IPP Project

parse.py

The main process of the SOL25 parser implementation works on the foundation of using Lark library, focused on defining and processing grammar in the code, with its specific set of features and limitations to consider beforehand. The grammar was defined as strictly as possible to be close to the main task, with slight differences needed for adjustment such as transforming individual rules, for example `Method → Selector Block Method` to `method: (selector block)*`. The specific approach required caused some limitations including later realization of potential simplification of the overall subsequent process by making the grammar more divided in smaller pieces, thus requiring less advanced reaches into the nested children of the Tree and Tokens later.

The lexical and syntactic analysis thus mainly rely on the Lark's ability to process these two, however there were some limitations like reserved keyword use, which needed to be handled later in the implementation. The builtin exceptions handle the `UnexpectedCharacters` and `UnexpectedToken` as lexical and syntactic Errors 21 and 22 respectively.

Semantic analysis is what most of the code consists of and is done through Lark's Visitor class instances and topdown approach of passing through them, individually made to check the specific error which were expected. They execute one by one, however firstly the Reserved Keyword visitor is used to check the incorrect use of unwanted keywords in the code. Following after is the check for Main class and run method, which verifies the presence of Main with its run method block, in other case causing the Error 31. After is the check for arity inside blocks, with block parameters count and preceding identifier variables potentially causing the Error 32. Next follow the verification of absence of colliding variables, causing Error 34, and also checking for duplicate parameters in the block.

Last is the biggest Visitor, which serves for verifying the use of only defined classes, methods, and variables, those in their own blocks exclusively. The implementation was longest as it required many nested conditional `if` statements and saving the individual elements for later checking. This visitor uses predefined elements, mostly methods, but also constructor such as `new` and `from`. Those were all implemented using sets and later utilized with the help of set unions. This Visitor class also uses the `__init__` constructor for storing variables that remain consistent throughout the life of the class, but there were also outside variables used for storing the saved elements.

The xml tree is created with the help of `xml.etree.ElementTree` which was implemented using Transformer that transforms individual parts from the grammar and appends the processed parts into the final result. This part also posed challenges and with some details was not as clear as some other parts of the implementation. The xml uses utf-8 encoding and decoding for the standard output also enabled in pretty form thanks to domXML class.