1. calculateFirstSets()

- initialize an empty set for each non-terminal.

- iteratively processes the productions (rules) of the grammar, adding terminals to the FIRST set of each non-terminal. When a production has a sequence of non-terminals that can derive an empty string (epsilon), it continues to the next symbol in the production.

2. calculateFollowSets()

- initialize an empty set for each non-terminal.

- The end-of-input marker `$` is added to the FOLLOW set of the start symbol.

- examine each production. When a non-terminal is found, the method looks at the symbols following it. If there is a terminal, it's added to the FOLLOW set. If there's a non-terminal, the FIRST set of that non-terminal is added to the FOLLOW set (excluding epsilon). If epsilon is in the FIRST set, or if the non-terminal is at the end, the FOLLOW set of the left-hand side non-terminal is added.

3. createParseTable()

1. Numbering Productions:
   * Before creating the parse table, the method calls numberingProductions(), which assigns a unique index to each production in the grammar. This indexing is used later in the parse table to refer to specific productions.
2. Initializing the Table:
   * The table is initialized for basic actions like pop and acc (accept). The method sets up actions for each terminal symbol and the end-of-input symbol ($). For example, it maps the pair of a terminal symbol followed by the same terminal symbol to the pop action, and the pair of $ followed by $ to the acc (accept) action.
3. Filling the Table with Grammar Rules:
   * The method iterates over each numbered production in the grammar. For each production, it examines the first symbol of the production's right-hand side.
   * There are two main cases to consider:

a. First Symbol is a Terminal or Epsilon: If the first symbol is a terminal (or epsilon ε), an entry is added to the parse table for this terminal (or epsilon) and the current non-terminal.

b. First Symbol is a Non-Terminal: If the first symbol is a non-terminal, the method retrieves the FIRST set of this non-terminal. An entry is added to the parse table for each terminal in this FIRST set and the current non-terminal. If the FIRST set contains epsilon, entries are also added for each terminal in the FOLLOW set of the current non-terminal.

1. Handling Epsilon Productions:
   * If a production can derive epsilon (i.e., it's an epsilon production), the method adds entries to the parse table for each terminal in the FOLLOW set of the non-terminal on the left-hand side of the production. These entries point to the epsilon action.

4. parse(List<String> w)

- It initialize stacks for the input and working process. The working stack starts with the start symbol of the grammar, and the input stack is loaded with the input tokens followed by the end marker `$`.

- The function then enters a loop, where it repeatedly looks at the top of both stacks (working and input). Using the parse table, it decides whether to apply a production (replacing the top of the working stack with the right-hand side of the production), pop a symbol (if the top symbols match), or accept the input if both stacks have only the end marker left.

- If the parse table does not have an entry for the current top symbols, it indicates a parsing error.