**Exercise 1**

1. New, Runnable, Running, Not Runnable, Terminated.
2. By declaring all public methods as synchronized, we can turn the java object into a monitor.
3. Java doesn’t support multiple inheritance, because of that if you need to inherit from another class then you can implement Runnable interface.
4. HELLOPS = ( hello -> HELLOPS | stop -> STOP ).

**Exercise 2**

1. If application is multithreaded, then two threads may try to initialize Singleton instance. One thread can create an instance of Singleton, but at the same time another thread can pass through if instance is initialized control before other thread updates instance variable. So two instances can be created and this violates safety and cause problems.
2. We can turn this Singleton to thread-safe singleton by adding synchronized keyword to getInstance method.
3. Yes, this implementation can introduce a bottleneck because synchronized keyword locks threads and enable them to reach critical section one by one. To overcome bottleneck, remove synchronized keyword from getInstance method. Remove if control and initialization from getInstance method. getInstance method will only contain return instance line. Lastly, we will change null assignment to instance. Instead of private static Singleton instance = null; , we will change this as private static Singleton instance = new Singleton(); . In Java, static initializers are thread-safe. This change will make Java to use static initializer. This changes will remove this bottleneck.

**Exercise 3**

* APPOINTMENT = ( hello -> converse -> goodbye -> STOP ).
* HOLIDAY = ( arrive -> relax -> leave -> HOLIDAY ).
* SPEED = (on -> ON ),

ON = (speed -> ON | off -> SPEED).

* LEFTONCE = (ahead -> AHEAD ),

AHEAD = (left -> STOP | right -> LEFTONCE).

* TREBLE = (in[1] -> out[3] -> TREBLE | in[2] -> out[6] -> TREBLE | in[3] -> out[9] -> TREBLE).
* FIVETICK (N=5) = FIVETICK[0],

FIVETICK[i:0..N] =

( when(i<N) tick->FIVETICK[i+1]).

* PERSON = (workday -> sleep -> work -> PERSON | holiday -> sleep -> {play, shop} -> PERSON).

**Exercise 4**

1. If number of steps is 5 then we will have 6 possible state for each competitor. Cartesian product of both is 6 \* 6 = 36 total states. For 1 competitor, we have 6 states and 5 steps that means 6 \* 5 = 30 possible transitions, but we have 2 competitors then 2 \* 30 = 60 transitions.
2. For n steps -> possible states for one process is n + 1 , cartesian product of two processes with same steps is n + 1 \* n + 1.   
   For n steps -> possible states for one process is n + 1, one process will have n + 1 \* n transitions. For both processes, it’s 2 \* ((n + 1) \* n).
3. Using formulas above,  
   For 1 step -> 4 states, 4 transitions

Result from LTS tool:

Composition:

Race5K = tortoise:Competitor || hare:Competitor

State Space:

2 \* 2 = 2 \*\* 2

Composing...

potential DEADLOCK

-- States: 4 Transitions: 4 Memory used: 19373K

Composed in 70ms

Both anwsers holds.

For 3 step -> 16 states, 24 transitions

Result from LTS tool:

Composition:

Race5K = tortoise:Competitor || hare:Competitor

State Space:

4 \* 4 = 2 \*\* 4

Composing...

potential DEADLOCK

-- States: 16 Transitions: 24 Memory used: 20043K

Composed in 3ms

Both anwsers holds.

For 5 step -> 36 states, 60 transitions

Composition:

Race5K = tortoise:Competitor || hare:Competitor

State Space:

6 \* 6 = 2 \*\* 6

Composing...

potential DEADLOCK

-- States: 36 Transitions: 60 Memory used: 19984K

Composed in 2ms

Both anwsers holds.