Assignment 2: External DSL interpreter

Description

> In this assignment, you should create an external DSL for a math interpreter. To complete the assignment, you should: fix the broken implementation sketch, both the Xtext grammar and code generator file (interpreter in this case). Use the JUnit tests to check how well your implementation works, and report on whether it passes all JUnit tests in a given test class. You can modify the parser and interpreter file as much as you need. Optionally, you can replace the dialog box with a hovering box, as shown in Bettini.
>
> Hand in your response as a single pdf that includes
>
> 1. A brief description of the status of your assignment, in particular, which JUnit tests you pass
> 2. Your xtext file
> 3. Your generator file
> 4. Optionally, include code needed to implement hovering

Solution

A math interpreter has been implemented, using xtext and code generation / calculation using xtend, as well as validation and scoping.

The solution passes all provided tests; `MathExampleTest`, `MathParsingTest`, `MathScopeTest` and `MathValidatorTest` tests.

The solution handles general math, as covered by the tests. This includes addition, subtraction, multiplication, and division.
It can also handle sub-expression, with variables being used only within the sub-expression. this is done by using the `let` keyword, a single variable declaration, `in` keyword, and then the expression.
Example: `let i = 2 in a * i end`

The system can handle that the variables are not defined in the order they are used in other variables.
If variables, that are not defined, are used, the system will provide an error. Variables defined inside sub-expressions are only available in that sub-expression. If an existing variable exists, it will be replaced within the sub-expression.

Hovering box is not implemented in this solution.

Calculated variables are stored in `output.txt` file in the `src-gen` directory.

Tests have been *slightly* modified to handle a top-level `Maths` model, instead of only a single variable.
It is only the `ParseHelper` injection that have been changed.

Example usage

The following is a valid math expression that can be parsed and calculated by the interpreter:

```
var a = 40
var b = let i = 2 in a * i end
var c = b * 3
var d = 40 + 2 * 4 + 80
var e = (40 + 2) * (4 + 80)
var x = let i = b in b + c + d + e + i end
var y = let y=2 in let z=3 in y*z end + y end + 79
```

The following would be the output of the above expression:

```
var a = 40
var b = 80
var c = 240
var d = 128
var e = 3528
var x = 4056
var y = 87
```

Source code

The source code for assignment 2 is available at github.com/The0mikkel/sdu-mbsd.
A pull request is available at github.com/The0mikkel/sdu-mbsd/pull/2, where changes from the provided code can be seen.

Xtext file is available at `Math.xtext`.
Generator file is available at `generator/MathGenerator.xtend`.

This document is available at github.com/The0mikkel/sdu-mbsd/assignment2/README.md.