CW2

Continuing from Week 1, we consider simple arithmetic problems, like those that pupils in primary school learn to solve. This week, we are considering more complex arithmetic expressions, which could be used to offer more complex exercises for those pupils. Since they usually need a lot of practice, we consider a database with a lot of such exercises and, to avoid wrong model answers (check out this [example](https://www.washingtonpost.com/blogs/answer-sheet/wp/2014/10/08/pearsons-wrong-answer-and-why-it-matters-in-the-high-stakes-testing-era/?tid=sm_fb)), we evaluate them programmatically.

That is, we ask you to write a program which takes arithmetic expressions (such as, 4+5 or 7\*(4-24)+7) in XML format, parses them, evaluates them, and returns the integer answer (in this case, 9 and -133, respectively). We are going to use this task to explore the DOM API in Java and the limits of RelaxNG based validation.

**Functional Description**

For this assignment, you must write a Java program which:

take XML documents, encoding arithmetic expressions according to the calc1.rnc

checks whether those documents are valid w.r.t. calc1.rnc; if yes, it will also check that the arguments are, in fact, integers; it should reject erroneous documents

returns the integer which is the result of evaluating the encoded expression

**Submission**

You will submit (by attachment):

a single file, CalculatorImpl.zip  which is a compressed version of a single file CalculatorImpl.java (15 points)

The program, when placed in the corresponding src directory, should compile using the bundled ant script. There should be no other dependencies.

You must use standard zip (not rar, tar.gz, or any other variants).

If your program fails to compile, then you will receive a 0 for this assignment.

A stub for this program is attached along with an ant build script, some supporting classes, calc1.rnc plus some test cases. You should only modify CalculatorImpl.java

**Tips**

Working programs are prefered to non-working that try to cover everything. For example you could start by evaluating simple expressions; hence you should start by creating some simple test cases (i.e., XML documents that are valid wrt calc1.rnc). Then you could try to work on more complex ones; hence you would then create more complex test cases. Then deal with the error handling.

We will **not** read your source code (except to check for API use) and we will **not** debug your program.

You should still comment your code reasonably.

Write test cases (see above); feel free to share these on the forum for Week 2.

Partial credit is given on the basis of testing. A (working) program that can only evaluate addition is worth some points.

As with all other coursework, it's not acceptable to collaborate on this assignment. However, you may post test cases to the Blackboard discussion area (and use such tests from there).

Test cases whose name conforms to a certain convention when placed in the "testfiles" directory are automatically evaluated by the ant run task. The key to the name convention is that the *expected result* is encoded in the name. E.g., bjp\_test1\_9.xml says that the expected result is 9. (The result appears between the last underscore and the last dot.) There are two special results for erroneous input: errnan for arguments which are not numbers and errinvalid for documents which are invalid. A test is errnan if and only if it is valid.

If you are going to share tests, please prefix the test name with your name and number your tests. E.g., bijan\_parsia-test4\_errnan.xml is my fourth test and it has a non-numberic argument.

M2

This modeling exercise is closely related to CW2. I suggest you look at and think about CW2 first. Then consider M2, then decide where to start working.

Our RelaxNG calc1.rnc (and the corresponding calc1 format) has a number of limitations: for example, it's possible to have documents which are valid w.r.t. it and yet are not legal arithmetic expressions. For example:

<expression>  
    <!-- This expression evaluates to 20 -->  
    <plus>  
        <int value="five"/>  
        <int value="10"/>  
        <int value="5"/>  
    </plus>

</expression>

Furthermore, calc1 doesn't cover many useful operations, such as division. Nor is it possible in RelaxNG to express certain constraints (e.g., that we accept numbers only in the range -1000 to 1000; why might we impose such a restriction?).

Your task is to produce a RelaxNG schema for an extended format calc2. The calc2 format is given below in English. Your schema should enforce all the given constraints. You should strive to make your schema "nice", i.e., readable, maintainable, evolvable, etc.

**Specification**

You will describe a new format, calc2, in a RelaxNG schema called calc2.rnc. This format has the following properties (the more of these properties your schema covers, the more marks you get):

Every XML document that is valid wrt calc1.rnc and has only integers inside the range -1000 to 1000 as the value of an int element is valid wrt calc2.rnc. Any other document which is valid wrt calc1.rnc is invalid wrt calc2.rnc.

calc2 must support decimal numbers with a new element named "dec" that can occur wherever int elements can occur.

calc2 must support unary plus and minus (e.g., to cover expressions like "3 \* (-3)").

calc2 must support a binary division operator, called "div".

calc2 does not permit inputing numbers outside the range -1000 to 1000 (whether integers or decimals)

calc2 must support a "comment" element. The content model of the comment is a subset of XHTML consisting of the "i" and "b" tags interleaved with text (but not nested), and comment elements can occur as optional child elements of expressions.

Your schema contains suitable comments for other users to understand it.

**Submission**

You will submit (by attachment)

a single file, calc2.rnc (10 points)

We've provided no stub, but for your convenience we've attached a copy of calc1.rnc. Look at this carefully: the comments will help you around the first constraints.

**Please note**

These constraints are strict: if your WXS satisfies all 7 constraints, then you will get full marks. For example, we ignore any issues around division by zero, i.e., a document corresponding to "1/0" can be valid wrt your WXS.

[calc1.rnc](https://online.manchester.ac.uk/bbcswebdav/pid-3733527-dt-content-rid-12501962_1/xid-12501962_1)

SE2:

Briefly (in **no more than** 300 words), explain an application where we have a situation that does not require input documents to be valid (against a DTD or a RelaxNG schema, etc.) but instead merely **well-formed**. Explain as well how a RelaxNG schema might still be useful in this situation.