# Laboratory 3

**Title of the Laboratory Exercise: Programs based on re-entrant Read Write locks**

1. **Introduction and Purpose of Experiment**

A ReadWriteLock implementation guarantees the following behaviours:

* Multiple threads can read the data at the same time, as long as there’s no thread is updating the data.
* Only one thread can update the data at a time, causing other threads (both readers and writers) block until the write lock is released.
* If a thread attempts to update the data while other threads are reading, the write thread also blocks until the read lock is released.

1. **Aim and Objectives**

**Aim**

* To develop programs using re-entrant read write locks in Java

1. **Experimental Procedure**
   * 1. Analyse the problem statement
     2. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
     3. Implement the algorithm in Java language
     4. Compile the Java program
     5. Test the implemented program
     6. Document the Results
     7. Analyse and discuss the outcomes of your experiment
2. **Question**

Create multithreaded programs using ReadWriteLock in Java

* A writer thread is responsible for randomly adds a number to the shared ReadWriteList list and reader thread is responsible for randomly gets an element from the shared list. The program creates and runs 10 reader threads and 5 writer threads that work on a shared ReadWriteList data structure.

1. **Pseudocode**

**main() {**

**ReadWriteList<Integer> sharedList = new ReadWriteList<>();**

**for (int i = 1; i <= WRITER\_SIZE; i++) {**

**new Writer(sharedList,i).start(); }**

**for (int i = 1; i <= READER\_SIZE; i++) {**

**new Reader(sharedList,i).start() }}**

**ReadWriteList<E> {**

**private List<E> list = new ArrayList<>();//Initialization**

**private ReadWriteLock rwLock = new ReentrantReadWriteLock();**

**add(E element,int wc) {**

**Lock writeLock = rwLock.writeLock();**

**writeLock.lock();**

**Display("Writer-"+wc+" has acquired the lock ...")**

**try {**

**list.add(element);**

**display("Writer-"+wc+" is writing "+element+" to sharedList");**

**} finally {**

**display("Writer-"+wc+" has finsihed writing");**

**display("Writer-"+wc+" has released the lock ...\n");**

**writeLock.unlock();} }**

**public void get(int index,int rc) {**

**Lock readLock = rwLock.readLock();**

**readLock.lock();**

**try {**

**display("Reader-"+rc+" is reading "+list.get(index)+" from sharedList");**

**} finally {**

**display("Reader-"+rc+" has finsihed reading\n");//output Statement**

**readLock.unlock(); }}**

**public int size() {**

**Lock readLock = rwLock.readLock();**

**readLock.lock();**

**try {return list.size();**

**} finally { readLock.unlock(); } }}**

**Writer extends Thread {**

**private ReadWriteList<Integer> sharedList;**

**int wc;**

**public Writer(ReadWriteList<Integer> sharedList,int wc) {**

**this.sharedList = sharedList;**

**this.wc = wc; }**

**@Override**

**public void run() {**

**Random random = new Random();**

**int number = random.nextInt(100);**

**sharedList.add(number,wc);**

**try {Thread.sleep(100);}**

**catch (InterruptedException ie) { ie.printStackTrace();}}}**

**Reader extends Thread {**

**private ReadWriteList<Integer> sharedList;**

**int rc;**

**public Reader(ReadWriteList<Integer> sharedList,int rc) {**

**this.sharedList = sharedList;**

**this.rc = rc; }**

**@Override**

**public void run() {**

**Random random = new Random();**

**int index = random.nextInt(sharedList.size());**

**sharedList.get(index,rc);**

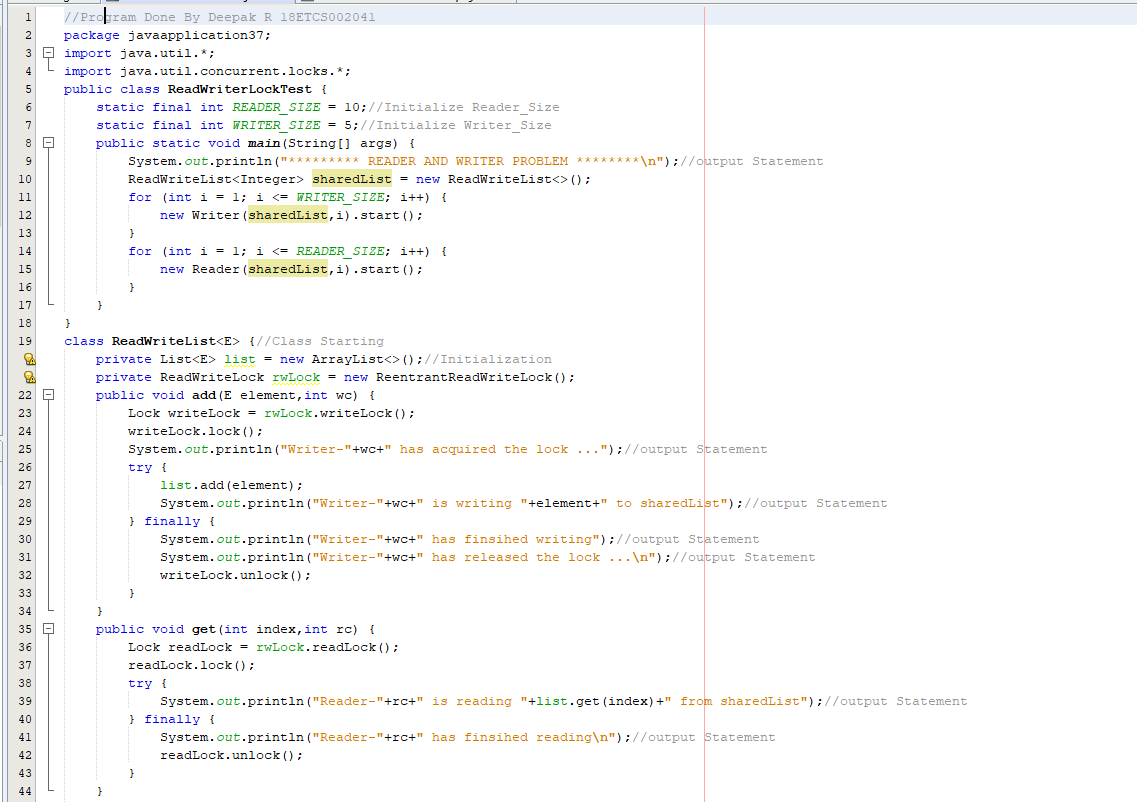
**try {**

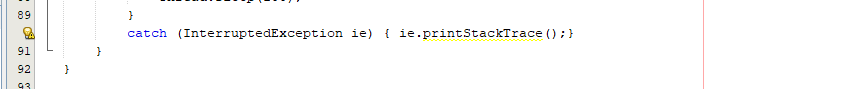
**Thread.sleep(100); }**

**catch (InterruptedException ie) { ie.printStackTrace();} } }**

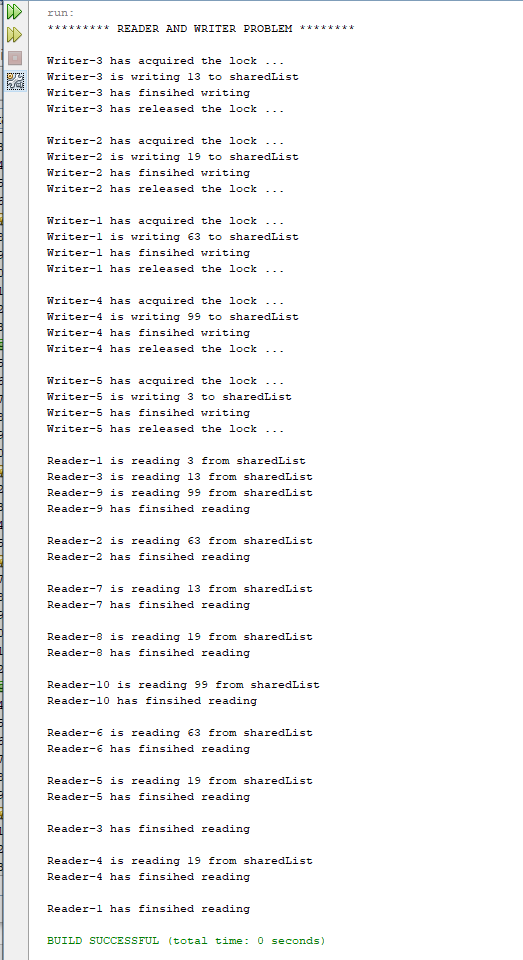
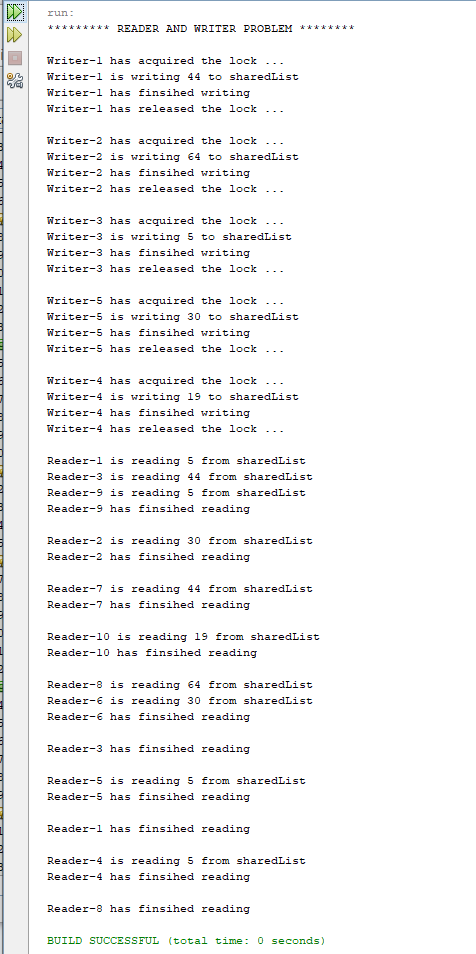
1. **Presentation of Results**

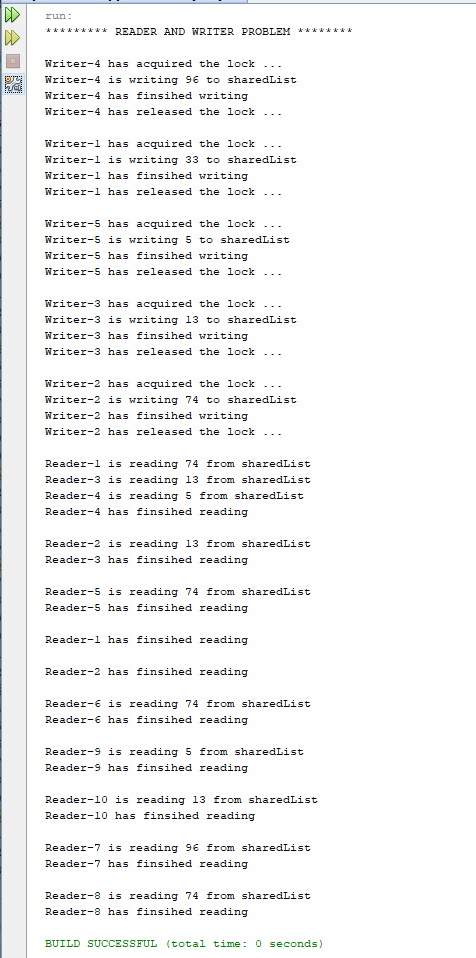
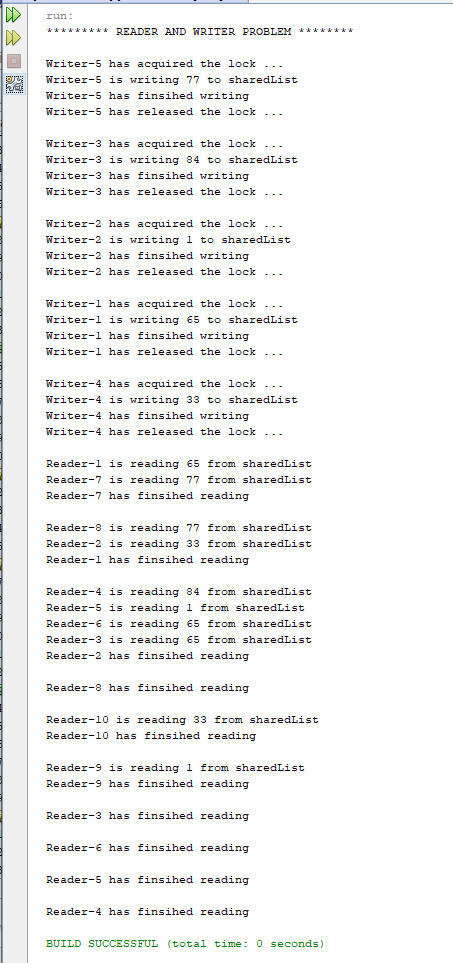
**Program Code**





**Output**





1. **Analysis and Discussions**

A ReadWriteLock implementation guarantees the following behaviors:

Multiple threads can read the data at the same time, as long as there’s no thread is updating the data.

Only one thread can update the data at a time, causing other threads (both readers and writers) block until the write lock is released.

If a thread attempts to update the data while other threads are reading, the write thread also blocks until the read lock is released.

**1.Limitations of Experiments**

So ReadWriteLock can be used to add concurrency features to a data structure, but it doesn’t guarantee the performance because it depends on various factors: how the data structure is designed, the contention of reader and writer threads at real time, CPU architecture (single core  or multicores), etc.

**2. Limitations of Results**

Tested only for small data result may vary if done for large data.

**3. Learning happened**

We learnt thatA ReadWriteLock implementation guarantees the following behaviours:

* Multiple threads can read the data at the same time, as long as there’s no thread is updating the data.
* Only one thread can update the data at a time, causing other threads (both readers and writers) block until the write lock is released.
* If a thread attempts to update the data while other threads are reading, the write thread also blocks until the read lock is released.

**4. Recommendations**

None