Synchronization Strategies for Shared Resources by using lock() and tryLock()

Overview

This document summarizes two approaches for synchronizing access to a shared resource (`itemsOnNotepad`) among multiple threads (Shopper instances) and explains why one approach is generally more efficient than the other.

Case 1: Using Lock (pencil.lock())

- **Mechanism**: A Shopper locks the pencil when it wants to modify `itemsOnNotepad`.
- **Behavior**: If a Shopper has acquired the lock, others must wait until it is released.
- **Impact**:

  - ✓ Increased waiting time due to only one thread being able to modify the resource at a time.
  - ✓ Potential idleness of threads when waiting, leading to longer overall execution times.

Case 2: Using tryLock()

- **Mechanism**: A Shopper attempts to acquire the lock using `pencil.tryLock()`.
- **Behavior**: If the lock is unavailable, the Shopper can proceed with alternative tasks (simulating buying other items).
- **Impact**:

  - ✓ Enhanced concurrent execution since threads continue progressing without being blocked.
  - ✓ Reduction in overall execution time through improved resource utilization.

Why is Case 2 Faster?

1. **Non-blocking Behavior**: Threads that fail to acquire the lock can still execute other code, allowing continuous progress.
2. **Efficient Resource Utilization**: Using `tryLock()` lets threads utilize CPU cycles more effectively, minimizing waiting times.
3. **Parallel Execution**: Threads can switch to other tasks quickly, facilitating faster overall operation and achieving the target of processing up to 20 items more efficiently.

Conclusion

The use of `tryLock()` in Case 2 promotes faster completion times by allowing threads to perform productive work even when locks are unavailable, while the locking mechanism in Case 1 can create bottlenecks and result in increased execution time.