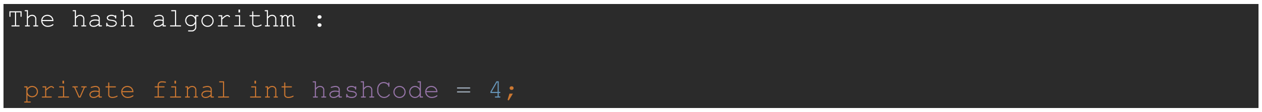
https://github.com/PopEmanuel/FLCD

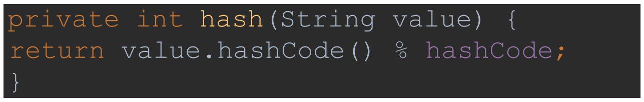
I created a scanner for verifying the lexical correctness of my program. It uses regexPatterns to find what category each character belongs to.

I created a SymbolTable for holding the indentifiers using a custom HashTable



```
The hash algorithm :

 private final int hashCode = 4;
```



```
private int hash(String value) {
return value.hashCode() % hashCode;
}
```

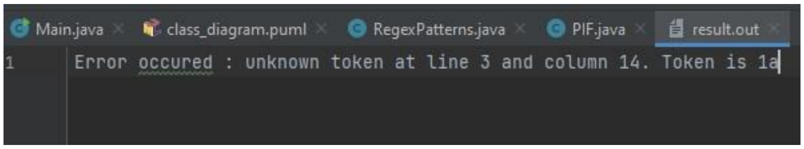
We always work with strings so we use the hashCode of the string and use the modulo with a hashCode held in the class

The way the HashTable works is when you want to add a new value, it computes the hash value of the string, verifies if it is the first one with that index and if so adds it to index 1 otherwise it searches on the index to find the value, if it doesn’t find it it will insert on the last index the value.

Getting the index of a value is easy, just searching on the index and returning the key value pair. Otherwise return (-1, -1)

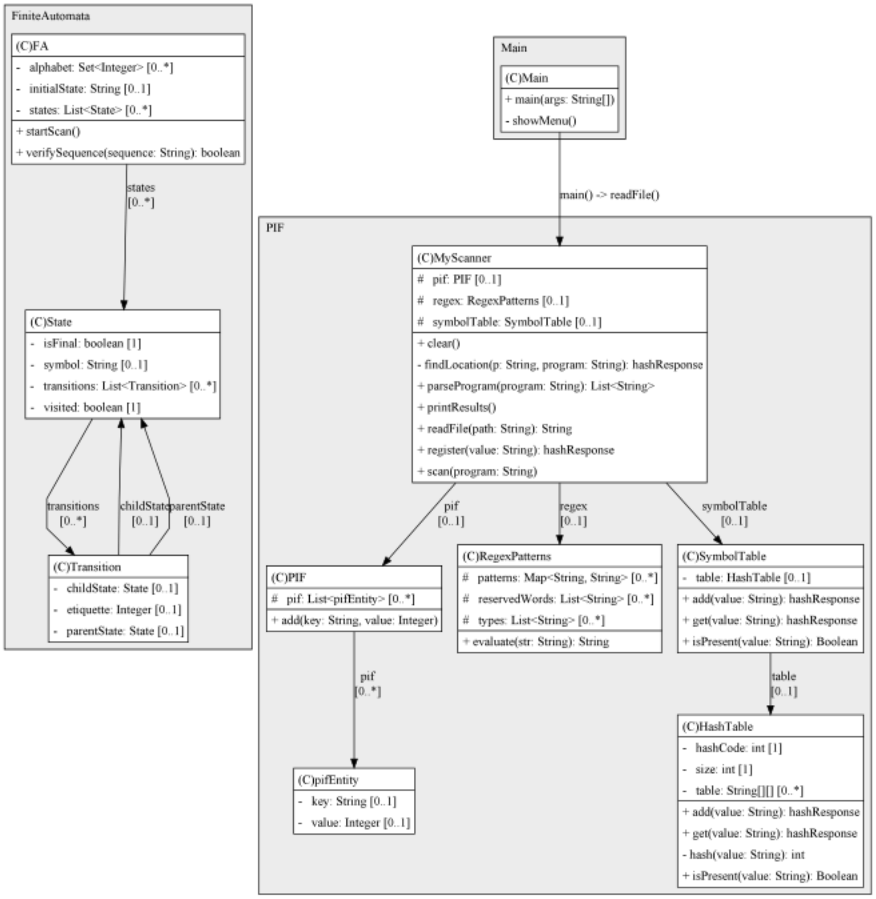
I also created a PIF using an ArrayList from java. I add pifEntities to the List, in order to keep track of each identifier.

The scanner discovers the errors and provides the line and column that it happened.



```
Error occured : unknown token at line 3 and column 14. Token is 1a
```

The diagram class provides more info about the structure of the program


FiniteAutomata
(C)FA
- alphabet: Set<Integer> [0..*]
- initialState: String [0..1]
- states: List<State> [0..*]
+ startScan()
+ verifySequence(sequence: String): boolean
states
[0..*]
(C)State
- isFinal: boolean [1]
- symbol: String [0..1]
- transitions: List<Transition> [0..*]
- visited: boolean [1]
transitions
[0..*]
childState
[0..1]
parentState
[0..1]
(C)Transition
- childState: State [0..1]
- etiquette: Integer [0..1]
- parentState: State [0..1]
Main
(C)Main
+ main(args: String[])
- showMenu()
main() -> readFile()
PIF
(C)MyScanner
# pif: PIF [0..1]
# regex: RegexPatterns [0..1]
# symbolTable: SymbolTable [0..1]
+ clear()
- findLocation(p: String, program: String): hashResponse
+ parseProgram(program: String): List<String>
+ printResults()
+ readFile(path: String): String
+ register(value: String): hashResponse
+ scan(program: String)
pif
[0..1]
regex
[0..1]
symbolTable
[0..1]
(C)PIF
# pif: List<pifEntity> [0..*]
+ add(key: String, value: Integer)
(C)RegexPatterns
# patterns: Map<String, String> [0..*]
# reservedWords: List<String> [0..*]
# types: List<String> [0..*]
+ evaluate(str: String): String
(C)SymbolTable
- table: HashTable [0..1]
+ add(value: String): hashResponse
+ get(value: String): hashResponse
+ isPresent(value: String): Boolean
pif
[0..*]
table
[0..1]
(C)pifEntity
- key: String [0..1]
- value: Integer [0..1]
(C)HashTable
- hashCode: int [1]
- size: int [1]
- table: String[][] [0..*]
+ add(value: String): hashResponse
+ get(value: String): hashResponse
- hash(value: String): int
+ isPresent(value: String): Boolean

I created a Finite Automata using 3 classes, Transition, State and the Automata itself and the application now shows a menu with different options for it.

The structure of the FA.in file should be :

```
FA := initial_state number_of_states {state}(number_of_states)
number_of_transitions {transition}(number_of_transitions)

transition := letter" "alphabet" "letter
alphabet := number
number_of_transitions := number
number_of_states := number
initial_state := state
state := letter" "type
type := n f         (not-final final)
letter := a|b|c|...|z|A|B|...|Z

digit = 0 | 1 |...| 9
nonzero-digit = 1 | 2 | ... | 9
number:=0|"-"no|no
no:=nonzero-digit{no2}
no2:=digit|{no2}

One example is  :
P              //initial state
3              //number of states
p n            //states
q f
r f
4              //number of transitions
p 0 q          //transitions
p 1 r
q 1 q
q 0 r
```