Part A: Investigating Java Synchronization

1. Synchronized / wait / notify
   Synchronized by itself is probably one of the most efficient, and every object more or less comes built in with a wait/notify natively already – so you can synchronize access to whichever variable(s) or object(s) you use to manage shared state.

2. Counting semaphores
   A 0/1 binary semaphore that only lets one person have access to food at any time has the downside of basically capping it at one philosopher eating at any time, or depending on the number of people seated at the table, can effectively halve performance, but is simple to implement and doesn't really fail if done correctly.

3. ReentrantLock
   RL more or less acts as synchronized and keeps trying to retry the lock. Reasonably easy to use but have to manually manage lock acquisition and releasing.

Step B: How my solution works

Sources: Operating System Concepts ('monitor DP pseudocode')

- Initially, all seated philosophers are idling. This status is accessed via synchronized keyword.
- The testAndEat method is used as a synchronized "guard" for both the take and return fork operations.
- When a philosopher wants to start eating, takeForks initially sets that philosopher's state to Hungry, then tests.
- testAndEat here validates both sides of the philosopher (via modulo on the array) to see if either side is eating – if either side is eating, then the shared utensil isn't available, and only acquiring one will eventually result in some form of deadlock if not released.
- If both sides of the philosopher are available/unused, then the philosopher can transition to eating.

Step C: Attached
Step D: Attached, all state changes are logged