```java
/****
 *
 * Class Types has static methods for type equivalencing and other forms of
 * type interrogation.  Note that this class must be compiled with a
 * language-specific sym.java file.  It therefore must be copied in source form
 * and compiled together with a specific parser, since it cannot be compiled in
 * with a the stand-alone a4-support.jar files.
 *
 */

public class Types {

    /**
     * Return true if t1 and t2 are structurally equivalent.
     */
    public static boolean equiv(TypeNode t1, TypeNode t2) {
        return
            samePrimitiveTypes(t1, t2)
                // ||
            /* perform parallel recursive descent on type structures */;
    }

    /**
     * Return true if t1 and t2 are name equivalent.
     */
    public static boolean equivName(TypeNode t1, TypeNode t2) {
        return
            samePrimitiveTypes(t1, t2)
                ||
            sameIdentTypes(t1, t2);
    }

    /****
     *
     * Return true if t1 and t2 are the same primitive types.
     *
     */
    public static boolean samePrimitiveTypes(TypeNode t1, TypeNode t2) {
        return
            isInt(t1) && isInt(t2)
                ||
            isFloat(t1) && isFloat(t2)
                ||
            isString(t1) && isString(t2)
                ||
            isBool(t1) && isBool(t2);
    }

    /**
     * Return true if t is a numberi type, i.e., INT or FLOAT.
     */
    public static boolean isNumeric(TypeNode t) {
        return isInt(t) || isFloat(t);
    }

    /**
     * Return true if t1 and t2 are the same identifier type, i.e., the have
     * the same type name.
     */
    public static boolean sameIdentTypes(TypeNode t1, TypeNode t2) {
        return
            (t1.id == sym.IDENT)
                &&
            (t2.id == sym.IDENT)
                &&
            (((LeafNode) t1.child1).value).equals(
                ((LeafNode) t2.child1).value);
    }

    /**
     * Return true if the given type is an atomic integer type.  This is the
     * case if the TypeNode id = INT or if the id is IDENT and its string ident
     * value is "integer".  This supports languages in which the integer type
     * is designated by a keyword, as well as languages where it is designated
     * by a pre-defined identifier named "integer".
     */
    public  static boolean isInt(TypeNode t) {
        return
            (t.id == sym.INT)
                ||
            (t.id == sym.IDENT) &&
                (((LeafNode) t.child1).value).equals("integer");
    }

    /**
     * Return true if the given type is an atomic floating point type.  This is
     * the case if the TypeNode id = FLOAT or if the id is IDENT and its string
     * ident value is "real".  This supports languages in which the integer
     * type is designated by a keyword, as well as languages where it is
     * designated by a pre-defined identifier named "real".
     */
    public static boolean isFloat(TypeNode t) {
        return
            (t.id == sym.FLOAT) ||
            (t.id == sym.IDENT) &&
                (((LeafNode) t.child1).value).equals("real");
    }

    /**
     * Return true if the given type is an atomic floating point type.  This is
     * the case if the TypeNode id = STRING or if the id is IDENT and its
     * string ident value is "string".  This supports languages in which the
     * integer type is designated by a keyword, as well as languages where it
     * is designated by a pre-defined identifier named "string".
     */
    public static boolean isString(TypeNode t) {
        return
            (t.id == sym.STRING) ||
            (t.id == sym.IDENT) &&
                (((LeafNode) t.child1).value).equals("string");
    }

```

```
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     /** The int type */
142     public static TypeNode IntType = new TypeNode(sym.INT);
143 
144     /** The float type */
145     public static TypeNode FloatType = new TypeNode(sym.FLOAT);
146 
147     /** The string type */
148     public static TypeNode StringType = new TypeNode(sym.STRING);
149 
150     /** The bool type */
151     public static TypeNode BoolType = new TypeNode(sym.BOOLEAN);
152 
153     /** The void type */
154     public static TypeNode VoidType = new TypeNode(sym.VOID);
155 
156 }
```