Andreas Slovacek

Fall 2018, CS5700

**Project 03 Report**

**Forewarning:** I errored along the way and ran out of time. The error is in red. The following is the approach.

**Technical Approach**

Create one FA to identify tokens versus comments; state diagram in Figure 1. This machine reads character-by-character, stores the tokens, and discards the comments. This machine is implemented in *JackProcessor*.

A second processes each token in the using “max munch.” See Figure 2 for the state diagram of that machine. This machine processes tokens in the *NFA*; but has states, alphabet, and transitions defined in *FiniteAutomaton*.

**Class Descriptions**

*FiniteAutomaton*:

* Base class for NFA (DFA not implemented)
* Implements: states, alphabet, transitions, start state, accept states
* addTransition function failed here: epsilon transitions out of start state didn’t stay in the transition map. Couldn’t find the commit where they did.

*NFA*:

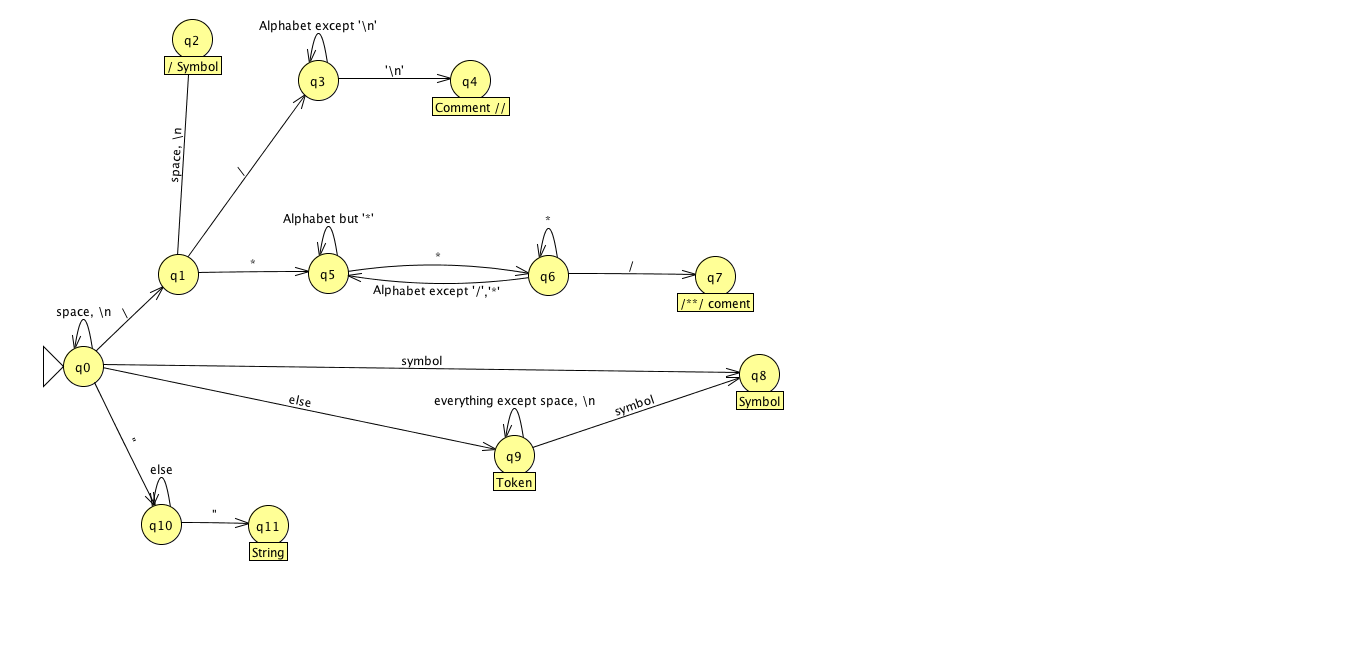
* Wrapper for FiniteAutomaton
* Implements epsilon transitions with a recursive algorithm ([here](https://condor.depaul.edu/ichu/csc416/notes/notes3/nfa/nfa.html))

*CompilationEngine*:

* Called from main, should have moved main call into *CompilationEngine*
* create *JackParser* to extract input strings from the jack files
* create *Simulator* to simulate transitions through the state machine
* create *Tokenizer* to write the tokens to .tok file
* create *Logger* to write the tokens to .log file

*JackProcessor*

* Process the input Jack file per the following state diagram



Figure

*Simulator*

* Simulate an NFA processing each token from *JackProcessor*
* Store the output for *Tokenizer*

**Source Details**

OS: OSX 10.13.4

Environment: Eclipse Neon (4.6.3)

JRE: 1.7

Maven

SCM: Github (<https://github.com/Andreas237/Jack_Lexer>)

