Sample Solution for Homework 7

Problem 1 AMP, p. 242: Exercise 121 (6 Points)

Please see the file BoundedQueue.java for the solution to part 1 of this exercise. A comprehensive discussion of the complications involving lock-free array-based queue implementations can be found at:
`http://www.codeproject.com/Articles/153898/Yet-another-implementation-of-a-lock-free-circular`

Problem 2 AMP, p. 242: Exercise 123 (7 Points)

We assume that a person cannot eat and feed at the same time. Otherwise, the solution to the exercise is trivial (everyone feeds its left neighbor). We assume the persons are numbered in the order in which they sit around the table. The algorithm proceeds in rounds. In the first round, person 0 feeds person 1 and person 2 feeds person 3. In each subsequent round, if person i has been fed in the previous round, it feeds person $(i + 1) \mod 5$ in this round. Otherwise, if i is fed in this round, it eats, or else it waits.
This algorithm is deterministic and has no contention. It is also maximally concurrent, since in each round 4 people are eating or feeding. However, the algorithm is not fully decentralized as it requires all people to be synchronized by a common clock.

Problem 3 AMP, p. 255: Exercise 128 (6 Points)

Please see the file LockFreeStack.java for the solution to this exercise.

Problem 4 AMP, p. 257: Exercise 132 (6 Points)

The following history shows that this stack implementation is not linearizable. Consider two threads T1 and T2. T1 first executes `push(1)` and then `pop()` in isolation, leaving the stack empty with value 1 stored in location `stack[0]`. Now, T2 executes `push(2)` up to the beginning of line 14. At this point, both `top` and `stack[0]` are 1. Now, T1 executes `pop()`, yielding 1, even though that value has already been popped before.
See the file Stack.java for an implementation that fixes this problem using the room data structure.