

Java: Lock and Condition (Multithreading Part4)

"Concept && Coding" YT Video Notes

ReentrantLock:

```

public class SharedResource {
    boolean isAvailable = false;
    ReentrantLock lock = new ReentrantLock();

    public void producer(){
        try {
            lock.lock();
            System.out.println("Lock acquired by: " + Thread.currentThread().getName());
            isAvailable = true;
            Thread.sleep(millis: 4000);
        } catch (Exception e) {
        }
        finally{
            lock.unlock();
            System.out.println("Lock release by: " + Thread.currentThread().getName());
        }
    }
}

```

ReadWriteLock:

ReadLock: More than 1 thread can acquire the read lock
WriteLock: Only 1 thread can acquire the write lock.

```

public class Main {
    public static void main(String args[]){

        SharedResource resource = new SharedResource();
        ReadWriteLock lock = new ReentrantReadWriteLock();

        Thread th1 = new Thread() -> {
            resource.producer(lock);
        };

        Thread th2 = new Thread() -> {
            resource.producer(lock);
        };

        SharedResource resource1 = new SharedResource();
        Thread th3 = new Thread() -> {
            resource1.consume(lock);
        };

        th1.start();
        th2.start();
        th3.start();
    }
}

```

```

public class SharedResource {
    boolean isAvailable = false;
    public void producer(ReadWriteLock lock){
        try {
            lock.readLock().lock();
            System.out.println("Read Lock acquired by: " + Thread.currentThread().getName());
            isAvailable = true;
            Thread.sleep(millis: 8000);
        } catch (Exception e) {
        }
        finally{
            lock.readLock().unlock();
            System.out.println("Read Lock release by: " + Thread.currentThread().getName());
        }
    }

    public void consume(ReadWriteLock lock){
        try {
            lock.writeLock().lock();
            System.out.println("Write Lock acquired by: " + Thread.currentThread().getName());
            isAvailable = false;
        } catch (Exception e) {
        }
        finally{
            lock.writeLock().unlock();
            System.out.println("Write Lock release by: " + Thread.currentThread().getName());
        }
    }
}

```

StampedLock:

- Support Read/Write Lock functionality like ReadWriteLock

```

public class Main {
    public static void main(String args[]){

        SharedResource resource = new SharedResource();

        Thread th1 = new Thread() -> {
            resource.producer();
        };

        Thread th2 = new Thread() -> {
            resource.producer();
        };

        Thread th3 = new Thread() -> {
            resource.consume();
        };

        th1.start();
        th2.start();
        th3.start();
    }
}

```

```

public class SharedResource {
    boolean isAvailable = false;
    StampedLock lock = new StampedLock();

    public void producer(){
        long stamp = lock.readLock();
        try {
            System.out.println("Read Lock acquired by: " + Thread.currentThread().getName());
            isAvailable = true;
            Thread.sleep(millis: 6000);
        } catch (Exception e) {
        }
        finally{
            lock.unlockRead(stamp);
            System.out.println("Read Lock release by: " + Thread.currentThread().getName());
        }
    }

    public void consume(){
        long stamp = lock.writeLock();
        try {
            System.out.println("Write Lock acquired by: " + Thread.currentThread().getName());
            isAvailable = false;
        } catch (Exception e) {
        }
        finally{
            lock.unlockWrite(stamp);
            System.out.println("Write Lock release by: " + Thread.currentThread().getName());
        }
    }
}

```

- Support Optimistic Lock functionality too

```

public class Main {
    public static void main(String args[]){

        SharedResource resource = new SharedResource();

        Thread th1 = new Thread() -> {
            resource.producer();
        };

        Thread th2 = new Thread() -> {
            resource.consumer();
        };

        th1.start();
        th2.start();
    }
}

```

```

public class SharedResource {
    int a = 10;
    StampedLock lock = new StampedLock();
    public void producer(){
        long stamp = lock.tryOptimisticRead();
        try {
            System.out.println("Taken optimistic lock");
            a = 11;
            Thread.sleep(millis: 4000);
            if(lock.validate(stamp)){
                System.out.println("updated a value successfully");
            }
            else {
                System.out.println("rollback of work");
                a = 10; //rollback
            }
        } catch (Exception e) {
        }
    }

    public void consumer(){
        long stamp = lock.writeLock();
        System.out.println("write lock acquired by: " + Thread.currentThread().getName());

        try {
            System.out.println("performing work");
            a = 9;
        } finally {
            lock.unlockWrite(stamp);
            System.out.println("write lock released by: " + Thread.currentThread().getName());
        }
    }
}

```

Semaphore Lock:

```

public class Main {
    public static void main(String args[]){

        SharedResource resource = new SharedResource();
        Thread th1 = new Thread() -> {
            resource.producer();
        };
        Thread th2 = new Thread() -> {
            resource.producer();
        };
        Thread th3 = new Thread() -> {
            resource.producer();
        };
        Thread th4 = new Thread() -> {
            resource.producer();
        };

        th1.start();
        th2.start();
        th3.start();
        th4.start();
    }
}

```

```

public class SharedResource {
    boolean isAvailable = false;
    Semaphore lock = new Semaphore(permits: 2);

    public void producer(){
        try {
            lock.acquire();
            System.out.println("Lock acquired by: " + Thread.currentThread().getName());
            isAvailable = true;
            Thread.sleep(millis: 4000);
        } catch (Exception e) {
        }
        finally{
            lock.release();
            System.out.println("Lock release by: " + Thread.currentThread().getName());
        }
    }
}

```

Condition

await() = wait()
signal() = notify()