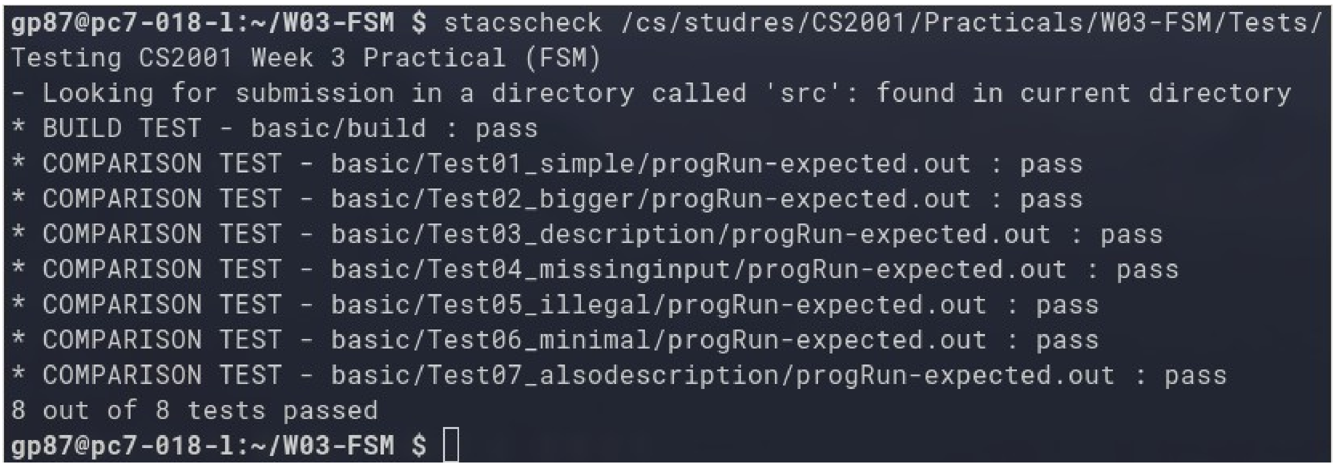
Practical W03-FSM Student ID: 200007413

Testing

Test 1: Stacscheck

Expected Output: pass all 8 tests
Actual Output: passed all 8 tests

Screenshot:

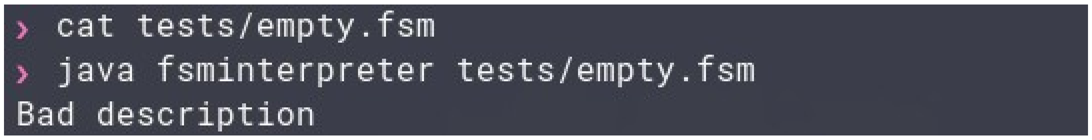


```
gp87@pc7-018-l:~/W03-FSM $ stacscheck /cs/studres/CS2001/Practicals/W03-FSM/Tests/
Testing CS2001 Week 3 Practical (FSM)
- Looking for submission in a directory called 'src': found in current directory
* BUILD TEST - basic/build : pass
* COMPARISON TEST - basic/Test01_simple/progRun-expected.out : pass
* COMPARISON TEST - basic/Test02_bigger/progRun-expected.out : pass
* COMPARISON TEST - basic/Test03_description/progRun-expected.out : pass
* COMPARISON TEST - basic/Test04_missinginput/progRun-expected.out : pass
* COMPARISON TEST - basic/Test05_illegal/progRun-expected.out : pass
* COMPARISON TEST - basic/Test06_minimal/progRun-expected.out : pass
* COMPARISON TEST - basic/Test07_alsodescription/progRun-expected.out : pass
8 out of 8 tests passed
gp87@pc7-018-l:~/W03-FSM $
```

Test 2: Empty FSM Description File

Expected Output: “Bad description” error message
Actual Output: “Bad description” error message

Screenshot:

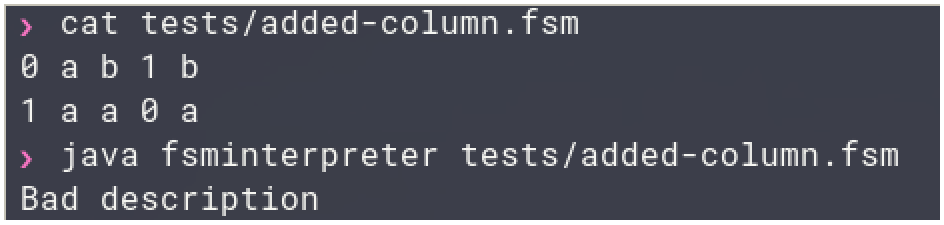


```
› cat tests/empty.fsm
› java fsminterpreter tests/empty.fsm
Bad description
```

Test 3: Added Column to FSM Description File

Expected Output: “Bad description” error message
Actual Output: “Bad description” error message

Screenshot:

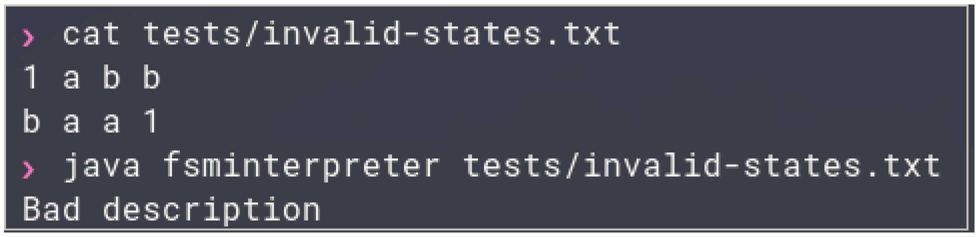


```
› cat tests/added-column.fsm
0 a b 1 b
1 a a 0 a
› java fsminterpreter tests/added-column.fsm
Bad description
```

Test 4: Non-Integer States

Expected Output: “Bad description” error message
Actual Output: “Bad description” error message

Screenshot:

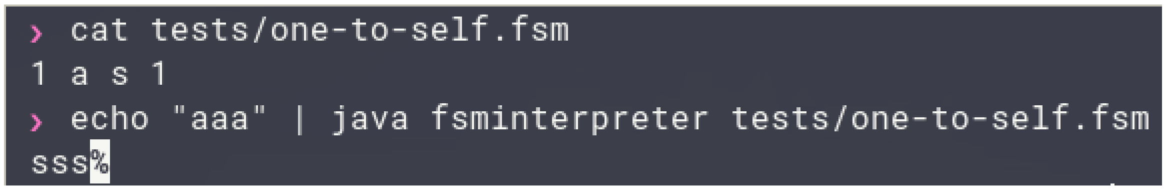


```
› cat tests/invalid-states.txt
1 a b b
b a a 1
› java fsminterpreter tests/invalid-states.txt
Bad description
```

Test 5: One State Pointing to Itself

Expected Output: Prints out ‘a’ as many times as the valid input is entered
Actual Output: Prints out ‘a’ as many times as the valid input was entered

Screenshot:

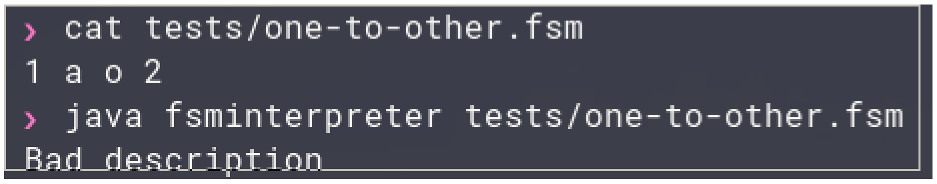


```
› cat tests/one-to-self.fsm
1 a s 1
› echo "aaa" | java fsminterpreter tests/one-to-self.fsm
sss%
```

Test 6: One State Pointing to a Different State

Expected Output: “Bad description” error message
Actual Output: “Bad description” error message

Screenshot:



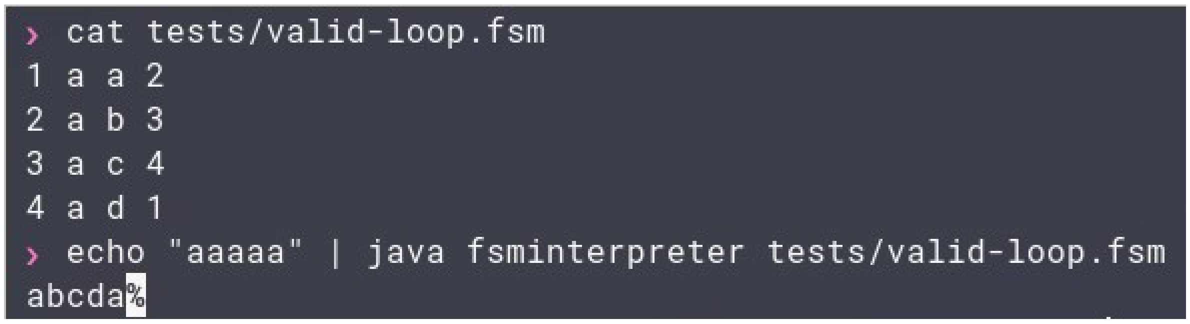
```
› cat tests/one-to-other.fsm
1 a o 2
› java fsminterpreter tests/one-to-other.fsm
Bad description
```

Test 7: Ring State Loop

Expected Output: Moves up one state at a time until it loops back to the beginning

Actual Output: Looped round each state, printing correctly

Screenshot:

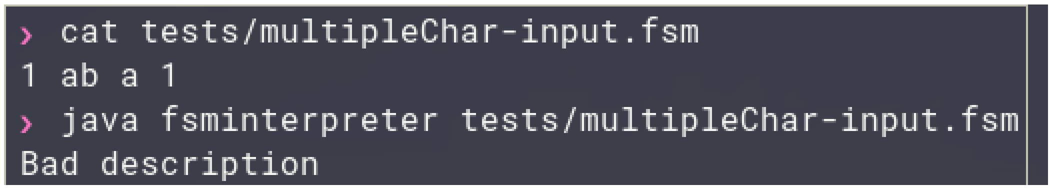


```
› cat tests/valid-loop.fsm
1 a a 2
2 a b 3
3 a c 4
4 a d 1
› echo "aaaaa" | java fsminterpreter tests/valid-loop.fsm
abcda%
```

Test 8: Multiple Character Input Strings

Expected Output: “Bad Description” error message
Actual Output: “Bad Description” error message

Screenshot:

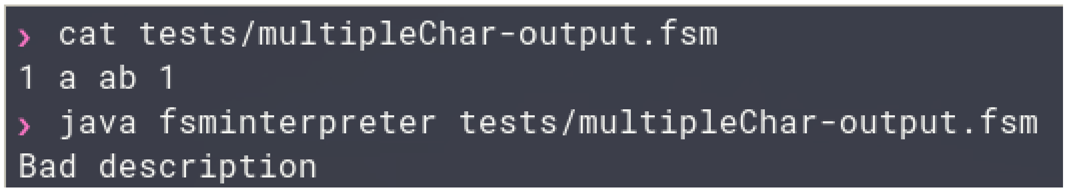


```
› cat tests/multipleChar-input.fsm
1 ab a 1
› java fsminterpreter tests/multipleChar-input.fsm
Bad description
```

Test 9: Multiple Character Output Strings

Expected Output: “Bad Description” error message
Actual Output: “Bad Description” error message

Screenshot:

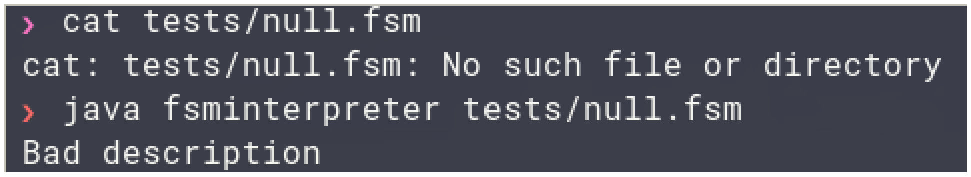


```
› cat tests/multipleChar-output.fsm
1 a ab 1
› java fsminterpreter tests/multipleChar-output.fsm
Bad description
```

Test 10: FSM Description File Doesn’t Exist

Expected Output: “Bad Description” error message
Actual Output: “Bad Description” error message

Screenshot:

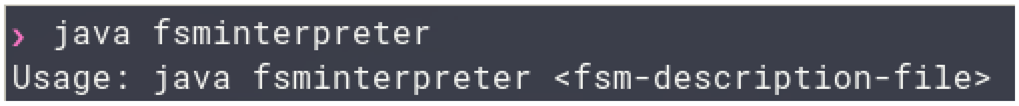


```
› cat tests/null.fsm
cat: tests/null.fsm: No such file or directory
› java fsminterpreter tests/null.fsm
Bad description
```

Test 11: No FSM File Given

Expected Output: Usage message displayed
Actual Output: Usage message displayed

Screenshot:

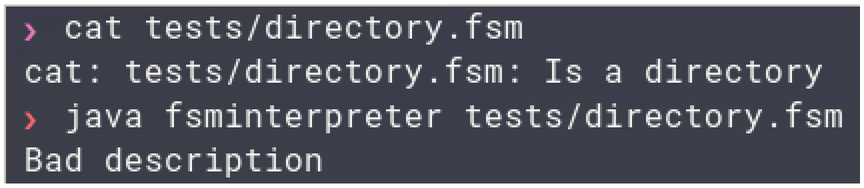


```
› java fsminterpreter
Usage: java fsminterpreter <fsm-description-file>
```

Test 12:
FSM Descriptor file given is a directory

Expected Output: “Bad Description” error message
Actual Output: “Bad Description” error message

Screenshot:



```
› cat tests/directory.fsm
cat: tests/directory.fsm: Is a directory
› java fsminterpreter tests/directory.fsm
Bad description
```

<u>Design</u>

My first design decision was to setup the fsminterpreter class to have the main() function pass the fsm description file argument into a constuctor for fsm instance. My reasoning for this was to allow the class to be extensible, as having all of the error handling of the many potential errors which can be thrown due to bad description files would mean that this error handling would need to be repeated everywhere an fsm was setup.

Having moved the fsm setup into a constructor for an instance allowed me to add instance attributes which would be unique for each fsm. This allows you to interpret many fsm’s without the attributes overwriting eachother and causing issues.

The way I removed the need to have excessive error handling in the main() function was to have the errors dealt with in the constructor. I created the BadDesciptionException, and BadInputException classes which allowed me to determine what was the cause of the error, and then throw those out of the constructor and run() public methods. I limited it to these because the specification only required the program to distinguish the cause of the error to be due to the description, or input, and makes handling exceptions when creating fsm’s much simpler.

I also created a State class, which would hold the states transitions. This is because for each state, there can be many transitions, which are a mapping of inputs to outputs and a next_state value. The data structures, if not a new class, would have been very complex and very hard to understand in terms of reading the code later.

Practical W03-FSM Student ID: 200007413

I added a valid_inputs attribute into the fsminterpreter class which fills up in the constructor. The reason for this is I wanted to make sure that each of the States had transitions for all potential inputs. This is shown in the Test04-Missing-Inputs stacscheck test, as all of the transitions in the fsm description file are legal, but for state 3, there is is no transition for the valid input of “b”.

I decided to put the valid_inputs attribute in the fsminterpreter class, instead of the State class, because for each FSM, the valid inputs remain the same. With many states, this attribute would have been needlessly copied into each State instance, and so this saves memory.

In the State class, I added two separate HashMap attibutes for each transition. This was because I wanted to have separate functionality, as the run() method needs to get the output and next_state at different times. This meant that I could add a getNextState() method, which I could then try to retrieve from the states attribute, and then call the getOutput() method, instead of having to separate the values in the run() method. Again, this meant that the State class is more extensible, as it is far easier to undestand what getNextState() and getOutput() are returning, rather than a getTransition() which would return an odd object holding an integer, and character pair.

Next, I decided to have a similar approach with the running of the FSM as to the creation. I added a run() method which took a single character input which would run on the instance level. This meant that I could look at the valid_input attribute and check whether the input was valid and immediately throw the BadInputException if not.