Enthuware Mobile Test Studio

Home | Certifications | **Test** | Review | Progress | Notes

Standard Tests - Test 3 : 2019-10-13 22:40

Q 59 of 86 Mark Concurrency enthuware.ocpjp.v8.2.1106

Consider the following code:

```
public class Student {
    private Map<String, Integer> marksObtained = new HashMap<String, Integer>();
    private ReadWriteLock lock = new ReentrantReadWriteLock();
    public void setMarksInSubject(String subject, Integer marks){
      // valid code to set marks for a given subject
    }
    public double getAverageMarks(){

        //1 - INSERT CODE HERE

        double sum = 0.0;
        try{
            for(Integer mark : marksObtained.values()){
                sum = sum + mark;
            }
            return sum/marksObtained.size();
        }finally{

               //2 - INSERT CODE HERE

        }
    }
}
```

What should be inserted at //1 and //2?

Answered Incorrectly You had to select 1 option(s)

- `lock.lock();` `lock.unlock();`
- **`lock.readLock();` `lock.readUnlock();`**
- `lock.read();` `lock.unlock();`
- **`lock.readLock().lock();` and `lock.readLock().unlock();`**

Previous | Next | Evaluate | Finish | Review

From a `ReadWriteLock`, you can get one read lock (by calling `lock.readLock()`) and one write lock (by calling

`lock.writeLock()`). Even if you call these methods multiple times, the same lock is returned. A read lock can be locked by multiple threads simultaneously (by calling `lock.readLock().lock()`), if the write lock is free. If the write lock is not free, a read lock cannot be locked. The write lock can be locked (by calling `lock.writeLock().lock()`) only by only one thread and only when no thread already has a read lock or the write lock. In other words, if one thread is reading, other threads can read, but no thread can write. If one thread is writing, no other thread can read or write.

Methods that do not modify the collection (i.e. the threads that just "read" a collection) should acquire a read lock and threads that modify a collection should acquire a write lock.

The benefit of this approach is that multiple reader threads can run without blocking if the write lock is free. This increases performance for read only operations. The following is the complete code that you should try to run:

```
public class Student {
    private Map<String, Integer> marksObtained = new HashMap<String, Integer>();
    private ReadWriteLock lock = new ReentrantReadWriteLock();
    public void setMarksInSubject(String subject, Integer marks){
        lock.writeLock().lock(); //1
        try{
            marksObtained.put(subject, marks);
        }finally{
            lock.writeLock().unlock(); //2
        }
    }
    public double getAverageMarks(){
        lock.readLock().lock(); //3
        double sum = 0.0;
        try{
            for(Integer mark : marksObtained.values()){
                sum = sum + mark;
            }
            return sum/marksObtained.size();
        }finally{
            lock.readLock().unlock();//4
        }
    }
    public static void main(String[] args) {

        final Student s = new Student();

        //create one thread that keeps adding marks
        new Thread(){
            public void run(){
                int x = 0;
                while(true){
                    int m = (int)(Math.random()*100);
                    s.setMarksInSubject("Sub "+x, m);
                    x++;
                }
            }
       }.start();

       //create 5 threads that get average marks
       for(int i=0;i<5; i++){
            new Thread(){
                public void run(){
                    while(true){
                        double av = s.getAverageMarks();
                        System.out.println(av);
                    }
                }
            }.start();
        }
    }
```

```
}
```

Note that if you remove the line //1, //2, //3, and //4, (i.e. if you don't use any locking), you will see a `ConcurrentModificationException`.

Add/Edit Note