Requirement: Write a program that:

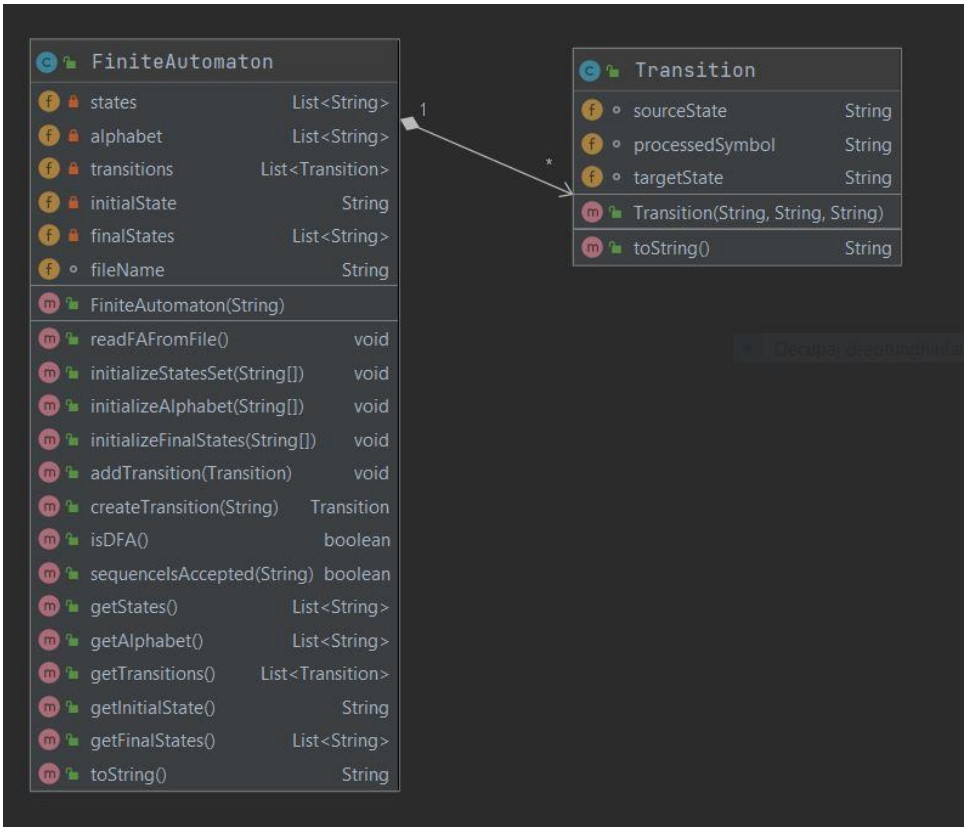
1. Reads the elements of a FA (from file)

2. Displays the elements of a finite automata, using a menu: the set of states, the alphabet, all the transitions, the set of final states.

3. For a DFA, verify if a sequence is accepted by the FA.

Use FA to detect tokens <identifier> and <integer constant> in the scanner program.

Analysis and Design:



FiniteAutomaton - the class representing the finite automaton.

- Attributes:
  - states: a list of strings, representing all the states of the automaton
  - alphabet: a list of strings, representing the alphabet of the automaton (the set of symbols)
  - transitions: a list of Transitions, representing the transitions of the automaton

  > Transition class contains a source state(string), a target state(string) and the symbol (string) that is processed in order to reach the target state from the source state.

  - finalStates: a list of strings, representing the final states.
  - initialState: a string, representing the initial state
  - fileName: a string, representing the name of the file in which the automaton is represented.

- Methods:
  - readFAFromFile() - scans the file in which the automaton is represented and finds the FA's attributes.

  > The representation of the FA in the file is the following:

fa ::== list_of_states "\n" alphabet "\n" initial_state "\n" list_of_final_states "\n" list_of_transitions

list_of_states ::== state | state " " list_of_states

alphabet ::== symbol | symbol " " alphabet

initial_state ::== state

list_of_final_states ::== state | state " " list_of_final_states

list_of_transitions ::== transition | transition "\n" list_of_transitions

transition ::== state " " symbol " " state

state ::== letter | letter {(letter|digit|underscore)}

symbol ::== letter | digit | underscore | "blank" | "?" | "!" | "#" | "*" | "."

letter ::== "A" | "B" | ... | "Z" | "a" | "b" | ... | "z"

digit ::== "0" | "1" | "2" | ... | "9"

underscore :== "_"

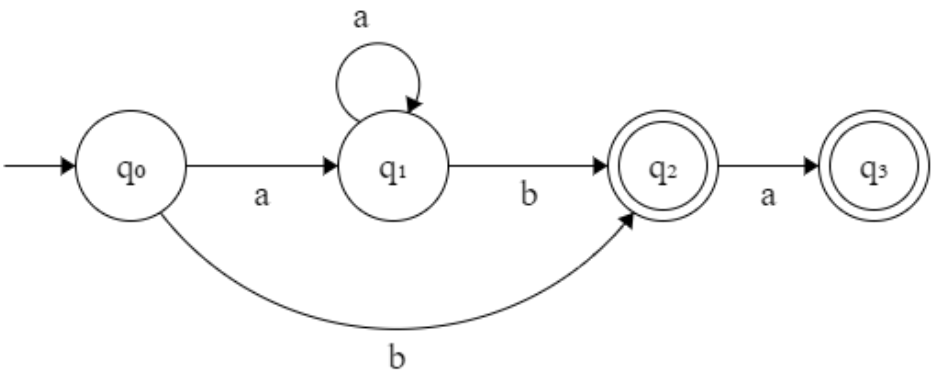
- initializeStatesSet(String[] states)
  - Pre: states = the list of states, read from file
  - Post: fa.states = states
- initializeAlphabet(String[] alphabet)
  - Pre: alphabet = the list of symbols forming the alphabet, read from file
  - Post: fa.alphabet = alphabet
- initializeFinalStates(String[] finalStates)
  - Pre: finalStates = the list of final states, read from file
  - Post: fa.finalStates = finalStates
- addTransition(Transition transition)
  - Pre: transition = a transition of the fa, read from file
  - Post: fa.transitions = fa.transitions U {transition}
- createTransition(String line)
  - Pre: line = a line from the file, containing the elements of a transition (sourceState, processedSymbol, targetState)
  - Post: Transition t, whose elements correspond to the elements from line.
- isDFA()
  - Post: true, if the finite automaton is deterministic; false, otherwise.
- sequenceIsAccepted(String sequence):
  - Pre: sequence = the sequence to be checked
  - Post: false if the FA is non-deterministic or the sequence is not accepted by the FA; true otherwise.

Implementation:

https://github.com/LaviniaGalan/FLCD/tree/master/Lab4

Testing:

1) For the DFA:



Represented in file as follows:

```
1    q0 q1 q2 q3
2    a b
3    q0
4    q2 q3
5    q0 a q1
6    q1 a q1
7    q1 b q2
8    q0 b q2
9    q2 a q3
```

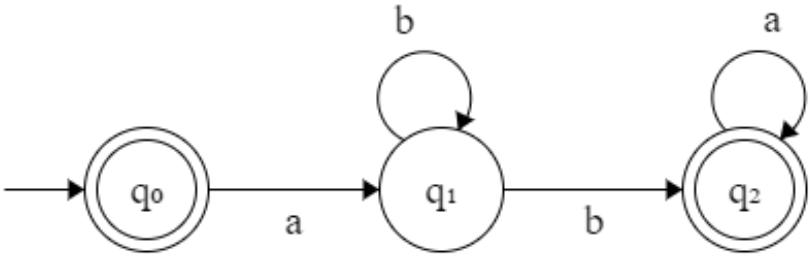
The results are:

```
0. Exit.
1. Show all states.
2. Show alphabet.
3. Show transitions.
4. Show initial state.
5. Show final states.
6. Check if the FA is deterministic.
7. Check if a token is accepted by FA.
Enter your command: >>
6
The FA is deterministic.
```

```
Enter your command: >>
7
Enter the sequence:
abba
Sequence NOT accepted!
```

```
Enter your command: >>
7
Enter the sequence:
aaaab
Sequence accepted!
```

2) For the NFA:



Represented in file as:

```
1   q0 q1 q2
2   a b
3   q0
4   q0 q2
5   q0 a q1
6   q1 b q1
7   q1 b q2
8   q2 a q2
```

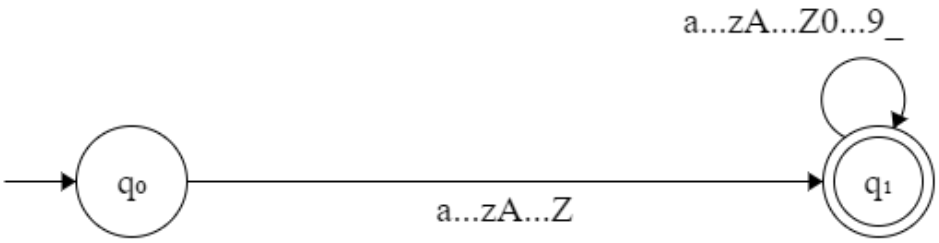
The results are:

```
Enter your command: >>
6
The FA is NOT deterministic.
```

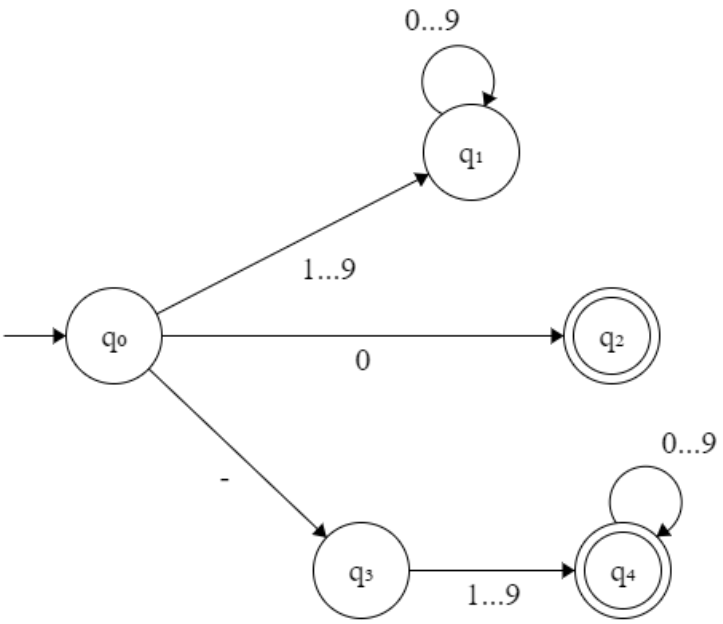
```
Enter your command: >>
7
Enter the sequence:
abba
Not DFA.
```

Integration in Scanner

To detect identifiers, I created the following FA:



To detect the numerical constants, I used the following FA:



I created a new type of classifier, represented by the class ClassifierWithFA. It has 2 finite automata as attributes – one for identifiers and one for numerical constants.

The method that checks if a token is identifier or numeric constant uses these FA: if the token is accepted by the FA for identifiers, then it is an identifier. If the token is accepted by the FA for numerical constants, then it is a numerical constant.