FA class https://github.com/gabiborlea/FLCD/blob/master/src/FA.java

Whole Integration: https://github.com/gabiborlea/FLCD

Requirement

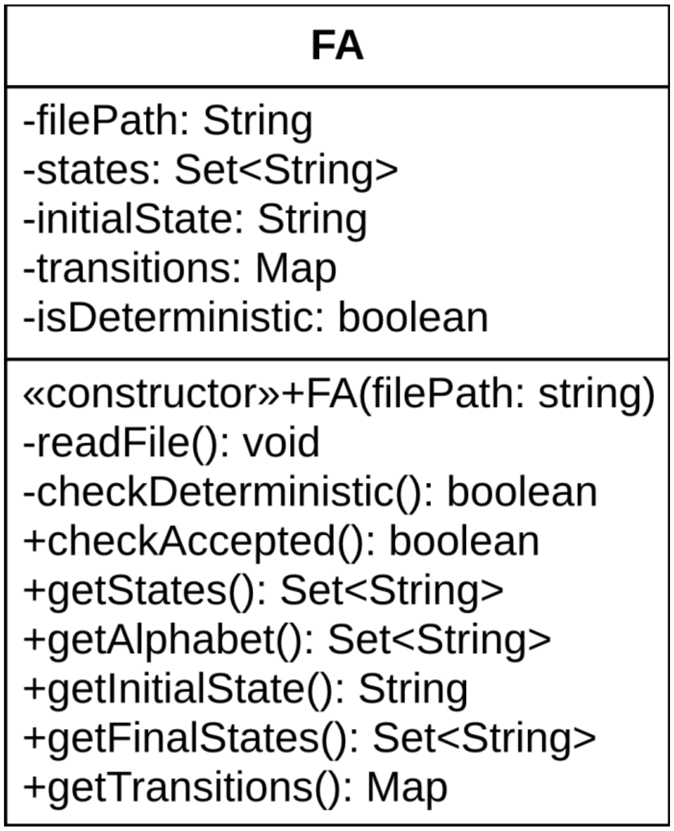
Write a program that:

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.

Structure



```
FA
-filePath: String
-states: Set<String>
-initialState: String
-transitions: Map
-isDeterministic: boolean
«constructor»+FA(filePath: string)
-readFile(): void
-checkDeterministic(): boolean
+checkAccepted(): boolean
+getStates(): Set<String>
+getAlphabet(): Set<String>
+getInitialState(): String
+getFinalStates(): Set<String>
+getTransitions(): Map
```

Methods:

FA(filePath: String)

- params:

- filePath: the path to the file where states, alphabet, initial state, final states and transitions are located.

readFile(): void

- parses the file and fills the attributes of the object

checkAccepted(sequence: String): boolean

- checks if the sequence is accepted
- params:
  - sequence: the sequence to the tested if accepted or not.

Input File (EBNF)

inputFile ::= states newline alphabet newline initialState newline finalStates newline transitions

states ::= state {state}

alphabet ::= symbol {symbol}

initialState ::= state

finalStates ::= state {state}

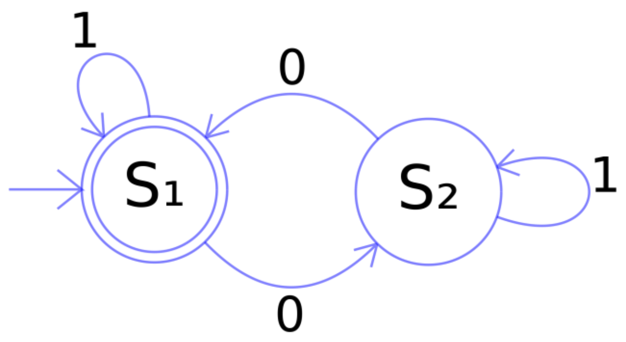
transitions ::= transition newline {transition newline}

tranistion ::= (state, symbol)>state

state ::= character

symbol ::= character

Tests



FA.in

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

```
110110

States: [A, B]

Alphabet: [0, 1]

Initial states: A

Final States: [A]

Transitions: {A=[1=A, 0=B], B=[0=A, 1=B]}

Is deterministic: true

Is accepted: true
```

```
1000

States: [A, B]

Alphabet: [0, 1]

Initial states: A

Final States: [A]

Transitions: {A=[1=A, 0=B], B=[0=A, 1=B]}

Is deterministic: true

Is accepted: false
```

FA_int_constant.in

```
A B C D
+ - 0 1 2 3 4 5 6 7 8 9
A
C D
(A,+)>B
```

```
(A,-)>B
(A,0)>D
(A,1)>C
(A,2)>C
(A,3)>C
(A,4)>C
(A,5)>C
(A,6)>C
(A,7)>C
(A,8)>C
(A,9)>C
(B,1)>C
(B,2)>C
(B,3)>C
(B,4)>C
(B,5)>C
(B,6)>C
(B,7)>C
(B,8)>C
(B,9)>C
(C,0)>C
(C,1)>C
(C,2)>C
(C,3)>C
(C,4)>C
(C,5)>C
(C,6)>C
(C,7)>C
(C,8)>C
(C,9)>C
```

78

```
States: [A, B, C, D]

Alphabet: [+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Initial states: A

Final States: [C, D]

Transitions: {A=[3=C, 2=C, 1=C, 0=D, 7=C, 6=C, 5=C, 4=C, +=B, 9=C, 8=C, -=B], B=[3=C, 2=C, 1=C, 7=C, 6=C, 5=C, 4=C, 9=C, 8=C], C=[3=C, 2=C, 1=C, 0=C, 7=C, 6=C, 5=C, 4=C, 9=C, 8=C]}

Is deterministic: true
Is accepted: true


078

States: [A, B, C, D]

Alphabet: [+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Initial states: A

Final States: [C, D]

Transitions: {A=[3=C, 2=C, 1=C, 0=D, 7=C, 6=C, 5=C, 4=C, +=B, 9=C, 8=C, -=B], B=[3=C, 2=C, 1=C, 7=C, 6=C,
5=C, 4=C, 9=C, 8=C], C=[3=C, 2=C, 1=C, 0=C, 7=C, 6=C, 5=C, 4=C, 9=C, 8=C]}

Is deterministic: true

Is accepted: false
```