```
1   // Copyright 2011 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 instanceof expression, having two
9    * arguments: an expression and a reference type.
10   */
11
12  class JInstanceOfOp
13      extends JExpression {
14
15      /** The expression denoting the value to be tested. */
16      private JExpression expr;
17
18      /** The reference type we are testing for. */
19      private Type typeSpec;
20
21      /**
22       * Construct an AST node for an instanceof expression given
23       * its line number, the relational expression and reference
24       * type.
25       *
26       * @param line
27       *                the line in which the instanceof expression
28       *                occurs in the source file.
29       * @param expr
30       *                the expression denoting the value to be
31       *                tested.
32       * @param typeSpec
33       *                the reference type we are testing for.
34       */
35
36      public JInstanceOfOp(int line, JExpression expr, Type typeSpec) {
37          super(line);
38          this.expr = expr;
39          this.typeSpec = typeSpec;
40      }
41
42      /**
43       * Analysis of an instanceof operation requires analyzing the
44       * expression to be tested, resolving the type was are
45       * testing for, and determining if the test is legal, or if
46       * the answer can be determined at compile time.
47       *
48       * @param context
49       *                context in which names are resolved.
50       * @return the analyzed (and possibly rewritten) AST subtree.
51       */
52
53      public JInstanceOfOp analyze(Context context) {
54          expr = (JExpression) expr.analyze(context);
55          typeSpec = typeSpec.resolve(context);
56          if (!typeSpec.isReference()) {
57              JAST.compilationUnit.reportSemanticError(line(),
58                  "Type argument to instanceof "
59                      + "operator must be a reference type");
60          } else if (!(expr.type() == Type.NULLTYPE
61              || expr.type() == Type.ANY || expr.type().isReference())) {
62              JAST.compilationUnit.reportSemanticError(line(),
63                  "operand to instanceof "
64                      + "operator must be a reference type");
65          } else if (expr.type().isReference()
66              && !typeSpec.isJavaAssignableFrom(expr.type())) {
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
67              JAST.compilationUnit.reportSemanticError(line(),
68                  "It is impossible for the expression "
69                      + "to be an instance of this type");
70          }
71          type = Type.BOOLEAN;
72          return this;
73      }
74
75      /**
76       * Generate code for the type test.
77       *
78       * @param output
79       *                the code emitter (basically an abstraction
80       *                for producing the .class file).
81       */
82
83      public void codegen(CLEmitter output) {
84          expr.codegen(output);
85          output.addReferenceInstruction(INSTANCEOF, typeSpec
86              .toDescriptor());
87      }
88
89      /**
90       * Short-circuiting branching for instanceof.
91       *
92       * @param output
93       *        code emitter.
94       * @param targetLabel
95       *            the label to which we should branch.
96       * @param onTrue
97       *            do we branch on true?
98       */
99
100     public void codegen(CLEmitter output, String targetLabel, boolean onTrue) {
101     codegen(output);
102     if (onTrue) {
103         // Branch on true
104         output.addBranchInstruction(IFNE, targetLabel);
105     } else {
106         // Branch on false
107         output.addBranchInstruction(IFEQ, targetLabel);
108     }
109     }
110
111     /**
112      * @inheritDoc
113      */
114
115     public void writeToStdOut(PrettyPrinter p) {
116         p.printf("<JInstanceOfOp line=\"%d\" type=\"%s\">\n", line(),
117             ((type == null) ? "" : type.toString()));
118         p.indentRight();
119         p.printf("<RelationalExpression>\n");
120         p.indentRight();
121         expr.writeToStdOut(p);
122         p.indentLeft();
123         p.printf("</RelationalExpression>\n");
124         p.printf("<ReferenceType value=\"%s\"/>\n",
125             ((typeSpec == null) ? "" : typeSpec.toString()));
126         p.indentLeft();
127         p.printf("</JInstanceOfOp>\n");
128     }
129 }
130
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder