**Read-Write Lock**

A **Read-Write Lock** allows multiple threads to read a resource simultaneously but restricts write access so that only one thread can modify the resource at any time. This is useful in scenarios where reads occur more frequently than writes, reducing contention and improving performance.

* **Java Implementation**: ReentrantReadWriteLock
* **Key Methods**:
  + readLock().lock(): Acquires the read lock (multiple threads can hold it).
  + writeLock().lock(): Acquires the write lock (only one thread can hold it).
* **package** Locking;
* **import** java.util.concurrent.locks.Lock;
* **import** java.util.concurrent.locks.ReadWriteLock;
* **import** java.util.concurrent.locks.ReentrantReadWriteLock;
* **public** **class** ReadWriteCounter {
* **private** **int** count = 0;
* **private** **final** ReadWriteLock lock = **new** ReentrantReadWriteLock();
* **private** **final** Lock readLock = lock.readLock();
* **private** **final** Lock writeLock = lock.writeLock();
* **public** **void** increment() {
* writeLock.lock();
* **try** {
* count++;
* Thread.*sleep*(50);
* } **catch** (InterruptedException e) {
* **throw** **new** RuntimeException(e);
* } **finally** {
* writeLock.unlock();
* }
* }
* **public** **int** getCount() {
* readLock.lock();
* **try** {
* **return** count;
* } **finally** {
* readLock.unlock();
* }
* }
* **public** **static** **void** main(String[] args) **throws** InterruptedException {
* ReadWriteCounter counter = **new** ReadWriteCounter();
* Runnable readTask = **new** Runnable() {
* @Override
* **public** **void** run() {
* **for** (**int** i = 0; i < 10; i++) {
* System.***out***.println(Thread.*currentThread*().getName() + " read: " + counter.getCount());
* }
* }
* };
* Runnable writeTask = **new** Runnable() {
* @Override
* **public** **void** run() {
* **for** (**int** i = 0; i < 10; i++) {
* counter.increment();
* System.***out***.println(Thread.*currentThread*().getName() + " incremented");
* }
* }
* };
* Thread writerThread = **new** Thread(writeTask);
* Thread readerThread1 = **new** Thread(readTask);
* Thread readerThread2 = **new** Thread(readTask);
* writerThread.start();
* readerThread1.start();
* readerThread2.start();
* writerThread.join();
* readerThread1.join();
* readerThread2.join();
* System.***out***.println("Final count: " + counter.getCount());
* }
* }