Dining Philosophers and Starvation

Why None of the Three Solutions Are Starvation-Free

Solution 1: `synchronized` + `wait()` / `notifyAll()`

This approach uses a shared monitor with `synchronized` blocks and `notifyAll()` to wake up all waiting philosophers. Each philosopher checks whether she can eat using a `test()` function.

Why starvation can occur:

- `notifyAll()` wakes all waiting philosophers, but only those whose neighbors are not eating are allowed to proceed.
- A philosopher may be consistently passed over if others are faster to re-acquire the lock and eat again before she gets her chance.
- Java does **not guarantee fairness** in thread scheduling or lock acquisition.

So, even if a philosopher has been hungry the longest, there's no mechanism to ensure she gets priority.

Solution 2: Semaphores

This solution models each chopstick as a `Semaphore(1)`. Philosophers attempt to acquire the left and right chopsticks in a fixed order.

Why starvation can occur:

- If some philosophers are faster than others (e.g., due to thread scheduling or shorter sleep times), they may always acquire both chopsticks before slower ones can.
- There is **no mechanism to ensure fair access** to chopsticks — each call to `acquire()` simply blocks or succeeds based on availability, not fairness or wait time.
- Some philosophers may repeatedly be unable to acquire both chopsticks, leading to starvation.

Even if deadlock is avoided by asymmetric pickup order, **starvation is still possible** if one philosopher repeatedly fails to acquire both chopsticks.

Solution 3: `ReentrantLock` **+** `Condition[]`

Each philosopher has a `Condition` object and waits on it if unable to eat. After finishing, a philosopher signals her neighbors if they might be able to proceed.

Why starvation can occur:

- Although the outer `ReentrantLock` can be fair (with `new ReentrantLock(true)`), the `Condition` signaling is **not inherently fair**.
- A philosopher only signals her immediate neighbors, and there is no global view or queue to ensure fairness across all philosophers.
- If neighbors keep eating alternately, a hungry philosopher who is not directly adjacent may **never be signaled**.

Thus, a philosopher can be hungry indefinitely if not explicitly woken up and allowed to eat.

Conclusion:
All three solutions can prevent deadlock with careful design, but **none include a mechanism to ensure long-waiting philosophers eventually eat**. Without fairness control (e.g., FIFO queues, aging, or explicit tracking of hunger duration), **starvation remains possible** in each case.