```java
import components.naturalnumber.NaturalNumber;

/**
 * CSE 2221 Project #7. Program to evaluate XMLTree expressions of {@code int}.
 *
 * @author Faye Leigh
 *
 */
public final class XMLTreeNNExpressionEvaluator {

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private XMLTreeNNExpressionEvaluator() {
    }

    /**
     * Evaluate the given expression.
     *
     * @param exp
     *            the {@code XMLTree} representing the expression
     * @return the value of the expression
     * @requires <pre>
     * [exp is a subtree of a well-formed XML arithmetic expression],
     * [the label of the root of exp is not "expression"],
     * divisor of divide operation != 0,
     * attribute value's "value" canSetFromString
     * </pre>
     * @ensures evaluate = [the value of the expression]
     */
    private static NaturalNumber evaluate(XMLTree exp) {
        assert exp != null : "Violation of: exp is not null";
        assert !exp.label().equals(
                "expression") : "Violation of: root of exp is not <expression>";

        NaturalNumber zero = new NaturalNumber2(0);
        NaturalNumber result = new NaturalNumber2(0);

        if (exp.label().equals("number")) {
            if (!result.canSetFromString(exp.attributeValue("value"))) {
                Reporter.fatalErrorToConsole(
                        "Cannot convert from string to Natural Number");
            }
            result = new NaturalNumber2(exp.attributeValue("value"));
        } else {
            result = new NaturalNumber2(evaluate(exp.child(0)));
            if (exp.label().equals("plus")) {
                result.add(evaluate(exp.child(1)));
            } else if (exp.label().equals("minus")) {
                if (result.compareTo(evaluate(exp.child(1))) < 0) {
                    Reporter.fatalErrorToConsole(
                            "Subtraction results in negative number");
                }
                result.subtract(evaluate(exp.child(1)));
            } else if (exp.label().equals("times")) {
                result.multiply(evaluate(exp.child(1)));
            } else if (exp.label().equals("divide")) {
                if (evaluate(exp.child(1)).compareTo(zero) < 1) {
                    if (evaluate(exp.child(1)).compareTo(zero) < 0) {
                        Reporter.fatalErrorToConsole("Divide by negative");
                    } else {
                        Reporter.fatalErrorToConsole("Divide by zero");
```

```
 71                     }
 72                 }
 73                 result.divide(evaluate(exp.child(1)));
 74             }
 75         }
 76
 77         return result;
 78     }
 79
 80     /**
 81      * Main method.
 82      *
 83      * @param args
 84      *             the command line arguments
 85      */
 86     public static void main(String[] args) {
 87         SimpleReader in = new SimpleReader1L();
 88         SimpleWriter out = new SimpleWriter1L();
 89
 90         out.print("Enter the name of an expression XML file: ");
 91 //         String file = "expression.xml";
 92         String file = in.nextLine();
 93         while (!file.equals("")) {
 94             XMLTree exp = new XMLTree1(file);
 95             out.println(evaluate(exp.child(0)));
 96             out.print("Enter the name of an expression XML file: ");
 97             file = in.nextLine();
 98         }
 99
100         in.close();
101         out.close();
102     }
103
104 }
105
```