A few months ago I had a task to write a kind of a special expression language for my Easy SmartHouse project. Actually, almost every mature framework or project has its own expression language on board and today we’ll give a try to create some semblance of it.

Of course, it’s a very difficult work to write something universal and suitable for production stage like [spring-expression](https://github.com/spring-projects/spring-framework/tree/master/spring-expression), but it wasn’t my goal. My purpose was to provide some flexible way to react on external events or circumstances that may occur in the smart home.

These events are defined with many devices and sensors which are located throughout the house and the logic of their interaction can be very complicated. For example, we may have 10 thermometers, 20 humidity sensors and only one barometer and if even one of humidity sensor's value rose up to 90%, a thermometer's measure dropped up to a value 10C and barometer's value dropped up suddenly it would be great if windows of our home will be closed up automatically. This logic defines the event and, as practice shows, it changes very often.

But using your expression language we may define this event in this way:

${(humiditySensor1>90 or humiditySensor2>90 or … humiditySensor10>90) and (thermometer 1<=10 or thermometer 2<=10 or … thermometer 10<=10) and (barometer1<765)}

This expression has to be parsed, every condition has to be checked and the result should be true or false. So what is the simplest way to achieve this?

For the first let’s define the expression itself:

Every operation (or, and, < etc.) may be considered as a small expression and interpreter pattern may help us to describe its interaction with an “EvaluationContext”.

For example, this is an “and” operation:

The next part of the application is an “EvaluationContext”, where evaluation results stored:

It’s the simplest implementation may use HashMap internally:

And obviously, the most essential part of an expression language is a parser:

As you see Its realization resembles [Dijkstra's algorithm](https://github.com/creepid/algorithms-research/blob/master/src/main/java/by/creepid/algorithms/basic/bags/DijkstrasSimple.java), a very popular approach for parsing expressions.

Now let's gather all the parts all in one place and test it.

ExternalContext is a container where external variables may be taken.

As a result we have a very simple expression language realization.

Links:

**This example in Github** - <https://github.com/creepid/easy-expression-language>

**Interpreter pattern** - http://www.tutorialspoint.com/design\_pattern/interpreter\_pattern.htm

**Dijkstra's algorithm** - https://github.com/creepid/algorithms-research/blob/master/src/main/java/by/creepid/algorithms/basic/bags/DijkstrasSimple.java