Implement a program that permits definition, creation and execution of finite state machine.

A state machine definition has the following

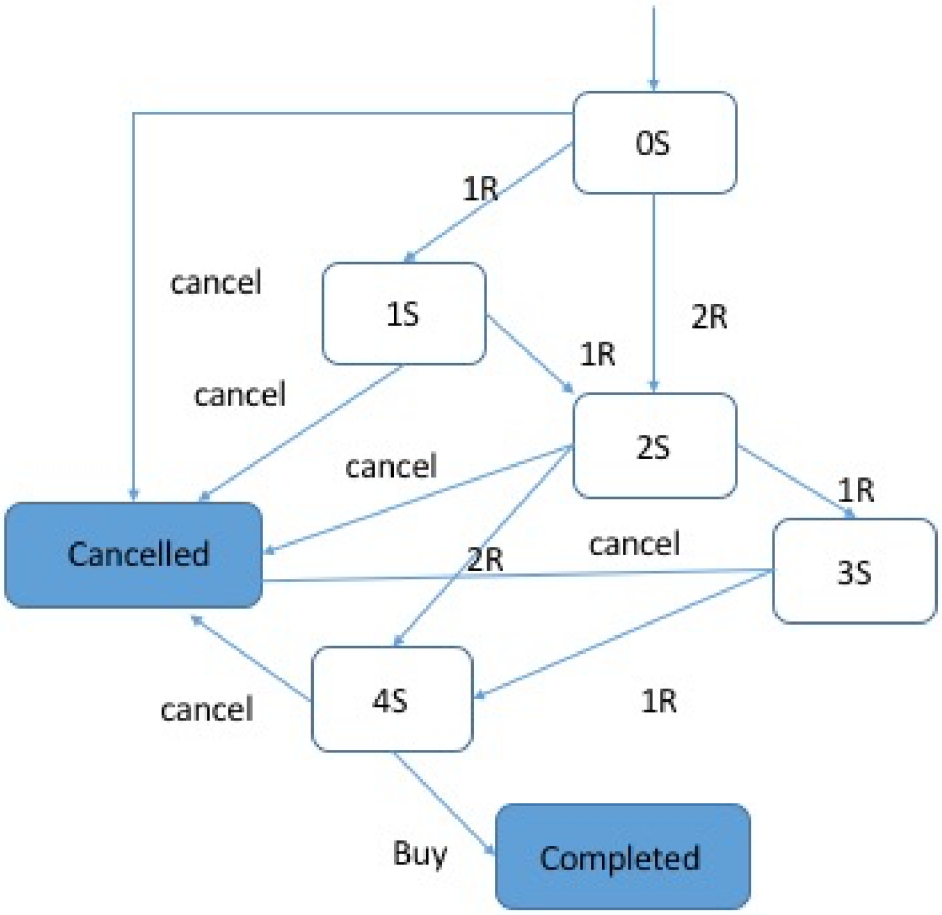
- Finite set of states
- One or more initial states
- One or more terminal states
- One or more events
- State transitions for events (explained in detail below)

Here is an example of a cash counting state machine. The machine will allow a purchase if and only the user deposits 4 rupees.Note that the lines starting with "//" are comments for your understanding

```
//states
0S
1S
2S
3S
4S
CANCELLED
COMPLETED


//initial state
0S

//terminal states
CANCELLED
COMPLETED

//events
1R
2R
CANCEL
BUY

// state transitions

(0S, 1R) 1S
(0S, 2R) 2S
(1S, 1R) 2S
```

(1S, 2R) 3S
(2S, 1R) 3S
(2S, 2R) 4S
(3S, 1R) 4S
(*, CANCEL) CANCELLED
(4S, COMPLETE) COMPLETED

The above input results in a state machine as follows



The next set of inputs/outputs show the creation and manipulation of a state machine i.e. an instance of the previously defined state machine is created and manipulated. Here's a sample interaction. Lines starting with '>' are inputs to the program, while lines starting with '<' are outputs from the program

>0S
<0S
>1R

<1S
>2R
<3S
>1R
<4S
>BUY
<!COMPLETED //! Indicates terminal state

Your program is expected to validate inputs for the state machine definition, which include the following scenarios

- Invalid states and events
- Unreachable terminal states
- Intermediate states which cannot lead to a terminal state

Subsequently an instance of a state machine is created, and events are received and the resulting state after processing each event is printed.

In case of unsupported (state, event) combinations such as (3S, 2R) no state transition should be performed and suitable error message should be printed.

Your program should be written according to Object Oriented Programming principles and you should demonstrate the functioning of your code with unit and functional test cases.