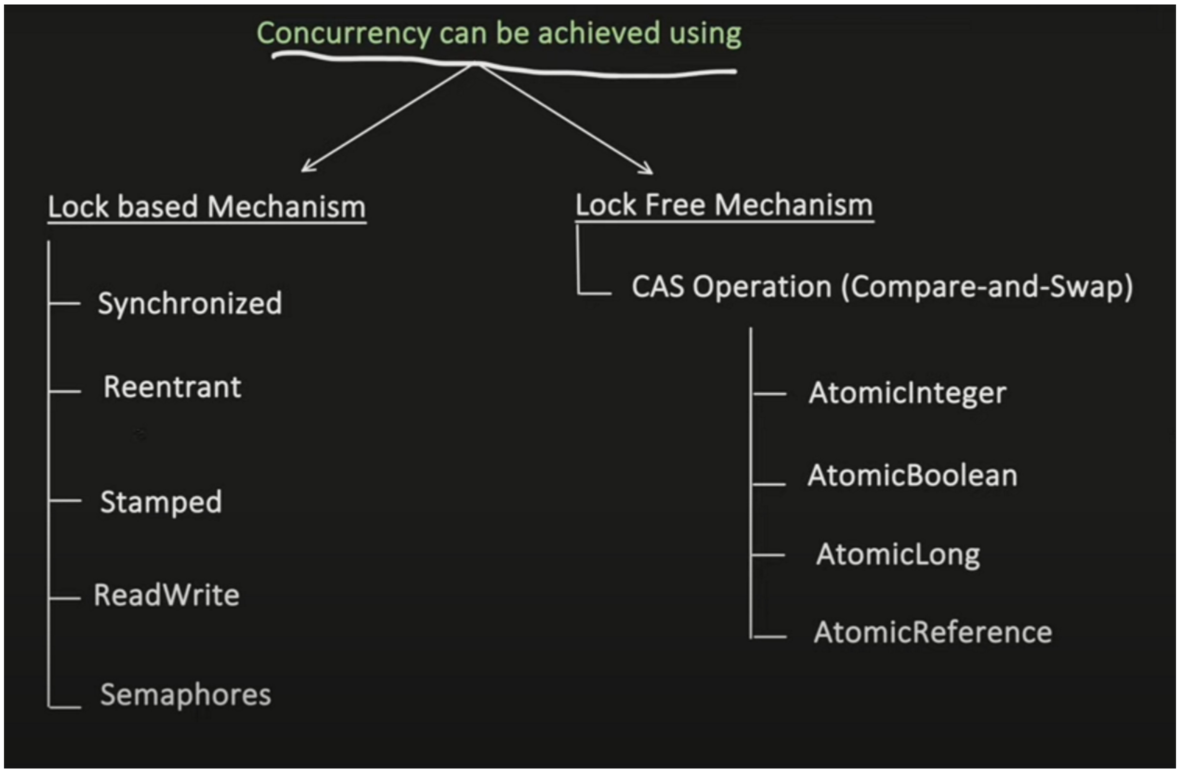
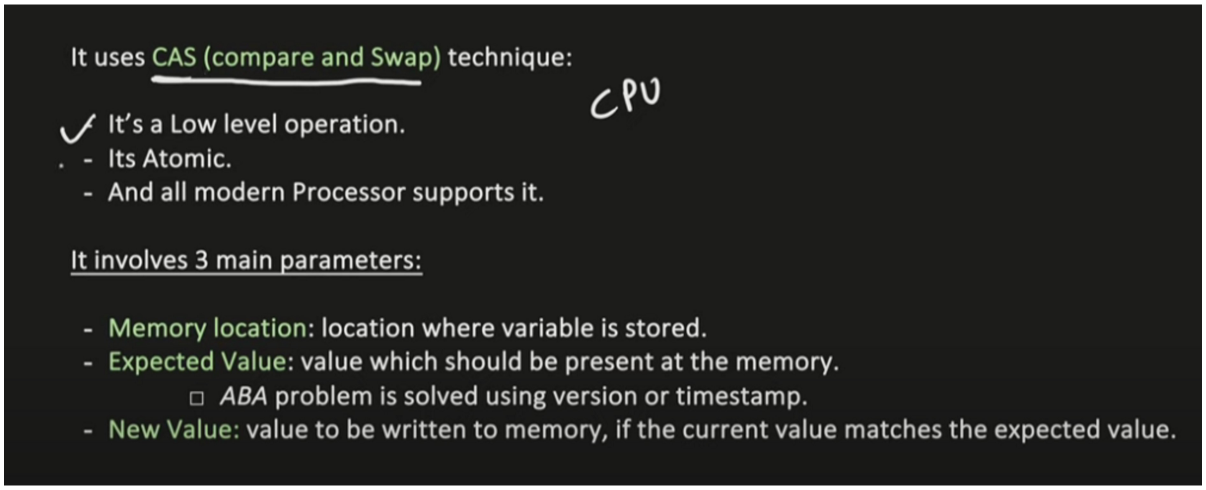
Lesson 5 : Atomic Variables & Concurrent Collections

Saturday, May 4, 2024 10:04 PM



Lock free mechanism is not alternative of lock based mechanism, it has specific use cases(read,update,modify)



It uses CAS (compare and Swap) technique:

CPU

- ✓ It's a Low level operation.
- Its Atomic.
- And all modern Processor supports it.

It involves 3 main parameters:

- Memory location: location where variable is stored.
- Expected Value: value which should be present at the memory.
  - *ABA* problem is solved using version or timestamp.
- New Value: value to be written to memory, if the current value matches the expected value.

CAS is just like optimistic concurrency control but at CPU level that is why it is called low level operation. It compares the memory data with expected value if there is match then sets new value to memory.

ABA problem : suppose t1 wants to update 10 to 13 but meantime somethread has changed value from 10 to 12 then again to 10, so the expected value and memory value will be same although it is not same 10 as was 2 versions ago so it will update the new value.

ATOMIC VARIABLES

Atomic means single or "all or nothing". Single operation

Example :

```
Int counter = 0;
public void increment(){
        counter++;
}
```

The counter++ statement is not atomic. Because :
1. Load counter value
2. Increase it by 1
3. Assign back

If 2 threads call increment(), then if they reach at same time, then

Operation	T1	T2
Read Value	0	0
Increase	1	1
Updated	1	1

So both threads invoked increment() but the value only increased by 1 but had to be increased by 2. Thus the method is not thread safe.

To resolve this we can either use locking mechanisms or go for lock free mechanism like Atomic Integer

```
AtomicInteger counter = new AtomicInteger(0);
public void increment(){
        counter.incrementAndGet();
}
```

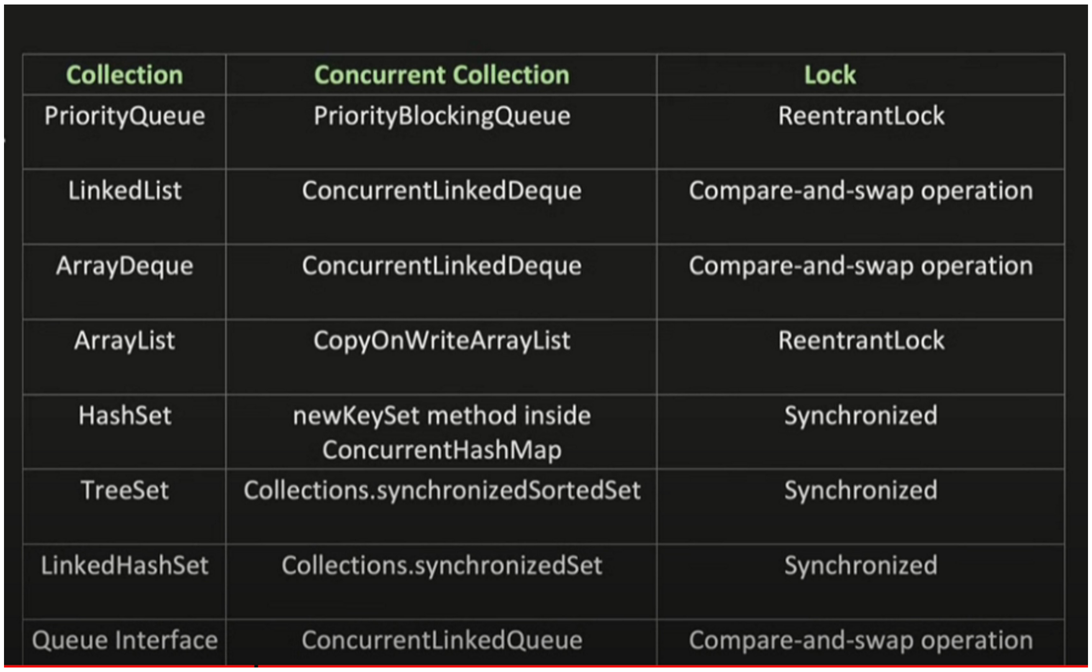
This is thread safe and we we see implmentation of incrementAndGet() we see it is using compareAndSwap
So if we have read,modify, update scenarios we can use atomic

VOLATILE KEYWORD

```
volatile int value = 10;
```

Each core has its own cache called L1 cache, so if any thread running on core 1 changes value it first updates it L1 cache to 11, but by then if thread running on another core tries to read value it will read from RAM where it still is 10. So there might be a miss. Eventually all L1 caches sync but in edge cases correct value might be missed. So volatile makes sure that read and write should happen from memory(RAM). It should not be confused that volatile solves concurrency problem, it just ensures that read/writes happen in RAM.

CONCURRENT COLLECTION



Collection	Concurrent Collection	Lock
PriorityQueue	PriorityBlockingQueue	ReentrantLock
LinkedList	ConcurrentLinkedDeque	Compare-and-swap operation
ArrayDeque	ConcurrentLinkedDeque	Compare-and-swap operation
ArrayList	CopyOnWriteArrayList	ReentrantLock
HashSet	newKeySet method inside ConcurrentHashMap	Synchronized
TreeSet	Collections.synchronizedSortedSet	Synchronized
LinkedHashSet	Collections.synchronizedSet	Synchronized
Queue Interface	ConcurrentLinkedQueue	Compare-and-swap operation