```java
import components.naturalnumber.NaturalNumber;
import components.naturalnumber.NaturalNumber2;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;
import components.xmltree.XMLTree;
import components.xmltree.XMLTree1;

/**
 * Program to evaluate XMLTree expressions of {@code NaturalNumber}.
 *
 * @author Kevin Haller
 *
 */
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]  and
     *  [the label of the root of exp is not "expression"]
     * </pre>
     * @ensures evaluate = [the value of the expression]
     */
    private static NaturalNumber evaluate(XMLTree exp) {

        //Result NaturalNumber
        NaturalNumber result = new NaturalNumber2(0);
        //Constant NaturalNumber zero
        NaturalNumber zero = new NaturalNumber2(0);

        if (exp.label().equals("plus")) {
            //Plus Operator
            NaturalNumber childZero = evaluate(exp.child(0));
            NaturalNumber childOne = evaluate(exp.child(1));
            result.add(childZero);
            result.add(childOne);
        } else if (exp.label().equals("minus")) {
            //Subtraction Operator
            NaturalNumber childZero = evaluate(exp.child(0));
            NaturalNumber childOne = evaluate(exp.child(1));

            //Catch of illegal operation
            if (childZero.compareTo(childOne) > 0) {
                childZero.subtract(childOne);
                result.add(childZero);
            } else {
                components.utilities.Reporter.fatalErrorToConsole(
                        "ERROR! Violation of .subtract requires clause. child0 < child1");
            }

        } else if (exp.label().equals("times")) {
            //Multiplication Operator
            NaturalNumber childZero = evaluate(exp.child(0));
            NaturalNumber childOne = evaluate(exp.child(1));
            childZero.multiply(childOne);
            result.add(childZero);
        } else if (exp.label().equals("divide")) {
            //Division Operator
            NaturalNumber childZero = evaluate(exp.child(0));
            NaturalNumber childOne = evaluate(exp.child(1));

            //Catch of illegal operation
            if (childOne.compareTo(zero) > 0) {
                childZero.divide(childOne);
                result.add(childZero);
            } else {
                components.utilities.Reporter.fatalErrorToConsole(
                        "ERROR! Violation of .divide requires clause. child1 <= 0");
            }
        } else {
            //Number
            NaturalNumber numNode = new NaturalNumber2(
                    exp.attributeValue("value"));
            result.copyFrom(numNode);
        }

        return result;

    }

    /**
     * Main method.
     *
     * @param args
     *            the command line arguments
     */
    public static void main(String[] args) {
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();

        out.print("Enter the name of an expression XML file: ");
        String file = in.nextLine();
        while (!file.equals("")) {
            XMLTree exp = new XMLTree1(file);
            out.println(evaluate(exp.child(0)));
            out.print("Enter the name of an expression XML file: ");
            file = in.nextLine();
        }

        in.close();
        out.close();
    }
```

```

}

```