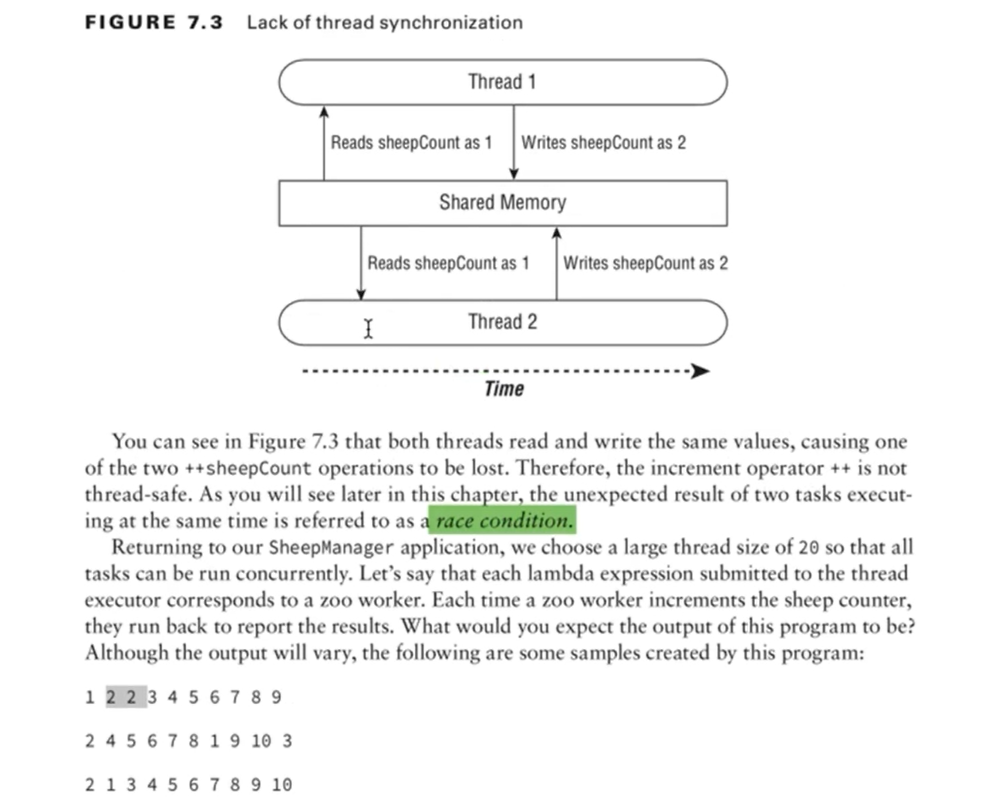
**CONCURRENCY (EŞ ZAMANLILIK)**

**ATOMIC**

Synchronized bir metoda aynı anda bir thread’in kullanmasını sağlamaktadır. Fakat synchronized keywordunun kullanımı maliyetli olduğundan onun yerine bazı durumlarda AtomicInteger’lar tercih edilir. Çünkü bu mekanizma kilit (lock) mekanizması yerine, daha hızlı ve low-level olan Compare-And-Set (CAS) mekanizmasını kullanmaktadır.

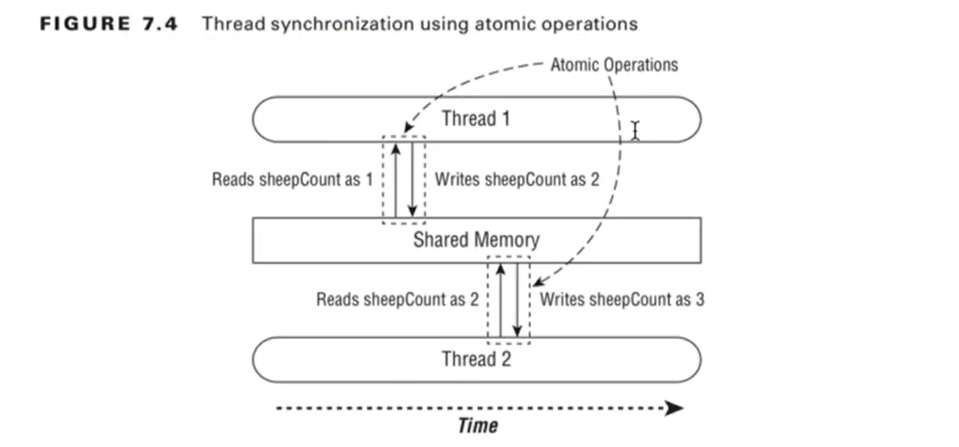


Bu gibi durumların önüne geçebilmek için atomic classlar kullanılmalıdır.

NOT: Synchronizing methodlar’da lock mekanizması olduğu için buralarda atomic integer kullanımına gerek yoktur. Fakat maliyetlidir. Zaten amacı datayı korumaktır o nedenle performanstan ödün vermektedir.

Maliyet Örneği:

50 thread x 100 milliseconds -> 5000 milliseconds -> 5 seconds



Atomic operasyonlarda işlem bir bütün olarak gerçekleşir. Böylelikle aradaki kayıpların engellenmesi hedeflenmiştir.



**CONCURRENT COLLECTIONS**

* Collectionlara thread-safe kabiliyeti kazandıran collectiondır.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yukarıdaki örnek kodları incelediğimiz de scynchronized kullanmak yerine ConcurrentHashMap kullanmak daha doğru bir yaklaşımdır. Böylelikle ilgili map üzerinde birden fazla thread güvenli ve hızlı bir şekilde çalışabilecektir.

* Birden fazla thread non-concurrent collection üzerinde işlem yapmaya çalıştığında JVM ConcurrentModificationException fırlatabilir. Concurrent özelliğe sahip collectionları kullanmak daha doğru bir yaklaşım olacaktır.

Collection sınıflarına karşılık gelen thread-safe yapıdaki sınıflardan birkaçını inceleyelim.

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

Çalışma mantığı bir değişiklik olduğunda listedeki bütün elemanları kopyalayarak baştan oluşturmaktadır.

**CopyOnWriteArrayList Class**

**Örnek ile açıklayalım:** ArrayList üzerinde aynı anda birden fazla işlem yapmaya çalışılan bir senaryo.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yukarıdaki örnek kod bloğunda bir thread okuma yaparken, diğer bir thread ekleme işlemi gerçekleştirmektedir. Sonuç olarak bu durumda ConcurrentModificationException hatası vermektedir.

**Örneği refactoring edelim:**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

ArrayList’in thread safe olması ve çalışma zamanı hatalarının önlenmesi için CopyOnWriteArrayList sınıfının kullanılması tercih edilir.

**ConcurrentHashMap** **Class**

**Örnek ile açıklayalım**: HashMap üzerinde aynı anda birden fazla işlem yapmaya çalışılan bir senaryo.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yukarıdaki örnek kod bloğunda bir thread okuma yaparken, diğer bir thread silme işlemi gerçekleştirmektedir. Sonuç olarak bu durumda ConcurrentModificationException hatası vermektedir.

**Örneği refactoring edelim:**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

HashMap’in thread safe olması ve çalışma zamanı hatalarının önlenmesi için ConcurrentHashMap sınıfının kullanılması tercih edilir.

**SYCHRONIZED COLLECTIONS**

Sychronize olmayan collecitonları sychronize yapmaya yarayan methodlardır. Mevcut bulunan listeyi thread-safe hale getirmek istediğimizde kullanılabilir. Yada Anlık thread-safe bir işlem yapılacağı zaman bu özellik kullanılıp tekrar non-synchronize hale getirilebilir. Aynı anda birden fazla işlem yapılmak istenildiğinde hata fırlatacaktır. Yani listeyi sychronize bir method içine almak gibide düşünebiliriz. Listeyi locklayarak aynı anda bir thread’in kullanmasını sağlayacaktır.

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

List<Integer> list = Arrays.asList(1,2,3,4,5,6,7);

List<Integer> synchronizedList = Collections.synchronizedList(list);