CSC319 Workshop 2

Instruction:

Commands and expressions interpretation or evaluation is commonplace in computing applications. Where comparation and conditional logics are straightforward to implement such interpretation tasks, one can also rely on the Interpreter pattern for this very purpose. If existing code exists that relies on the typical **if-else** or **switch-case** statements to interpret and respond to the commands, it might be better to refactor to implement the Interpreter instead.

In this workshop, you will perform the following tasks:

1. Learn, using online resources, about the Interpreter pattern.
2. [**Iteration** 1] Implement a simple arithmetic expression evaluator by using the Interpreter pattern. For this simple task, you only need to have a way to evaluate the integer number and the addition expressions. Both the integer value and the addition expression should be treated as the "terminal expression" of the Interpreter tree.
Remark: Important! 1 class per file only.
3. [**Iteration** 2] Extend the current Interpreter by adding the following capabilities:
    - The "++" operator that takes on just one single integer operand and increment its value by 1;
    - The "random" operator that takes on 2 integer values and produce the random integer value in the range of [*smallerValue*, *biggerValue*). However, if only 1 integer value is provided, it will be interpreted as producing a random integer of value [0, *singleValue*). If no integer parameter value is provided at all, it will be interpreted as producing a random integer of value [0, *maximumIntegerValue*).
4. [**Iteration** 3] Extend the current Interpreter by allowing it to interpret Boolean expressions.
    - Do NOT overload the "interpret or evaluate" method of your Interpreter tree. There must be only one instance of this method in each of the components in the interpretation tree implementation.
    - Add the capability to perform the XOR operation of 2 Boolean operands.
5. [**Iteration** 4] Add a special "context" that will reverse the expressive values when being interpreted by the Interpreter. In other words, your Interpreter will operate in 2 modes: the normal mode and the reverse mode. For example, in the reverse context, the value of '1' will be interpreted as '-1'; 'true' as 'false' and the random value of range [0, 9) will be interpreted as [-9, 0).

Common requirements for all iterations:

- Draw the corresponding class diagram of each iteration's implementation using either Astah or Visual Paradigm. Screenshot and make the pdf file out of it. Place this pdf file in each corresponding iteration directory.
- On you class diagram, you must also "explicitly specify" the class with the main method, i.e. *public static void main(String[] args)*.
- Make sure that the iteration directory is flat.
- Put all iteration folders in the same "workshop_2" directory. Zip and submit as one single zipped file.

Interpreter Pattern:
https://medium.com/@rajeshvelmani/understanding-language-interpretation-with-the-interpreter-design-pattern-in-java-b2a3969eaf9