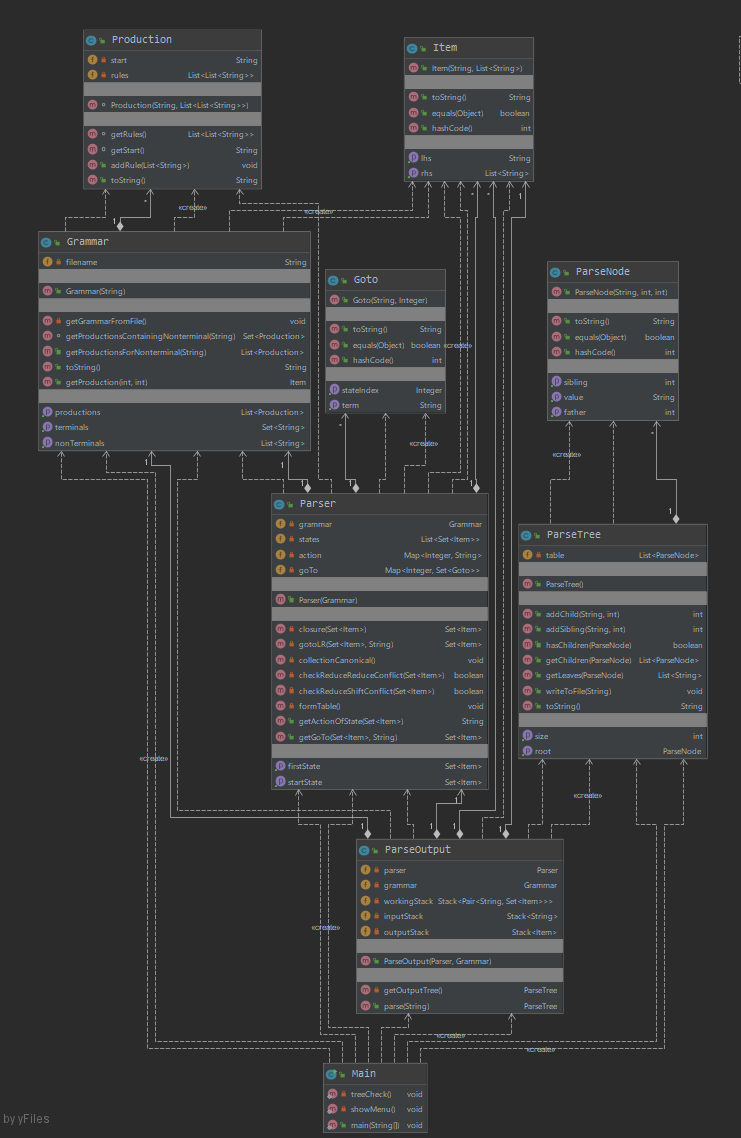
<https://github.com/NeniscaMaria/Parsing>



Class Main:

* Methods:
  + treeCheck(): function that tests the ParseTree class
  + showMenu(): function that shows the menu from which the user can see their options
  + main()

Class ParseOutput:

* Fields:
  + Parser: a non-null parser of the grammar
  + Grammar: a non-null grammar of our language
  + workingStack: a stack with elements of the form <String, Set<Item>>. Set<Item> is the representation of a state.
  + inputStack: a stack with strings
  + outputStack: a stack with Items = the productions
* Methods:
  + Constructor(): receives the parser and the grammar of the language
  + Parse(String):
    - INPUT: sequence = the sequence we want to parse
    - OUTPUT: parsing table if the sequence was accepted, and null otherwise
  + getOutputTree():
    - OUTPUT: parsing table corresponding to the current outputStack

Class ParseTree:

* Fields:
  + Table: a list of ParseNodes
* Methods:
  + Constructor(): no parameters
  + addChild():
    - INPUT: childValue = non null string and parentPosition = int
    - OUTPUT: the position of which the child was placed
    - DESCR: adds a child with the parent on position parentPosition and returns the position in the table were the child was placed
  + addSibling():
    - INPUT: siblingValue = non null string and siblingPosition = int
    - OUTPUT: the position on which the sibling was placed
    - DESCR: adds a sibling with value siblingValue to the node on position siblingPosition and returns the index in the table where the new node was placed
  + hasChildren():
    - INPUT: node : ParseNode nonull
    - OUTPUT: True if the node has children, False otherwise
  + getRoot():
    - DESCR: returns the node at the root of the tree
  + getChildren:
    - INPUT: node : ParseNode nonnull
    - OUPUT: a list of nodes which are the children of node
  + getLeaves():
    - INPUT: node = ParseNode nonull
    - OUTPUT: list of the values of the leaves of the tree with the root node
  + writeToFile():
    - INPUT: filename = string
    - DESCR: writes the current table to the file filename

Class Parser:

* Fields:
  + Grammar = the grammar of the language
  + States = list of the states found during initialization
  + Action = the action table of the language
    - A map with inputs of the form: <index in states list, action>
  + GoTo = the goto table
    - A map with inputs of the form: <index in the states list, GoTo>
* Methods:
  + Constructor: takes the grammar
  + Closure():
    - INPUT: state = Set<Item>
    - OUTPUT: state = Set<Item>
    - DESCR: returns what a state contains
  + gotoLR():
    - INPUT: s=Set<Item> and x = String
    - OUTPUT: state=Set<Item>
    - DESCR: returns how to move from state s by x to another state
  + getStartState():
    - OUTPUT: returns the first state from the states list
  + getFirstState():
    - OUTPUT: returns the first production with a dot on the first position of the rhs
  + collectionCanonical():
    - DESCR: constructs the set of states
  + checkReduceReduceConflict():
    - INPUT: state = Set<Item>
    - OUTPUT: True if the state has a reduce-reduce conflict, false otherwise
  + checkReduce-ShiftConflict():
    - INPUT: state = Set<Item>
    - OUTPUT: True if the state has a reduce-shift conflict, false otherwise
  + formTable():
    - DESCR: forms the action and goto table
  + getActionOfState():
    - INPUT: state: Set<Item>
    - OUTPUT: String = error if there is no such state or the action corresponding to the input state
  + getGoTo():
    - INPUT: state=Set<Item> and a = string
    - OUTPUT: state=Set<Item>
    - DESCR: returns the state in which the given state goes through a, or null if there is no state with such property

Class GoTo:

* Fields:
  + Term = String
  + stateIndex = integer
* methods:
  + constructor():
    - INPUT: term string and stateIndex integer
    - DESCR: the GoTo class maps a term to the state it goes to

Class Item:

* Fields:
  + Lhs = string = left hand side of a production
  + Rhs = List<string> = right hand side of a production
* Methods:
  + Constructor():
    - INPUT: lhs string and rhs List<String>
    - DESCR: The Item class describes an item of a state.

Class Production:

* Fields:
  + Start = string
  + Rules = List<List<String>>
* Methods:
  + Constructor():
    - INPUT: start string and rules List<List<Rules>>
    - DESCR: production class is the representation of a production. Ex: S – A | aA | bB

Class Grammar:

* Fields:
  + NonTerminals: a list of NonTerminals
  + Terminals: a set of terminals
  + Productions: a list with productions
  + FileName: a string with fileName
* Methods:
  + Constructor(): receives the fileName
  + getGrammarFromFile():
    - OUTPUT: get the grammar from the file
  + getProductionsContainingNonterminal():
    - INPUT: string NonTerminals
    - OUTPUT: list of productions that containing Nonterminals
  + getProductionsForTerminal():
    - INPUT: string Terminals
    - OUTPUT: list of productions that containing terminals
  + getProduction():
    - INPUT: int posInProds, int posInRules
    - OUTPUT: production