```java
/****
 *
 * TypeNode is a specialized extension of TreeNode4 intended for use in tree
 * evaluation contexts where the node is known to be a type.  The point of this
 * is to allow users of a TypeNode value to assume specific properties about
 * the node, without having to cast more generic TreeNodes in various ways.
 *                                                                          <p>
 * One specific property of a TypeNode is that its ID should be one of a fixed
 * set of values that are legal for identifying types.  These ID values can
 * vary among different languages, but should be limited in scope, and have a
 * specific meaning in the context of a TypeNode, even if they have another
 * meaning in the context of some other type of TreeNode.
 *                                                                          <p>
 * Another known property of a TypeNode is that it has four TreeNode children,
 * zero or more of which can be used to hold data for different types of node.
 * For example, built-in atomic types typically use none of the children,
 * relying on the ID to uniquely identify the type.  As another example, a
 * composite array type will typically use two children -- one for the base
 * type of the array, the other for the dimensions.
 *
 * A final specialized component of TypeNode is a data field of type
 * SymbolTable.  This is used for types that need a symbol table reference,
 * such as struct, record, and class types.
 */

public class TypeNode extends TreeNode4 {

    /**
     * Construct this with the given id and null children.
     */
    public TypeNode(int id) {
        super(id, null, null, null, null);
    }

    /**
     * Construct this with the given id and given single child.
     */
    public TypeNode(int id, TreeNode child1) {
        super(id, child1, null, null, null);
    }

    /**
     * Construct this with the given id and given two children.
     */
    public TypeNode(int id, TreeNode child1, TreeNode child2) {
        super(id, child1, child2, null, null);
    }

    /**
     * Construct this with the given id and given three children.
     */
    public TypeNode(int id, TreeNode child1, TreeNode child2,
            TreeNode child3) {
        super(id, child1, child2, child3, null);
    }

    /**
     * Construct this with the given id and given four children.
     */
    public TypeNode(int id, TreeNode child1, TreeNode child2,
            TreeNode child3, TreeNode child4) {
        super(id, child1, child2, child3, child4);
    }

    /**
     * A la the other constructor, but with line and column numbers.
     */
    public TypeNode(int id, int line, int column) {
        super(id, null, null, null, null, line, column);
    }

    /**
     * A la the other constructor, but with line and column numbers.
     */
    public TypeNode(int id, TreeNode child, int line, int column) {
        super(id, child, null, null, null, line, column);
    }

    /**
     * A la the other constructor, but with line and column numbers.
     */
    public TypeNode(int id, TreeNode child1, TreeNode child2, int line,
            int column) {
        super(id, child1, child2, null, null, line, column);
    }

    /**
     * A la the other constructor, but with line and column numbers.
     */
    public TypeNode(int id, TreeNode child1, TreeNode child2,
            TreeNode child3, int line, int column) {
        super(id,  child1, child2, child3, null, line, column);
    }

    /**
     * A la the other constructor, but with line and column numbers.
     */
    public TypeNode(int id, TreeNode child1, TreeNode child2,
            TreeNode child3, TreeNode child4, int line, int column) {
        super(id, child1, child2, child3, child4, line, column);
    }

    /**
     * Return the String representation of this subtree, which is the String
     * value of its ID, followed on the next zero to four indented lines by the
     * recursive toString of its four children.  Null children are not printed
     * at all.  See the documentation for <a href= "TreeNode.html#toString()">
     * TreeNode.toString() </a> for a general description the way trees are
     * represented as strings.
     */
    public String toString(int level) {
        String indent = "";
        for (int i = 0; i < level; i++) {
            indent += "  ";
        }
        return symPrint(id) + toStringLineAndColumn(" ") +
            (child1 == null ? "" : ("\n" + indent + "  " +
                child1.toString(level+1))) +
            (child2 == null ? "" : ("\n" + indent + "  " +
                child2.toString(level+1))) +
            (child3 == null ? "" : ("\n" + indent + "  " +
                child3.toString(level+1))) +
            (child4 == null ? "" : ("\n" + indent + "  " +
                child4.toString(level+1)));
    }

    /** Reference to a symbol table, for struct, record, and class types. */
    public SymbolTable symtab;

}
```