection, but if it's
112         // (still) a JVariable
113         if (iDefn != null && !(iDefn instanceof LocalVariableDefn)) {
114             JAST.compilationUnit.reportSemanticError(line(), name
115                 + " is a bad lhs to a =");
116         }
117     }
118     return newTree;
119 }
120
121 /**
122  * Generate code to load value of variable on stack.
123  *
124  * @param output
125  *     the code emitter (basically an abstraction for producing the
126  *     .class file).
127  */
128
129 public void codegen(CLEmitter output) {
130     if (iDefn instanceof LocalVariableDefn) {
131         int offset = ((LocalVariableDefn) iDefn).offset();
132         if (type.isReference()) {
133             switch (offset) {
134                 case 0:
135                     output.addNoArgInstruction(ALOAD_0);

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

136         break;
137     case 1:
138         output.addNoArgInstruction(ALOAD_1);
139         break;
140     case 2:
141         output.addNoArgInstruction(ALOAD_2);
142         break;
143     case 3:
144         output.addNoArgInstruction(ALOAD_3);
145         break;
146     default:
147         output.addOneArgInstruction(ALOAD, offset);
148         break;
149     }
150 } else {
151     // Primitive types
152     if (type == Type.INT || type == Type.BOOLEAN
153         || type == Type.CHAR) {
154         switch (offset) {
155             case 0:
156                 output.addNoArgInstruction(ILOAD_0);
157                 break;
158             case 1:
159                 output.addNoArgInstruction(ILOAD_1);
160                 break;
161             case 2:
162                 output.addNoArgInstruction(ILOAD_2);
163                 break;
164             case 3:
165                 output.addNoArgInstruction(ILOAD_3);
166                 break;
167             default:
168                 output.addOneArgInstruction(ILOAD, offset);
169                 break;
170         }
171     }
172 }
173 }
174 }
175
176 /**
177  * The semantics of j-- require that we implement short-circuiting branching
178  * in implementing the identifier expression.
179  *
180  * @param output
181  *     the code emitter (basically an abstraction for producing the
182  *     .class file).
183  * @param targetLabel
184  *     the label to which we should branch.
185  * @param onTrue
186  *     do we branch on true?
187  */
188
189 public void codegen(CLEmitter output, String targetLabel, boolean onTrue) {
190     if (iDefn instanceof LocalVariableDefn) {
191         // Push the value
192         codegen(output);
193
194         if (onTrue) {
195             // Branch on true
196             output.addBranchInstruction(IFNE, targetLabel);
197         } else {
198             // Branch on false
199             output.addBranchInstruction(IFEQ, targetLabel);
200         }
201     }
202 }
203
204 /**

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

205 * Generate the code required for setting up an Lvalue, eg for use in an
206 * assignment. Here, this requires nothing; all information is in the the
207 * store instruction.
208 *
209 * @param output
210 *     the emitter (an abstraction of the class file).
211 */
212
213 public void codegenLoadLhsLvalue(CLEmitter output) {
214     // Nothing goes here.
215 }
216
217 /**
218 * Generate the code required for loading an Rvalue for this variable, eg
219 * for use in a +=. Here, this requires loading the Rvalue for the variable
220 *
221 * @param output
222 *     the emitter (an abstraction of the class file).
223 */
224
225 public void codegenLoadLhsRvalue(CLEmitter output) {
226     codegen(output);
227 }
228
229 /**
230 * Generate the code required for duplicating the Rvalue that is on the
231 * stack because it is to be used in a surrounding expression, as in a[i] =
232 * x = <expr> or x = y--. Here this means simply duplicating the value on
233 * the stack.
234 *
235 * @param output
236 *     the code emitter (basically an abstraction for producing the
237 *     .class file).
238 */
239
240 public void codegenDuplicateRvalue(CLEmitter output) {
241     if (iDefn instanceof LocalVariableDefn) {
242         // It's copied atop the stack.
243         output.addNoArgInstruction(DUP);
244     }
245 }
246
247 /**
248 * Generate the code required for doing the actual assignment. Here, this
249 * requires storing what's on the stack at the appropriate offset.
250 *
251 * @param output
252 *     the code emitter (basically an abstraction for producing the
253 *     .class file).
254 */
255
256 public void codegenStore(CLEmitter output) {
257     if (iDefn instanceof LocalVariableDefn) {
258         int offset = ((LocalVariableDefn) iDefn).offset();
259         if (type.isReference()) {
260             switch (offset) {
261                 case 0:
262                     output.addNoArgInstruction(ASTORE_0);
263                     break;
264                 case 1:
265                     output.addNoArgInstruction(ASTORE_1);
266                     break;
267                 case 2:
268                     output.addNoArgInstruction(ASTORE_2);
269                     break;
270                 case 3:
271                     output.addNoArgInstruction(ASTORE_3);
272                     break;
273                 default:

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

274         output.addOneArgInstruction(ASTORE, offset);
275         break;
276     }
277 } else {
278     // Primitive types
279     if (type == Type.INT || type == Type.BOOLEAN
280         || type == Type.CHAR) {
281         switch (offset) {
282             case 0:
283                 output.addNoArgInstruction(ISTORE_0);
284                 break;
285             case 1:
286                 output.addNoArgInstruction(ISTORE_1);
287                 break;
288             case 2:
289                 output.addNoArgInstruction(ISTORE_2);
290                 break;
291             case 3:
292                 output.addNoArgInstruction(ISTORE_3);
293                 break;
294             default:
295                 output.addOneArgInstruction(ISTORE, offset);
296                 break;
297         }
298     }
299 }
300 }
301 }
302
303 /**
304  * @inheritDoc
305  */
306
307 public void writeToStdOut(PrettyPrinter p) {
308     p.println("<Variable name=\"" + name + "\" />");
309 }
310
311 }
312

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder