LR(0) Parser

class: Grammar
- class responsible for holding all the parameters of a grammar: the set of terminals, the set of non terminals, the productions and the starting non terminal
void readFromFile(file)
- function responsible for parsing the grammar file (g1.txt/ g2.txt) to retrieve the parameters of the grammar: the set of terminals, the set of non terminals, the productions and the starting non terminal; the function parses the file according to the structure mentioned below:

g1.txt:
S, A
a, b, c
S
<S> ::= a <A>
<A> ::= b <A>
<A> ::= c

Boolean isCFG()
- function responsible for checking is the grammar is CFG or not; the function iterates through the map of productions and checks if the lhs of each production has only one element, then the grammar is CFG
Production enhanceGrammar()
- function responsible for enhancing the grammar with an 'initial' production SS goes into the starting symbol

Void printTerminals()
- function responsible for printing all the terminals of the grammar

Void printNonTerminals()
- function responsible for printing all the nonterminals of the grammar

Void printProductions()
- function responsible for printing all the productions of the grammar

Void printProductionsForGivenNonTerminal()
- function responsible for printing all the productions of the grammar that have on the left hand side a certain terminal

class: ProductionSet
- class responsible for holding the productions used in the grammar: a map of the form List<String> as key representing the lhs of a production and List<List<String>> as value representing the rhs of a production

class: Production
- class responsible for holding the parameters of a production: the rhs, the lhs and the index of the point
Boolean isPointAtEnd()
String getNextSymbol()
- function that gets the next symbol after the point of the rhs

class: LRParser

- class responsible for holding all the parameters of a LR(0) Parser: the grammar, the canonical collection, the parsing table, the parsing strategy and the parsing output which computes the parsing tree

boolean hasNonTerminalAfterDot(Production production)

- checks if a production has a non terminal right after the dot position

State goTo(state, symbol)

- for each LR0 item in the state, move the dot if the symbol follows it; perform closure on the modified item

State closure(productions)

- for each item in the item list, if after dot there is a non-terminal, add it to the list of items and repeat the process

void computeCanonicalCollection()

- for each state s in the canonical collection, for each symbol X (in both terminals and non-terminals), check if goto(s,X) result is not an empty list nor exists already in the canonical collection and if so add it to the canonical collection

void createParsingTable()

- Creates the parsing table from the states of the canonical collection and the existent symbols in the grammar. The parsing table is filled out row by row by iterating through the states, completing the action column depending on the productions in the respective state from the row and by filling out the gotos of that respective state with all symbols that led to a non empty state. In case there are contradicting actions an error is outputted specifying the row in the parsing table.

Boolean parsingAlgorithm()

- performs the parsing algorithm using 3 stacks (input, working and output) handling each type of action for a state: shift, reduce or accept.; the workingStack is a list considered a stack -> (meaning the top of the stack is the right most element), and the inputStack is also a list considered a stack <- (meaning the first element is the top of the stack); at the end the output band is computed and it is checked if the sequence is accepted or not

Boolean isSequenceAccepted(String sequence)

- checks if a sequence is accepted

Boolean isSequenceAccepted(List<String> sequence)

- checks if a sequence is accepted

class: ParsingStrategy

- class responsible for holding the parameters used for parsing a sequence: work stack, input stack and output band, all represented as lists

class: State
- class responsible for holding the parameters of a state: a set of productions

class: ParsingTable
- class responsible for holding the parameters of a parsing table: the table itself, the number of states (= no of rows in the table) and the symbols

Void addPredefinedHeaders()
- function responsible for adding the column headers: action + symbols of the grammar

Void setCell(int row, String column, String value)
- function responsible for setting the value in the parsing table located at a certain row and column to a given new value

String setCell(int row, String column)
- function responsible for getting the value in the parsing table located at a certain row and column

class: Pair
- class responsible for holding a cell in the parsing table, identified by row and column

class: ParsingOutput
- class responsible for translating the productions string to father-sibling tree representation and for printing the tree to the screen and in the file

List<Node> getParsingTree(List<String> productionsString, Grammar grammar)
- function responsible for transforming the productions string to tree representation (as a list of nodes)

List<Node> printParsingTreeToScreen(List<String> productionsString, Grammar grammar, String fileName)
- function responsible for printing the parsing tree resulted from the productions string to the screen and in a specified file

class: Node
- class responsible for holding the parameters of a node used in the parsing tree: the parent, the right sibling and the value

class: HelperNode
- class responsible for holding the parameters of a helper node used to help the nodes of the parsing tree: the parent (stored as a reference to another Helper Node), the right sibling, the value and the level