Formal Languages and Compiler Design - Lab4

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.

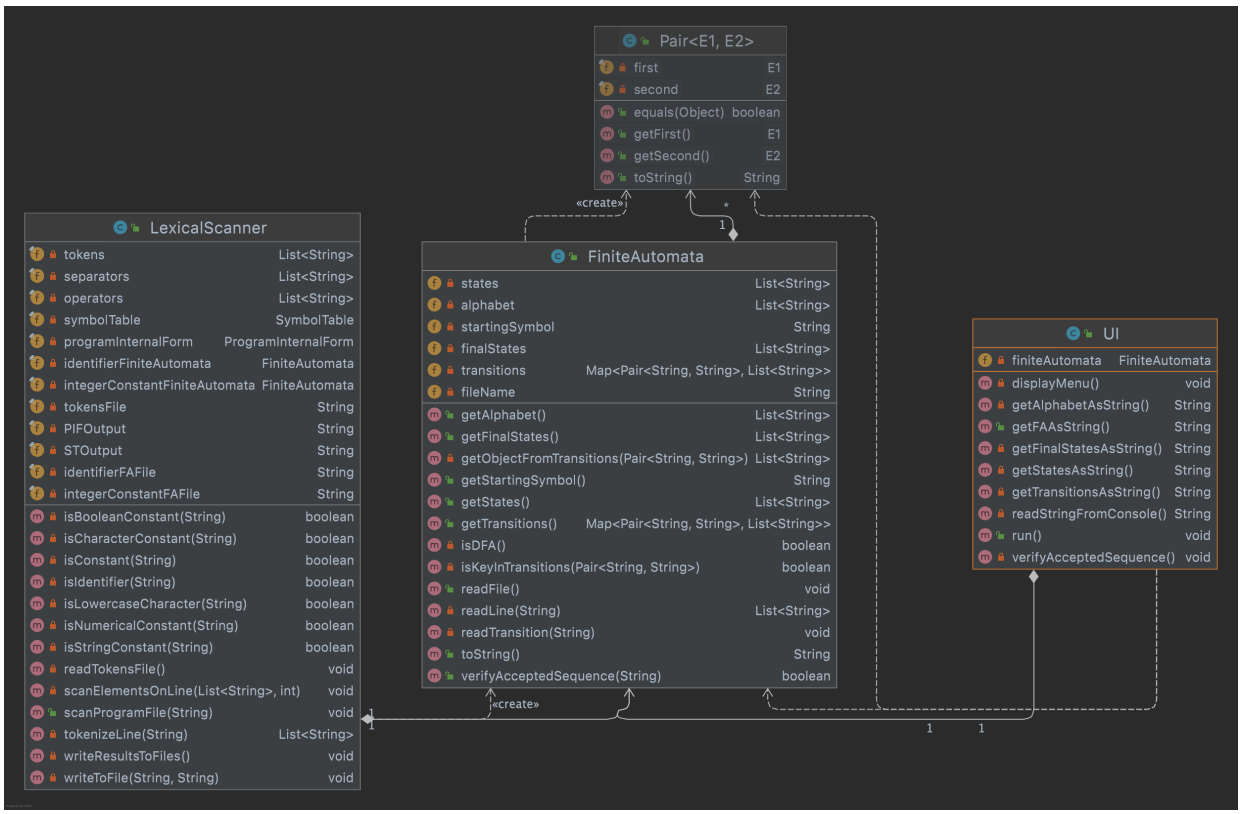
Deliverables:

1. FA.in - input file (*on Github*)
2. Source code (*on Github*)
3. Documentation. It should also include in BNF or EBNF format the form in which the FA.in file should be written (*on Moodle and Github*)

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

Design

Class Diagram



Finite Automata

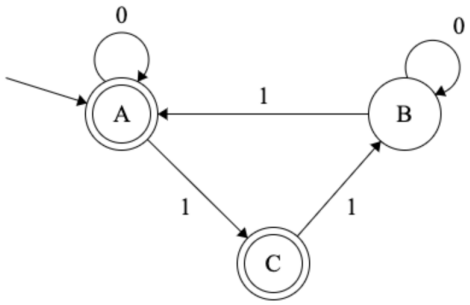
```
public class FiniteAutomata {
    private List<String> states;
    private List<String> alphabet;
    private String startingSymbol;
    private List<String> finalStates;
    private Map<Pair<String, String>, List<String>> transitions;

    private String fileName;
}
```

Methods

- `readFile()`
  - reads a file with the given format
- `readLine(String line)`
  - splits a line from the given file by spaces
  - used for reading the states, alphabet, starting symbol and final states
- `readTransition(String line)`
  - reads a new transition from the file
  - verifies if the pair already exists in the transitions (in the case of nondeterministic finite automata)
    - adds a new element to the associated list of the pair if it exists
    - adds a new pair with an associated list of strings containing the current element
- `isKeyInTransitions(Pair key)`
  - checks if a given pair exists as a key in transitions
- `getObjectFromTransitions(Pair key)`
  - returns the associated list of strings
- `isDFA()`
  - checks if the FA is a DFA
- `verifyAcceptedSequence(String sequence)`
  - checks if a given sequence is accepted by the FA in case of DFA

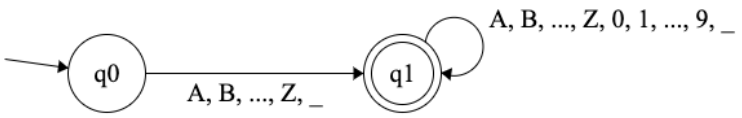
File format

```
A B C // states
0 1 // alphabet
A // starting symbol
A C // final states
A 0 A // transitions
A 1 C
B 0 B
B 1 A
C 1 B
```

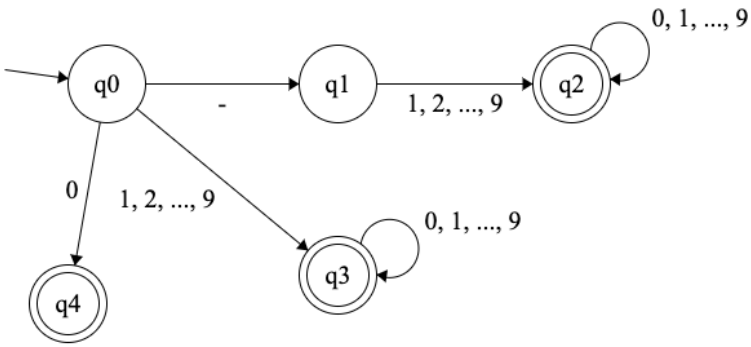


- file lines
  - first line: states separated by space
  - second line: alphabet separated by space
  - third line: starting symbol
  - fourth line: final states separated by space
  - remaining lines: transitions separated by new line
- EBNF format
  - file := states newline alphabet newline startingSymbol newline finalStates newline transitions
    - states := state {state}
      - state := letter{letter | digit}
        - letter := “A” | “B” | ... | “Z” | “a” | “b” | ... | “z”
        - digit := “0” | “1” | ... | “9”
    - alphabet := character {character}
      - character := letter | digit | symbol
        - letter := “A” | “B” | ... | “Z” | “a” | “b” | ... | “z”
        - digit := “0” | “1” | ... | “9”
        - symbol := “-”
    - startingSymbol := letter{letter | digit}
        - letter := “A” | “B” | ... | “Z” | “a” | “b” | ... | “z”
        - digit := “0” | “1” | ... | “9”
    - finalStates := state {state}
      - state := letter{letter | digit}
        - letter := “A” | “B” | ... | “Z” | “a” | “b” | ... | “z”
        - digit := “0” | “1” | ... | “9”
    - transitions := transition newline {transition}
      - transition := state character state

Finite Automata for detecting identifiers



Finite Automata for detecting integer constants

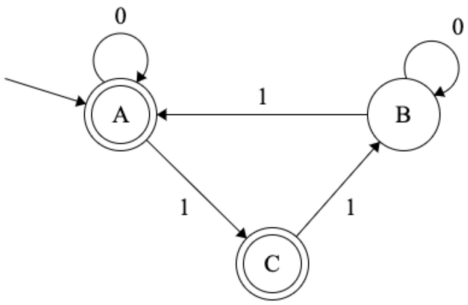


Implementation

Github Link

Tests

Deterministic FA



Input file

```
A B C
0 1
A
A C
A 0 A
A 1 C
B 0 B
B 1 A
C 1 B
```

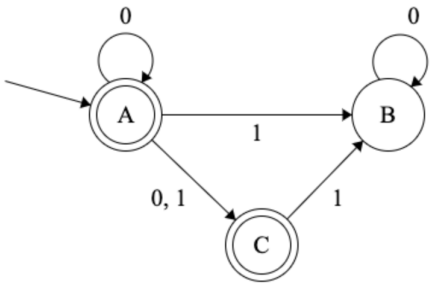
Test

```
        FiniteAutomata finiteAutomata = new FiniteAutomata("src/files/finiteAutomata/F

        System.out.println(finiteAutomata.verifyAcceptedSequence("01"));
        System.out.println(finiteAutomata.verifyAcceptedSequence("011"));
        System.out.println(finiteAutomata.verifyAcceptedSequence("0"));
        System.out.println(finiteAutomata.verifyAcceptedSequence("010"));
        System.out.println(finiteAutomata.verifyAcceptedSequence("0111"));
        System.out.println(finiteAutomata.verifyAcceptedSequence("01111"));

        UI ui = new UI(finiteAutomata);
        System.out.println(ui.getFAAsString());
```

Result

```
true
false
true
false
true
true
States = { A B C }
Alphabet = { 0 1 }
Starting symbol = A
Final states = { A C }
Transitions = {
(A, 0) -> A
(A, 1) -> C
(C, 1) -> B
(B, 0) -> B
(B, 1) -> A
}
```

Nondeterministic FA



Input file

```
A B C
0 1
A
A C
A 0 A
A 0 C
A 1 C
B 0 B
B 1 A
C 1 B
```

Test

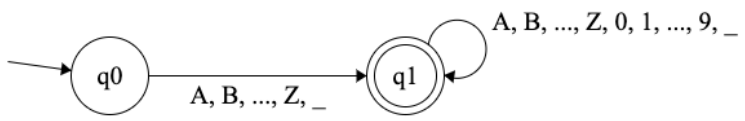
```
        FiniteAutomata finiteAutomata = new FiniteAutomata("src/files/finiteAutomata/M
        System.out.println(finiteAutomata);

        System.out.println(finiteAutomata.verifyAcceptedSequence("01"));
        System.out.println(finiteAutomata.verifyAcceptedSequence("011"));
        System.out.println(finiteAutomata.verifyAcceptedSequence("0"));
        System.out.println(finiteAutomata.verifyAcceptedSequence("010"));
        System.out.println(finiteAutomata.verifyAcceptedSequence("0111"));
        System.out.println(finiteAutomata.verifyAcceptedSequence("01111"));
```

Result

```
false
false
false
false
false
false
States = { A B C }
Alphabet = { 0 1 }
Starting symbol = A
Final states = { A C }
Transitions = {
```

```
(A, 0) -> A
(A, 0) -> C
(A, 1) -> C
(C, 1) -> B
(B, 0) -> B
(B, 1) -> A
}
```

Finite Automata for detecting identifiers



Input file

```
q0 q1
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 _
q0
q1
q0 A q1
q0 B q1
q0 C q1
q0 D q1
q0 E q1
q0 F q1
q0 G q1
q0 H q1
q0 I q1
q0 J q1
q0 K q1
q0 L q1
q0 M q1
q0 N q1
q0 O q1
q0 P q1
q0 Q q1
q0 R q1
q0 S q1
q0 T q1
q0 U q1
q0 V q1
q0 W q1
q0 X q1
q0 Y q1
q0 Z q1
```

```
q0 _ q1
q1 A q1
q1 B q1
q1 C q1
q1 D q1
q1 E q1
q1 F q1
q1 G q1
q1 H q1
q1 I q1
q1 J q1
q1 K q1
q1 L q1
q1 M q1
q1 N q1
q1 O q1
q1 P q1
q1 Q q1
q1 R q1
q1 S q1
q1 T q1
q1 U q1
q1 V q1
q1 W q1
q1 X q1
q1 Y q1
q1 Z q1
q1 _ q1
q1 0 q1
q1 1 q1
q1 2 q1
q1 3 q1
q1 4 q1
q1 5 q1
q1 6 q1
q1 7 q1
q1 8 q1
q1 9 q1
```

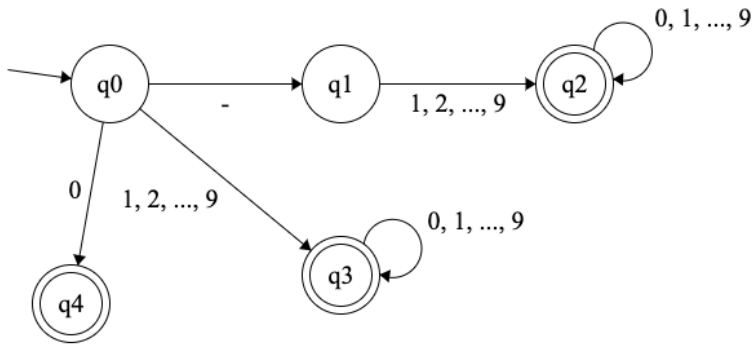
Test

```
        FiniteAutomata identifierFiniteAutomata = new FiniteAutomata(
        "src/files/finiteAutomata/identifierFA.in");

        System.out.println(identifierFiniteAutomata.verifyAcceptedSequence("ABC"));
        System.out.println(identifierFiniteAutomata.verifyAcceptedSequence("A123"));
        System.out.println(identifierFiniteAutomata.verifyAcceptedSequence("0"));
        System.out.println(identifierFiniteAutomata.verifyAcceptedSequence("10BCD"));
        System.out.println(identifierFiniteAutomata.verifyAcceptedSequence("_ABC"));
        System.out.println(identifierFiniteAutomata.verifyAcceptedSequence("_189"));
```

Result

```
true
true
false
false
true
true
```

Finite Automata for detecting integer constants



Input file

```
q0 q1 q2 q3 q4
0 1 2 3 4 5 6 7 8 9 -
q0
q2 q3 q4
q0 - q1
q1 1 q2
q1 2 q2
q1 3 q2
q1 4 q2
q1 5 q2
q1 6 q2
q1 7 q2
q1 8 q2
q1 9 q2
q2 0 q2
q2 1 q2
q2 2 q2
q2 3 q2
q2 4 q2
q2 5 q2
q2 6 q2
q2 7 q2
q2 8 q2
q2 9 q2
q0 1 q3
```

```
q0 2 q3
q0 3 q3
q0 4 q3
q0 5 q3
q0 6 q3
q0 7 q3
q0 8 q3
q0 9 q3
q3 0 q3
q3 1 q3
q3 2 q3
q3 3 q3
q3 4 q3
q3 5 q3
q3 6 q3
q3 7 q3
q3 8 q3
q3 9 q3
q0 0 q4
```

Test

```
        FiniteAutomata integerConstantFiniteAutomata = new FiniteAutomata(
                "src/files/finiteAutomata/integerConstantFA.in");

        System.out.println(integerConstantFiniteAutomata.verifyAcceptedSequence("120")
        System.out.println(integerConstantFiniteAutomata.verifyAcceptedSequence("-5"))
        System.out.println(integerConstantFiniteAutomata.verifyAcceptedSequence("0"));
        System.out.println(integerConstantFiniteAutomata.verifyAcceptedSequence("-0"))
        System.out.println(integerConstantFiniteAutomata.verifyAcceptedSequence("-08")
        System.out.println(integerConstantFiniteAutomata.verifyAcceptedSequence("-100'
        System.out.println(integerConstantFiniteAutomata.verifyAcceptedSequence("-100A
```

Result

```
true
true
true
false
false
true
false
```