Thoughts and Observations

Important Points:

Validation/Grammar

- Grammar is too lenient for writing validations.
- Different sections/areas of Symboleo code share the same terminals in the grammar and too many things in the grammar were simply defined as a “name”.
- Grammar needs to impose validation (e.g. a Declarations should reference a contract parameter in the grammar, instead of creating a new “name” variable and expect the user to name them the same thing). We cannot validate if a newly created variable that is just the name is the same type as another “name” variable created in the contract parameters.

Testing Notes (Testing Generator):

What is New

- Added Tests to” SymboleoParsingTest.xtend” under Symboleo.tests > src.
- Added Resources folder under symboleo.tests
- Resources Folder contains “input.properties” which holds all the inputs used in tests.
- “expected.properties” contains all the tests’ expected results
- “GetInputValues.java” is the getter class called from “SymboleoParsingTest.xtend” in order to retrieve the input and expected values.

*NOTE: When calling GetInputValues class from a test you MUST specify whether your requested value is an input or the expected result (“input” or “expected”). Spelling must be correct.

```
val model = parseHelper.parse(this.inputProperties.getInputValues("badDeclarationCombinationVariable", "input"));
val expected = this.expectedProperties.getInputValues("badDeclarationCombinationVariable" , "expected");
```

How it works

- All Test case formats are the same. Compare String input in Symboleo to expected String Output in Prolog

Format

1. Create temporary file in memory
2. Create Generator context
3. Retrieve input and expected output data
4. Call Generator by passing in these values
5. AssertEquals your expected results and the actual generated results.

Important Notes

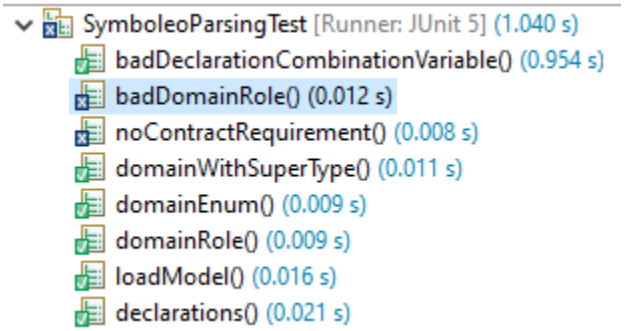
- Use the helper method “removeWhiteSpaces(String)” before comparing actual results v.s expected results.
- If this is not done then tests WILL FAIL even though the expected result and actual result are exactly the same values. This is due to an issue concerning white spaces and white empty lines.
- As a convention, the variables in the “.properties” files carrying the input and expected strings should have variables that have the same names as the respective test case names they will be used in. Preferably everything should be added in the same order as well inside the files.
- To create a variable that spans multiple lines in the properties files you must add a “\” at the end of the line. You MUST also add a white space “ “ before or after the backslash so the next line does not get directly concatenated to the previous, this will result in your tests failing.

Negative Tests

Certain rules need to be met for a contract to be valid, otherwise the generator should not generate anything. Right now, only in the testing environment the generator *WILL* produce prologue code for an invalid contract. These tests should be expecting a NullPointerException as the generator shouldn’t be producing any code.

AssertThrows() is what should be used instead of AssertEquals and should be expecting a NullPointerException, although this is not working right now.

A work-around solution was implemented using “try-catch” statements where the test is rigged to pass if a NullPointerException is caught, and otherwise fails. 2 tests currently fail as expected since the generator should not be producing any prolog with their respective invalid contract inputs.



Resources used to set up testing

- https://crunchify.com/java-properties-file-how-to-read-config-properties-values-in-java/
- https://docs.oracle.com/cd/E23095_01/Platform.93/ATGProgGuide/html/s0204propertiesfileformat01.html
- https://howtodoinjava.com/junit5/expected-exception-example/