Dining Philosophers and Deadlocks

How to Introduce Deadlocks in the Three Dining Philosophers Solutions

---

1. `DiningPhilosophersSync` — Deadlock by Disabling Neighbor Checks

To create a deadlock in the synchronized solution, remove the calls to `test()` in the `putdown()` method. For example:

```java
public synchronized void putdown(int i) {
    state[i] = THINKING;
    // test((i + 4) % NUM_PHILOSOPHERS); // disabled
    // test((i + 1) % NUM_PHILOSOPHERS); // disabled
    notifyAll();
}
```

Result:
Philosophers who are waiting will never be re-evaluated. Even if both of their neighbors have finished eating, they remain blocked in `wait()`. Over time, all philosophers may become stuck in the HUNGRY state, leading to a deadlock.

---

2. `DiningPhilosophersSem` — Deadlock by Circular Wait

To cause a deadlock with semaphores, make all philosophers acquire chopsticks in the same order — first the left, then the right:

```java
public void pickup(int i) throws InterruptedException {
    int left = i;
    int right = (i + 1) % NUM_PHILOSOPHERS;

    chopsticks[left].acquire();
    Thread.sleep(10); // optional: increase timing sensitivity
```

```
    chopsticks[right].acquire();
}
```

Result:
Each philosopher acquires their left chopstick and waits for the right one. But the right chopstick is already held by their neighbor, who is doing the same. This creates a circular wait — a textbook deadlock condition.

---

3. `DiningPhilosophersLock` — Deadlock by Not Signaling Neighbors

Similar to the synchronized version, comment out the `test()` calls in `putdown()`:

```java
public void putdown(int i) {
    lock.lock();
    try {
        state[i] = THINKING;
        // test((i + 4) % NUM_PHILOSOPHERS); // disabled
        // test((i + 1) % NUM_PHILOSOPHERS); // disabled
    } finally {
        lock.unlock();
    }
}
```

Result:
Waiting philosophers are never signaled because no `test()` is performed after a philosopher finishes eating. They remain blocked on their `Condition` objects. This can stall the entire system, resulting in deadlock.

In each case, the deadlock is introduced by violating proper signaling or by establishing a circular resource wait. These examples are useful for demonstrating how easily synchronization errors can freeze concurrent systems.