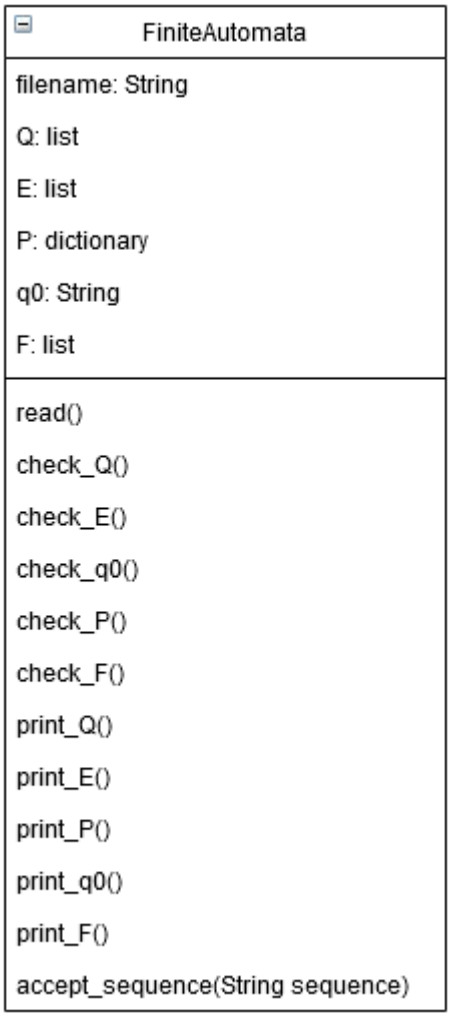
LAB 4

Finite automata implementation



We initialize a FiniteAutomata object which will automatically read its values from the file fa.in, shown below.

```
Q = { p, q, r }
E = { 0, 1 }
q0 = p
F = { p, r }
P = {
    (p, 0) -> q,
    (p, 1) -> p,
    (q, 0) -> r,
    (q, 1) -> p,
    (r, 0) -> r,
    (r, 1) -> r
}
```

- Q will be the set of states
- E will be the alphabet
- Q0 will be the initial state
- F will be the set of final states

- P will be the list (represented as a dictionary) of transitions. The left hand side will be represented by a tuple composed of the starting state and the route it will take, while the right hand side will be represented by the final state of that particular route.

In our check methods, we will check:

- That Q, E and F are sets (they have unique values)
- That q0 is just a string, and not a list or set or anything else
- That P has transitions according to existing states and routes in Q, respectively E

The method accept_sequence will check if, for a DFA, our FA will accept a given sequence. We check if our FA is a DFA. Then we enter the sequence by hand, and it should be composed of only routes that exist in E. If they do, we keep checking for valid transitions in P and move on with the next character of the sequence, while updating the current state with the new starting state in the valid transition. Note that the current state begins from q0. If we reach the end of the sequence we check whether our current state is in F.

Tests:

For a DFA let's use the FA in fa.in. So for sequences:

1. 01 – the sequence should be accepted
2. 1001 - the sequence should be accepted
3. 0 – the sequence should NOT be accepted

If we change our FA to not be a DFA, for example:

```
Q = { p, q, r }
E = { 0, 1 }
q0 = p
F = { p, r }
P = {
    (p, 0) -> q,
    (p, 0) -> r,
    (p, 1) -> p,
    (q, 0) -> r,
    (q, 1) -> p,
    (r, 0) -> r,
    (r, 1) -> r
}
```

Then any sequence we input will not be accepted since it will not go past the DFA check.