Mutex

Mutex 1

- A mutex (or mutual exclusion) is the simplest type of *synchronizer* – it **ensures that only one thread can execute the critical section of a computer program at a time.**
- To access a critical section, a thread acquires the mutex, then accesses the critical section, and finally releases the mutex. In the meantime, **all other threads block till the mutex releases.**
- As soon as a thread exits the critical section, another thread can enter the critical section.
- Mutex (o'zaro istisno) *synchronizer ni eng oddiy turi bo'lib u* dasturni bitta qismini bir vaqtni o'zida bitta Thread ishlatishini taminlaydi.
- Thread dasturni muxim bo'lagiga kirish uchun mutex ni talab qiladi, mutexni olganidna keyin muhim joyga kiradi va ishlaydi. Muhim joydan chiqib ketgach mutex ni qo'yib yuboradi.
- Thread muhim joydan chiqib ketgan boshqa Thread lar muhim joyni talab qilib unga kirishi mumkin.

Mutex 2

- To avoid such race conditions, we need to **make sure that only one thread can execute the shared_resource/method at a time**. In such scenarios, we can use a mutex to synchronize the threads.
- There are various ways, we can implement a mutex in Java. So, next, we'll see the different ways to implement a mutex for our *SequenceGenerator* class.

- Race Condition lardna qochish uchun bir vatni o’zida bitta Thread umumiy_manmani/methodni ishlatishi kerak. Shu holatda biz Thread larni synchronize qilish uchun mutex dan foydalanishimi mumkin.
- Javada mutex ni ishlatilishi mumkin holatlar juda ko’p.

Mutex 3

- Mutex is not a class. IT is definition. Mutux can be implemented by several ways:
  - ***Using synchronized* method**
  - ***Using synchronized* block**
  - **Using *ReentrantLock***
  - **Using *Semaphore***
  - **Using Guava's *Monitor* Class**
- Mutex bitta class emas. U bitta tushuncha. Mutex bir nechta yo’l bilan amalga ochirish mumkin. Ular: …

Resource with out Mutex

- So, to avoid such race conditions, we need to **make sure that only one thread can execute the *getNextSequence* method at a time**. In such scenarios, we can use a mutex to synchronize the threads.

```
public class SequenceGenerator {

    private int currentValue = 0;

    public int getNextSequence() {
        currentValue = currentValue + 1;
        return currentValue;
    }

}
```

Mutex **Using *synchronized method***

```
public class SequenceGenerator {

    private int currentValue = 0;

    public synchronized  int getNextSequence() {
        currentValue = currentValue + 1;
        return currentValue;
    }

}
```

Mutex **Using *synchronized block***

```
public class SequenceGenerator {

    private int currentValue = 0;

    public  int getNextSequence() {
            synchronized (currentValue) {
                    currentValue = currentValue + 1;
                    return currentValue;
             }
    }

}
```

Mutex using *ReentrantLock*

```
public class SequenceGenerator {

    private int currentValue = 0;
    private ReentrantLock mutex = new ReentrantLock();

    public int getNextSequence() {
        try {
            mutex.lock();
            currentValue = currentValue + 1;
            return currentValue;
        } finally {
            mutex.unlock();
        }
    }

}
```

Mutex Using *Semaphore*

- While in case of a mutex only one thread can access a critical section, *Semaphore* allows **a fixed number of threads to access a critical section**. Therefore, **we can also implement a mutex by setting the number of allowed threads in a *Semaphore* to one.**

```
public class SequenceGenerator {

    private int currentValue = 0;
    private Semaphore mutex = new Semaphore(1);


    public int getNextSequence() {
                        try {
            mutex.acquire();
            currentValue = currentValue + 1;
                                return currentValue;
        } finally {
            mutex.release();
        }
    }
}
```

Mutex using **Guava's *Monitor* Class3**

```
public class SequenceGenerator {
    private int currentValue = 0;
    private Monitor mutex = new Monitor();

     public int getNextSequence() {
          try {
                mutex.enter();
                currentValue = currentValue + 1;
               return currentValue;
          } finally {
           mutex.leave();
        }
    }
}
```

```
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>31.0.1-jre</version>
</dependency>
```

Mutex Links

- https://www.baeldung.com/java-mutex
- https://mkyong.com/java/java-thread-mutex-and-semaphore-example/