**COMP 3002 Winter 2021 Assignment #4**

**Building FiniteStateMachines**

**Due: Tuesday Midnight Feb 23 (After Break Week).**

**Basic goals** To implement a finite state machine. To do that, we need at least the following 4 classes

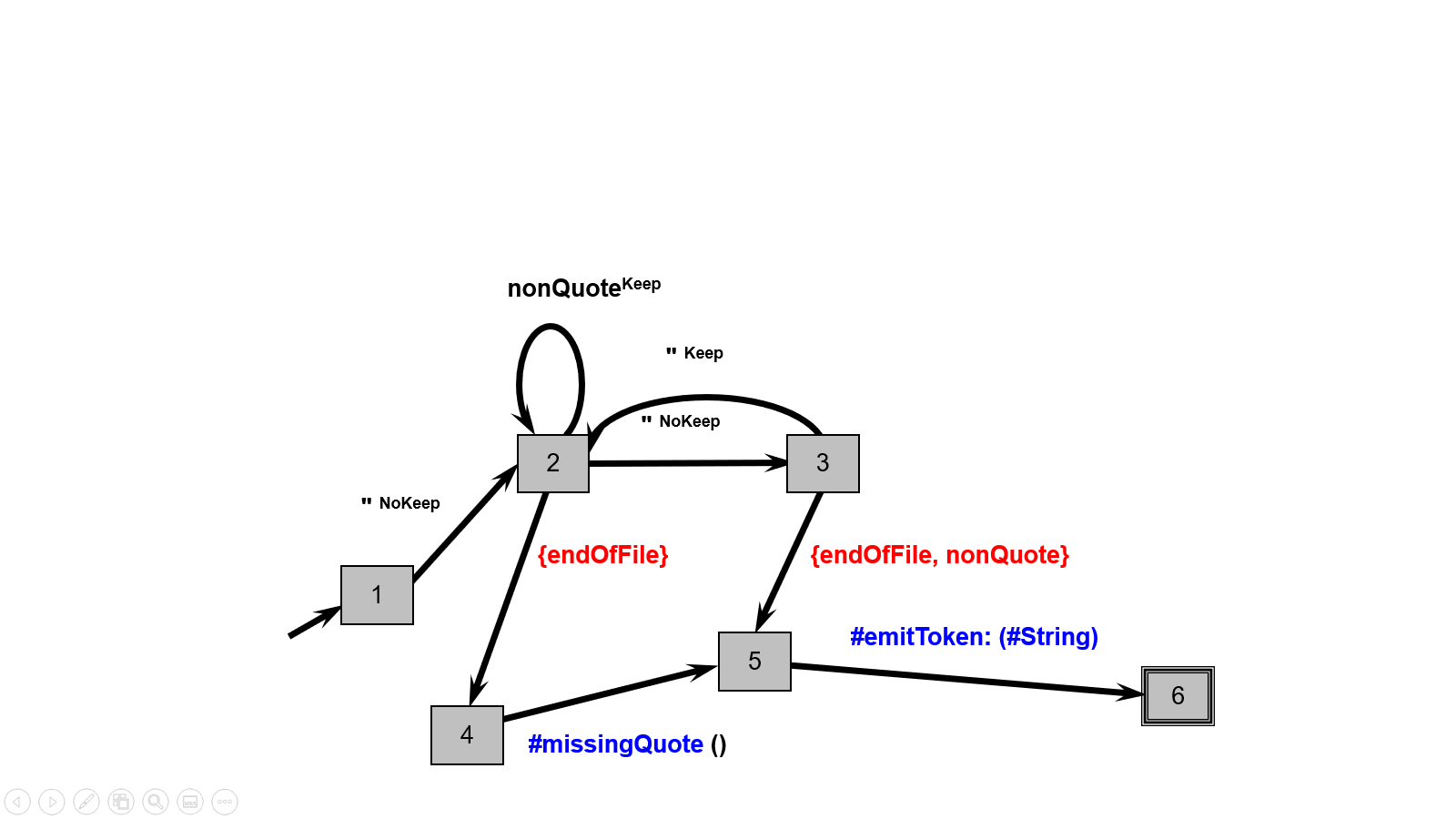
FiniteStateMachine with states

FiniteStateMachineState with stateNumber isInitial isFinal transitions

Transition with transitionName goto

TransitionName with symbolOrPair attributes action parameters isRootBuilding

Note that all the instance variables starting with “is” are meant to be booleans. You don’t know what “isRootBuilding” is all about, so initialize it to true as the default. As for attributes, use either 'RK' (read keep), 'R' (read noKeep), or 'L' (look transitions are those surrounded by braces). Note that this FSM has 1 initial state and 1 final state. FSMs in general can have many.



Then you should be able to construct the FSM above in an example method (but you will have to build the example method yourself). For the transitions, use symbols including #'"' (this is # followed by single quote followed by double quote followed by single quote; i.e., a double quote inside a single quoted string), #a, #b, #endOfFile, #missingQuote, #emitToken: (with a color), and #String.

Page 1 of 2

In addition to building this FSM, you want to be able to print it. So you need to be able to implement a method called “printOn: aStream”. An example FSM output (not the one above) is shown below. Note: there many examples of “printOn:” methods you could look at; e.g., the one in Tree or Token. Note that in the output, the words FSM and END line up, then state is indented in (say 3 spaces), and the transitions are indented in (3 more spaces).

FSM

state: 1; initial; final;

for: #S properties: "RSN" goto: 2;

for: #a properties: "RSN" goto: 5;

state: 2;

for: #a properties: "L" goto: 3;

state: 3;

for: semanticAction: #buildTree: parameters: #("list") goto: 4;

state: 4; final;

state: 5; final;

END

Finally, if you inspect an FSM and you click on self, it should print the above. If you double click, go inside, and look at state 1, when you click on it, it will print just state 1 and its transitions. If you keep inspecting until you get to a specific transition, when you click on it, it will print just that transition. This suggests that an FSM’s printOn: method should make use of a state’s printOn: method which in turn should make use of a transition’s printOn: method.

Note: When you say something like one of the following

Transcript << aTransition

aStream << aTransition

that causes the printOn: method to be used automatically. Try it.