Kanica Ahuja, Nikhita Naramsetti, Tinesh Madan Mohan

Abstract

Create a program that generates syntactically correct but semantically meaningless program based on parsing the JAVA BNF Grammar rules

Final Project – cs474

Object Orient Languages and Environment

**ABSTRACT**

The project creates a generator that creates a syntactically correct but semantically meaningless Java application for a given configuration file. Configuration file(input.xml) will be having the number of classes, methods, expression and other parameters that constrain generation of java program.

A subset of Java BNF grammars had been used while generating the Java code from the already defined Java grammar. When the “Main” is run, the program auto generates syntactically correct but semantically meaningless Java code.

There exists class where production rule of grammar has been specified. “ParseGrammar” is the class which decodes the production rule. In addition to the rules that are found in a typical context-free grammar of a programming language, additional rules and constraints that are imposed by the programming language specification are considered by the “Structure\_Final” class which generates the program.

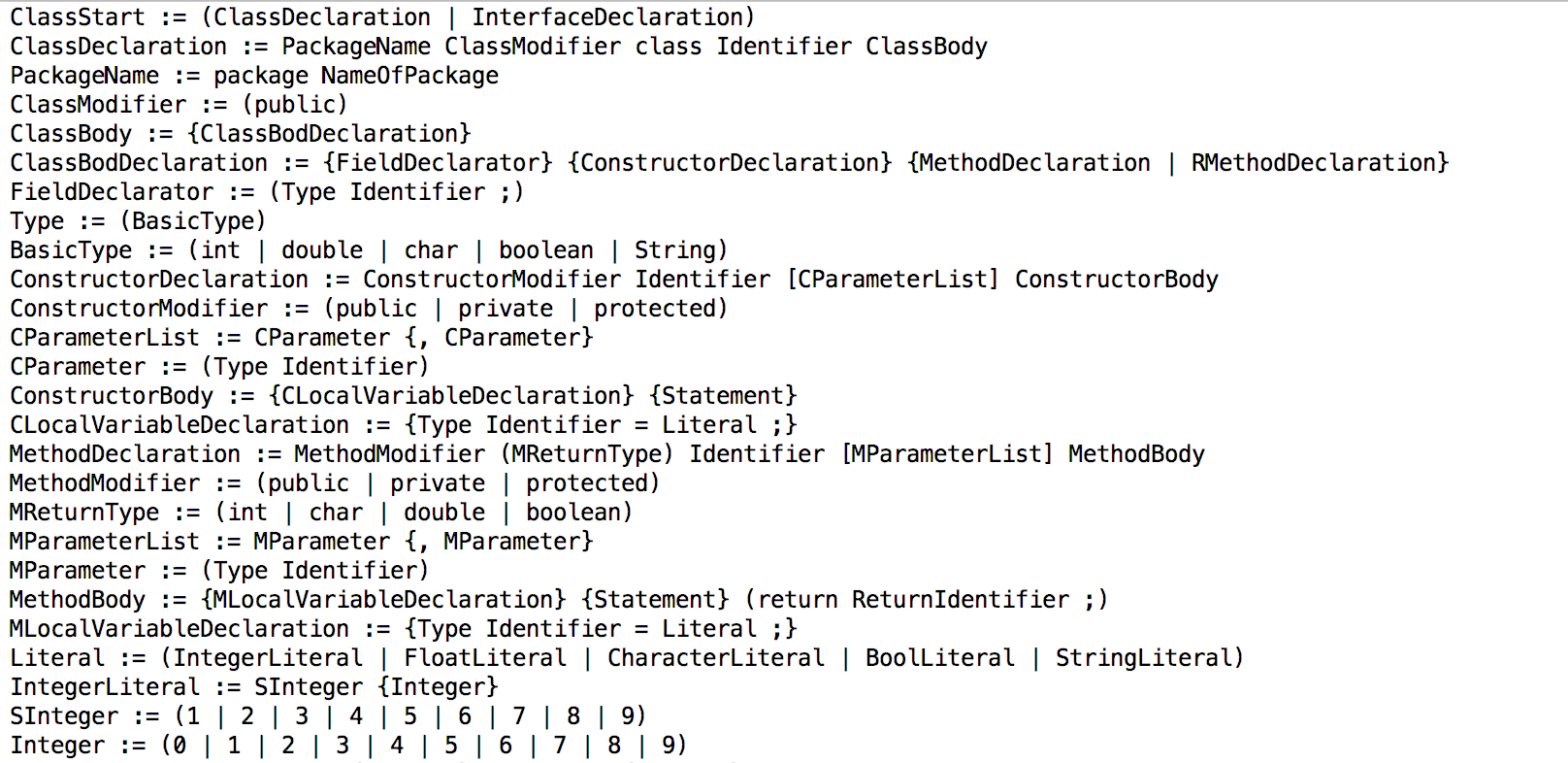
**INTRODUCTION**

**Parsing:**

The first step in the implementation of this project is to parse the grammar that is written in the “java3bnf.grammar” file. The **ParserGrammar** java class does the parsing with the main logic residing in the **parse\_rhs** function. The function parses through the grammar and stores the result in a *LinkedHashMap* of Node. The Node is the java file **Node**, which contains a basic data structure to store the details of the node, relating to whether it’s a terminal or the next rules we can apply on it.

*Grammar Rules:*

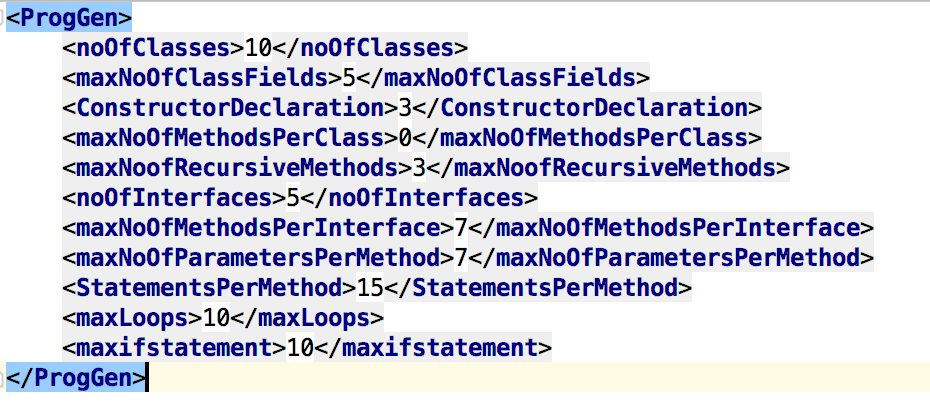
* {] **->** There can be Zero or Many occurrences
* [] **->** There can be Zero or one occurrences
* () **->** It is a single occurrence and is mainly used for grouping
* No Brackets **->** Non – grouped single occurance
* | **->** To indicate an OR.



**Input Configuration:**

The next step in the implementation is to read the **input.xml** and parse the XML values into the **Generator\_Specs**  object which is done by the **XMLConfig** java. The Generator\_Specs contains class fields for the corresponding entries in the XML input to configure the parser as much as possible.

*XML Rules added as :*



To add new XML rules, keep appending at the end of the list, the corresponding variables need to declare in the *Generator\_Specs* with respect to the current implementation, which will be used by the generation function.

**Program Generation:**

The main step in the implementation would be to parse through the grammar and generate the program. This would require us to implement the JLS specifications while generating to make sure our code is syntactically correct and compiles. The logic is handled in **Structure\_Final** java file. The main processing resides in the **generate\_program** function which recursively iterates through the production rules provided after parsing through the grammar file.

The function takes care of basic constraints like the occurrences, randomization while parsing through the production rules. It provides chances to various symbols based on the OR condition in the group, choosing one from it and specifying a random number of times a production rule or symbol is called. While the basics are straightforward, they do not guarantee a syntactically correct code since the JLS specification provides constraints like type matching, initialization of variables before usage, etc.

The generator while recursively calling itself, appends the generated code to the StringBuilder object that is been passed to it, appending more and more as we progress through the grammar. The function handles few corner cases like:

* Constructor duplication
* Constructor overloading
* Method Parameter and local variable Scoping
* Looping levels
* If, Else if Condition levels
* Basic Operations
* Recursion
* Return types
* Etc.

The scoping is handled by **ScopeTable** java file that handles all the necessary scoping information and finally post generation, we write it onto the corresponding files.

**How to run Project:**

* Make sure the all dependency library issues are all resolved.
* Configure input.xml by providing the desired input of various fields.
* Then run the main from Main.java
* Syntactically correct but meaningless java programs will be generated with auto creation of respective class files in a different folder “generated\_programs/GenaratedClasses”.

*Sample Directory Structure:*

