In a multithreaded Java program read and write operation from different thread at the same time creates race condition. The simple example here gives different output at different times:

```
public class RaceCondition {

    public static void main(String[] args) throws
InterruptedException{

        final LongWrapper longWrapper = new
LongWrapper(0L);

        Runnable runnable = new Runnable() {
            @Override
            public void run() {
                for(int i =0; i < 1_000; i++) {
                    longWrapper.incrementValue();
                }
            }
        };

    Thread[] threads = new Thread[1_000];

    for(int i =0; i< threads.length; i++){
        threads[i] = new Thread(runnable);
        threads[i].start();
    }

   for(int i =0; i< threads.length; i++){
        threads[i].join();
   }

   System.out.println("Value " + longWrapper.getValue());
   }
}
```

And the LongWrapper class:

```
public class LongWrapper {
```

```
    private long l;

    public LongWrapper(long l){
        this.l = l;
    }

    public long getValue(){
        return  l;
    }

    public void incrementValue(){
        l = l + 1;
    }
}
```

So how to fix this? Just change the LongWrapper Class as follows.
The class level object here will work as a key to lock the synchronized code block to run with only one thread at a time, thus removing the occurrence of Race condition:

```
public class LongWrapper {
    private Object key = new Object();
    private long l;

    public LongWrapper(long l){
        this.l = l;
    }

    public long getValue(){
        return  l;
    }

    public void incrementValue(){
        synchronized (key) {
            l = l + 1;
        }
    }
}
```