```
1   // Copyright 2013 Bill Campbell, Swami Iyer and Bahar Akbal-Delibas
2
3   package jminusminus;
4
5   import java.util.StringTokenizer;
6
7   /**
8    * Ambiguous names are meant to deal with snippets like
9    *
10   * <pre>
11   *   x.y.z
12   *   a.b.c()
13   * </pre>
14   *
15   * Clearly, z is a field and c is a method. But what about x.y and a.b ? x could
16   * be a package name and y a type, making for a (static) class field selection.
17   * Or, x could be a local variable and y an instance field. The parser cannot
18   * know how to parse these.
19   *
20   * Disambiguating the ambiguity must wait until analysis time. The parser can,
21   * with x.y.z, treat the .z as a field selection, but constructs an
22   * AmbiguousName object encapsulating the x.y . And it can, with a.b.c(), treat
23   * the .c() as a message expression, but constructs an AbiguousName object
24   * encapsulating a.b.
25   *
26   * reclassify() is called upon in JFieldSelection.analyze() and
27   * JMessageExpression.analyze() to reclassify the components and construct the
28   * proper ast, following the rules for names in the Java Language Specification
29   * (Third Edition), section 6.5.2. In j--, both x.y and a.b are clearly
30   * expressions in these contexts. If inner types were to be introduced, their
31   * meaning and their reclassification would necessarily be more complicated.
32   */
33
34  class AmbiguousName {
35
36      /**
37       * Line in which the ambiguous name occurs in the source file.
38       */
39      private int line;
40
41      /** The ambiguous part, eg x.y */
42      private String name;
43
44      /**
45       * Construct an encapsulation of the ambiguous portion of a snippet like
46       * x.y.z.
47       *
48       * @param line
49       *             line in which the ambiguous name occurs in the source file.
50       * @param name
51       *             the ambiguous part.
52       */
53
54      public AmbiguousName(int line, String name) {
55          this.line = line;
56          this.name = name;
57      }
58
59      /**
60       * Reclassify the name according to the rules in the Java Language
61       * Specification.
62       *
63       * @param context
64       *             context in which we look up the component names.
65       * @return the properly parsed AST.
66       */
```

```
67
68      public JExpression reclassify(Context context) {
69          // Easier because we require all types to be imported.
70          JExpression result = null;
71          StringTokenizer st = new StringTokenizer(name, ".");
72
73          // Firstly, find a variable or Type.
74          String newName = st.nextToken();
75          IDefn iDefn = null;
76
77          do {
78              iDefn = context.lookup(newName);
79              if (iDefn != null) {
80                  result = new JVariable(line, newName);
81                  break;
82              } else if (!st.hasMoreTokens()) {
83                  // Nothing found. :(
84                  JAST.compilationUnit.reportSemanticError(line,
85                          "Cannot find name " + newName);
86                  return null;
87              } else {
88                  newName += "." + st.nextToken();
89              }
90          } while (true);
91
92          // For now we can assume everything else is fields.
93          while (st.hasMoreTokens()) {
94              result = new JFieldSelection(line, result, st.nextToken());
95          }
96          return result;
97      }
98
99  }
100
```