Laboratory

### Problem: 1. Supermarket inventory

1000 of records  
  
  
Threads:  
 5 -> 0.072 seconds  
 500 -> 0.128 seconds  
 1000 -> 0.192 seconds

Computer specs:

Intel I5-8300H (4Cores & 8 Threads)

16GB RAM DDR3 2666 Mhz

**Shared Data**

* **Inventory**: Represents the available stock of products. Multiple threads can access and modify it simultaneously.
* **Bill**: Represents a record of a specific sale. Each Transaction thread generates its own Bill, which is not shared between threads, so it doesn't need synchronization.
* **Transactions**: Each thread operates on its own Transaction object but modifies the shared Inventory.

**Mutexes (Locks)**

Mutexes are used to protect shared resources. Here, we use a ReentrantLock to ensure that only one thread can modify a particular resource at a time.

**Inventory Lock (Granular locking per product)**

Each product in the inventory is a shared resource that can be modified by multiple threads concurrently. To ensure thread safety, each product's operations should be locked when modifying its quantity.

* **Locking Strategy**:
  + Each operation (add or remove) on the product's quantity in the inventory must be done inside a locked section.
  + Use product-level locks to allow different products to be modified concurrently by different threads, but prevent simultaneous modification of the same product by multiple threads.

**Rule**: Protect any modification (addition or removal) of a product's quantity with a ReentrantLock specific to that product.

* Mutex protects for each run the modification of the quantity of each product in order to avoid the deadlock.
* A verification could occure with a probability of 0.6 for each transaction.
* **Granularity of Locks**: By using a lock per product rather than locking the entire inventory, you allow multiple transactions to occur concurrently on different products. This increases performance because threads do not block each other when they are operating on different products.
* **Deadlocks**: Ensure that locks are released in the correct order (in a finally block) to avoid deadlocks where two threads wait for each other to release locks.
* **Contention**: Monitor the number of threads trying to modify the same product, as this can cause contention and reduce performance.