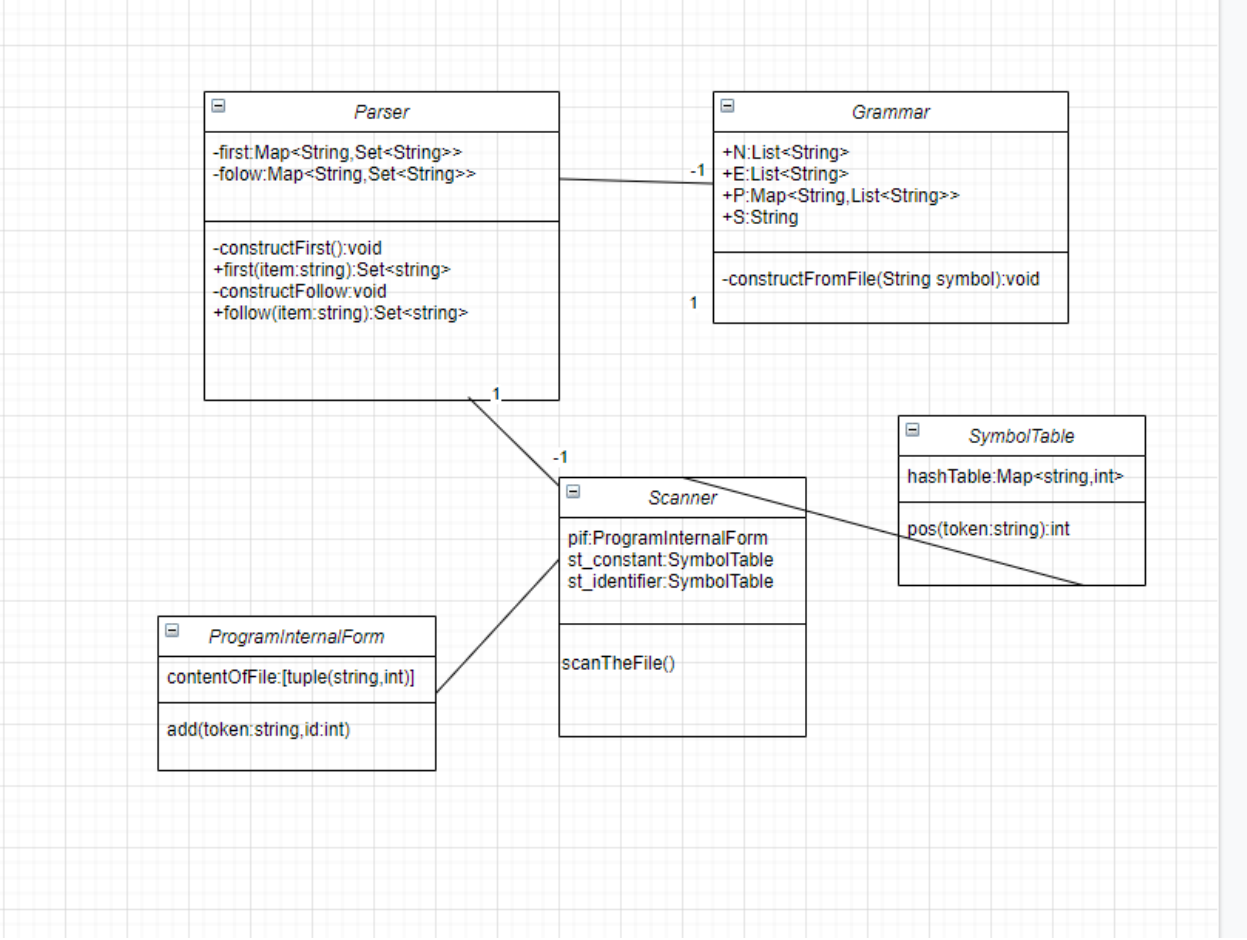

Class diagram for the parser and the grammar in the current state of the application.

```
def constructFirst(self):
    for terminal in self.grammar.getTerminals():
        self.firstSet[terminal] = {terminal}

    for nonTerminal in self.grammar.getNonTerminals():
        initial = set()
        for production in self.grammar.getProductsForNonTerminal(nonTerminal):
            productionElements = production.split()
            if productionElements[0] in self.grammar.getTerminals():
                initial.add(productionElements[0])
        self.firstSet[nonTerminal] = initial

    modified = True
    while modified:
        modified = False
        for nonTerminal in self.grammar.getNonTerminals():
            for production in self.grammar.getProductsForNonTerminal(nonTerminal):
                productionElements = production.split()
                go = True
                for item in productionElements:
                    if len(self.firstSet[item]) == 0:
                        go = False
                        break
                if go:
                    concatResult = copy.deepcopy(self.firstSet[productionElements[0]])
                    if 'ε' in concatResult:
                        concatResult.remove('ε')

                    for i in range(len(productionElements)):
                            concatResult = self.generateConcatenationOfLengthOne(concatResult,productionElements[i])

                    if not concatResult.issubset(self.firstSet[nonTerminal]):
                        self.firstSet[nonTerminal] = self.firstSet[nonTerminal].union(self.firstSet[productionElements[0]])
                        modified = True
```

FIRST algorithm. It follows what was described in the professor's description only that not all sets are checked but the a boolean value is observed to check wheter any changes occur.

At the initalization the all terminals get the first set as themselves then all non-terminals will get the the first terminal that appears as the first item in each production where the non-terminal is at the left hand side. After that for each non terminal if the for all items on the right hand side we have a non-empty first set then we will generate in the concat result is the result of the concatenation of length one.

```
def generateConcatenationOfLengthOne(self,firstSet,secondSet):
    resultSet = set()
    for itemOne in firstSet:
        for itemTwo in secondSet:

            concatItem = (itemOne,itemTwo)
            if concatItem[0] != 'ε':
                resultSet.add(concatItem[0])
            else:
                resultSet.add(concatItem[1])
    return resultSet
```

The operation looks like this. Where if we have double epsilon we preserve it otherwise the first or second item will remain in the set depending on if the first item is epsilon or not.

```
def constructFollow(self):
    for nonTerminal in self.grammar.getNonTerminals():
        if nonTerminal == self.grammar.getStartingSymbol()[0]:
            self.followSet[nonTerminal] = set("ε")
        else:
            self.followSet[nonTerminal] = set()

    modified = True
    while modified:
        modified = False
        for nonTerminal in self.grammar.getNonTerminals():
            for lhs, rhs in self.grammar.getProductions().items():
                for production in rhs:
                    elements = production.split()
                    if nonTerminal in elements:
                        index = elements.index(nonTerminal)
                        temp = elements[index + 1:]
                        if len(temp) == 0:
                            if not self.followSet[lhs].issubset(self.followSet[nonTerminal]):
                                self.followSet[nonTerminal] = self.followSet[nonTerminal].union(self.followSet[lhs])
                                modified = True
                        else:
                            if "ε" in self.firstSet[temp[0]]:
                                temporarySet = copy.deepcopy(self.firstSet[temp[0]])
                                temporarySet.remove('ε')
                                temporarySet = temporarySet.union(self.followSet[lhs])

                                if not temporarySet.issubset(self.followSet[nonTerminal]):
                                    self.followSet[nonTerminal] = self.followSet[nonTerminal].union(temporarySet)
                                    modified = True
                            else:
                                if not self.firstSet[temp[0]].issubset(self.followSet[nonTerminal]):
                                    self.followSet[nonTerminal] = self.followSet[nonTerminal].union(self.firstSet[temp[0]])
                                    modified = True
    print(self.followSet)
```

For the follow set creation it is initialized as depicted in the course with the starting symbol having epsilon then the loops represent the iteration and the modified behaves similarly as in the first set. When finding the item on the right hand side we get what is following it and depending on if the first contains epsilon (or it is an empty list which is the same) we will add the first of the that part or the follow of the left hand side