**Exercise 1**

1. Immutable classes are inherently safe because their object’s state don’t change after creation. Therefore, they avoid safety problems by avoiding state changes.
2. “Balking” is a pattern for full synchronization which ensures in state-dependent actions that preconditions for state holds. So, an action on an object executes when the object is in a particular state.
3. If you have parts in your code which deals with immutable instance variables, then you can split this part from mutable variables and you can put this piece of code out of synchronized blocks. This is called partial synchronization and this works faster than compared to fully synchronized code. By this you allow other threads to execute immutable part and wait only for mutable part.
4. Containment avoid the need for synchronization with achieving safety by avoiding shared variables. Unsynchronized objects are contained inside other object that have at most one thread active at a time.
5. In balking design, clients are forced to busy-wait if a request fails. This can lead to starvation. Also, keeping synchronized methods in full synchronization with short operations is important because this can lead to starvation.
6. When using partial synchronization model, it’s legitimate to declare some methods synchronized that are dealing with mutable objects. Also, in containment unsynchronized objects should be contained in synchronized objects.

**Exercise 2**

const M = 6

POT = POT[0],

POT[v:0..M] = ( when (v == 0) fillpot -> POT[M] | when (v > 0) getserving -> POT[v-1]).

SAVAGE = (getserving -> SAVAGE).

COOK = (fillpot -> COOK).

|| DINING = (POT||SAVAGE||COOK).

**Exercise 3**

1. It can take integer values 0 to 8. Also, it can have value 9, but that leads to an error.
2. A safety property is used which means this process is composed with a system to check that an error state cannot be reached.
3. enter, enter, enter, enter, enter, enter, enter, enter, enter -> goes to error state
4. exit (on first state), or enter, exit.

**Exercise 4**

1. Implementation works.
2. I used monitor implementation to ensure safety. Safety holds, but liveness can suffer in decrease in capacity.
3. It scales if you increase the capacity, if you decrease capacity you need to be careful with the initial values you’re pushing shouldn’t exceed capacity.