

```

1  /**
2  *
3  * Class Types has static methods for type equivalencing and other forms of
4  * type interrogation. Note that this class must be compiled with a
5  * language-specific sym.java file. It therefore must be copied in source form
6  * and compiled together with a specific parser, since it cannot be compiled in
7  * with a the stand-alone a4-support.jar files.
8  *
9  */
10
11 public class Types {
12
13     /**
14      * Return true if t1 and t2 are structurally equivalent.
15      */
16     public static boolean equiv(TypeNode t1, TypeNode t2) {
17         return
18             samePrimitiveTypes(t1, t2)
19             // ||
20             /* perform parallel recursive descent on type structures */;
21     }
22
23     /**
24      * Return true if t1 and t2 are name equivalent.
25      */
26     public static boolean equivName(TypeNode t1, TypeNode t2) {
27         return
28             samePrimitiveTypes(t1, t2)
29             ||
30             sameIdentTypes(t1, t2);
31     }
32
33     /**
34      *
35      * Return true if t1 and t2 are the same primitive types.
36      *
37      */
38     public static boolean samePrimitiveTypes(TypeNode t1, TypeNode t2) {
39         return
40             isInt(t1) && isInt(t2)
41             ||
42             isFloat(t1) && isFloat(t2)
43             ||
44             isString(t1) && isString(t2)
45             ||
46             isBool(t1) && isBool(t2);
47     }
48
49     /**
50      * Return true if t is a numeri type, i.e., INT or FLOAT.
51      */
52     public static boolean isNumeric(TypeNode t) {
53         return isInt(t) || isFloat(t);
54     }
55
56     /**
57
58     * Return true if t1 and t2 are the same identifier type, i.e., the have
59     * the same type name.
60     */
61     public static boolean sameIdentTypes(TypeNode t1, TypeNode t2) {
62         return
63             (t1.id == sym.IDENT)
64             &&
65             (t2.id == sym.IDENT)
66             &&
67             (((LeafNode) t1.child1).value).equals(
68                 ((LeafNode) t2.child1).value);
69     }
70
71     /**
72      * Return true if the given type is an atomic integer type. This is the
73      * case if the TypeNode id = INT or if the id is IDENT and its string ident
74      * value is "integer". This supports languages in which the integer type
75      * is designated by a keyword, as well as languages where it is designated
76      * by a pre-defined identifier named "integer".
77      */
78     public static boolean isInt(TypeNode t) {
79         return
80             (t.id == sym.INT)
81             ||
82             (t.id == sym.IDENT) &&
83                 (((LeafNode) t.child1).value).equals("integer");
84     }
85
86     /**
87      * Return true if the given type is an atomic floating point type. This is
88      * the case if the TypeNode id = FLOAT or if the id is IDENT and its string
89      * ident value is "real". This supports languages in which the integer
90      * type is designated by a keyword, as well as languages where it is
91      * designated by a pre-defined identifier named "real".
92      */
93     public static boolean isFloat(TypeNode t) {
94         return
95             (t.id == sym.FLOAT) ||
96             (t.id == sym.IDENT) &&
97                 (((LeafNode) t.child1).value).equals("real");
98     }
99
100     /**
101      * Return true if the given type is an atomic floating point type. This is
102      * the case if the TypeNode id = STRING or if the id is IDENT and its
103      * string ident value is "string". This supports languages in which the
104      * integer type is designated by a keyword, as well as languages where it
105      * is designated by a pre-defined identifier named "string".
106      */
107     public static boolean isString(TypeNode t) {
108         return
109             (t.id == sym.STRING) ||
110             (t.id == sym.IDENT) &&
111                 (((LeafNode) t.child1).value).equals("string");
112     }

```

```
113     /**
114     * Return true if the given type is an atomic floating point type. This is
115     * the case if the TypeNode id = BOOLEAN or if the id is IDENT and its
116     * string ident value is "boolean". This supports languages in which the
117     * integer type is designated by a keyword, as well as languages where it
118     * is designated by a pre-defined identifier named "boolean".
119     */
120     public static boolean isBool(TypeNode t) {
121         return
122             (t.id == sym.BOOLEAN) ||
123             (t.id == sym.IDENT) &&
124             (((LeafNode) t.child1).value).equals("boolean");
125     }
126
127     /**
128     * Return true if the given type is an atomic floating point type. This is
129     * the case if the TypeNode id = VOID or if the id is IDENT and its string
130     * ident value is "void". This supports languages in which the integer
131     * type is designated by a keyword, as well as languages where it is
132     * designated by a pre-defined identifier named "void".
133     */
134     public static boolean isVoid(TypeNode t) {
135         return
136             (t.id == sym.VOID) ||
137             (t.id == sym.IDENT) &&
138             (((LeafNode) t.child1).value).equals("void");
139     }
140
141 }
```