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.

2. **Aim and Objectives**

   Aim

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

3. **Experimental Procedure**

   i. Analyse the problem statement

   ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code

   iii. Implement the algorithm in Java language

   iv. Compile the Java program

   v. Test the implemented program

   vi. Document the Results

   vii. Analyse and discuss the outcomes of your experiment

4. **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.

5. 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();} } }
```

6. Presentation of Results

Program Code

```
1  //Program Done By Deepak R 18ETCS002041
2  package javaapplication37;
3  import java.util.*;
4  import java.util.concurrent.locks.*;
5  public class ReadWriterLockTest {
6      static final int READER_SIZE = 10;//Initialize Reader_Size
7      static final int WRITER_SIZE = 5;//Initialize Writer_Size
8      public static void main(String[] args) {
9          System.out.println("********* READER AND WRITER PROBLEM ********\n");//output Statement
10         ReadWriteList<Integer> sharedList = new ReadWriteList<>();
11         for (int i = 1; i <= WRITER_SIZE; i++) {
12             new Writer(sharedList,i).start();
13         }
14         for (int i = 1; i <= READER_SIZE; i++) {
15             new Reader(sharedList,i).start();
16         }
17     }
18 }
19 class ReadWriteList<E> {//Class Starting
       private List<E> list = new ArrayList<>();//Initialization
       private ReadWriteLock rwLock = new ReentrantReadWriteLock();
22     public void add(E element,int wc) {
23         Lock writeLock = rwLock.writeLock();
24         writeLock.lock();
25         System.out.println("Writer-"+wc+" has acquired the lock ...");//output Statement
26         try {
27             list.add(element);
28             System.out.println("Writer-"+wc+" is writing "+element+" to sharedList");//output Statement
29         } finally {
30             System.out.println("Writer-"+wc+" has finsihed writing");//output Statement
31             System.out.println("Writer-"+wc+" has released the lock ...\n");//output Statement
32             writeLock.unlock();
33         }
34     }
35     public void get(int index,int rc) {
36         Lock readLock = rwLock.readLock();
37         readLock.lock();
38         try {
39             System.out.println("Reader-"+rc+" is reading "+list.get(index)+" from sharedList");//output Statement
40         } finally {
41             System.out.println("Reader-"+rc+" has finsihed reading\n");//output Statement
42             readLock.unlock();
43         }
44     }
```

```
45      public int size() {
46          Lock readLock = rwLock.readLock();
47          readLock.lock();
48
49          try {
50              return list.size();
51          } finally {
52              readLock.unlock();
53          }
54      }
55  }
56  class Writer extends Thread {//Writer Class Starting
58      private ReadWriteList<Integer> sharedList;
58      int wc;
59      public Writer(ReadWriteList<Integer> sharedList,int wc) {
60          this.sharedList = sharedList;
61          this.wc = wc;
62      }
63      @Override
65      public void run() {
65          Random random = new Random();
66          int number = random.nextInt(100);
67          sharedList.add(number,wc);
68          try {
69              Thread.sleep(100);
70          }
72          catch (InterruptedException ie) { ie.printStackTrace();}
72      }
73  }
74
75  class Reader extends Thread {//Reader Class Starting
77      private ReadWriteList<Integer> sharedList;
77      int rc;
78      public Reader(ReadWriteList<Integer> sharedList,int rc) {
79          this.sharedList = sharedList;
80          this.rc = rc;
81      }
82      @Override
84      public void run() {//Run
84          Random random = new Random();
85          int index = random.nextInt(sharedList.size());
86          sharedList.get(index,rc);
87          try {
88              Thread.sleep(100);
89          }
91          catch (InterruptedException ie) { ie.printStackTrace();}
91      }
92  }
93
```

Output

```
run:
********* READER AND WRITER PROBLEM ********

Writer-1 has acquired the lock ...
Writer-1 is writing 44 to sharedList
Writer-1 has finsihed writing
Writer-1 has released the lock ...

Writer-2 has acquired the lock ...
Writer-2 is writing 64 to sharedList
Writer-2 has finsihed writing
Writer-2 has released the lock ...

Writer-3 has acquired the lock ...
Writer-3 is writing 5 to sharedList
Writer-3 has finsihed writing
Writer-3 has released the lock ...

Writer-5 has acquired the lock ...
Writer-5 is writing 30 to sharedList
Writer-5 has finsihed writing
Writer-5 has released the lock ...

Writer-4 has acquired the lock ...
Writer-4 is writing 19 to sharedList
Writer-4 has finsihed writing
Writer-4 has released the lock ...

Reader-1 is reading 5 from sharedList
Reader-3 is reading 44 from sharedList
Reader-9 is reading 5 from sharedList
Reader-9 has finsihed reading

Reader-2 is reading 30 from sharedList
Reader-2 has finsihed reading

Reader-7 is reading 44 from sharedList
Reader-7 has finsihed reading

Reader-10 is reading 19 from sharedList
Reader-10 has finsihed reading

Reader-8 is reading 64 from sharedList
Reader-6 is reading 30 from sharedList
Reader-6 has finsihed reading

Reader-3 has finsihed reading

Reader-5 is reading 5 from sharedList
Reader-5 has finsihed reading

Reader-1 has finsihed reading

Reader-4 is reading 5 from sharedList
Reader-4 has finsihed reading

Reader-8 has finsihed reading

BUILD SUCCESSFUL (total time: 0 seconds)
```

```
run:
********* READER AND WRITER PROBLEM ********

Writer-3 has acquired the lock ...
Writer-3 is writing 13 to sharedList
Writer-3 has finsihed writing
Writer-3 has released the lock ...

Writer-2 has acquired the lock ...
Writer-2 is writing 19 to sharedList
Writer-2 has finsihed writing
Writer-2 has released the lock ...

Writer-1 has acquired the lock ...
Writer-1 is writing 63 to sharedList
Writer-1 has finsihed writing
Writer-1 has released the lock ...

Writer-4 has acquired the lock ...
Writer-4 is writing 99 to sharedList
Writer-4 has finsihed writing
Writer-4 has released the lock ...

Writer-5 has acquired the lock ...
Writer-5 is writing 3 to sharedList
Writer-5 has finsihed writing
Writer-5 has released the lock ...

Reader-1 is reading 3 from sharedList
Reader-3 is reading 13 from sharedList
Reader-9 is reading 99 from sharedList
Reader-9 has finsihed reading

Reader-2 is reading 63 from sharedList
Reader-2 has finsihed reading

Reader-7 is reading 13 from sharedList
Reader-7 has finsihed reading

Reader-8 is reading 19 from sharedList
Reader-8 has finsihed reading

Reader-10 is reading 99 from sharedList
Reader-10 has finsihed reading

Reader-6 is reading 63 from sharedList
Reader-6 has finsihed reading

Reader-5 is reading 19 from sharedList
Reader-5 has finsihed reading

Reader-3 has finsihed reading

Reader-4 is reading 19 from sharedList
Reader-4 has finsihed reading

Reader-1 has finsihed reading

BUILD SUCCESSFUL (total time: 0 seconds)
```

```
run:
********* READER AND WRITER PROBLEM ********

Writer-5 has acquired the lock ...
Writer-5 is writing 77 to sharedList
Writer-5 has finsihed writing
Writer-5 has released the lock ...

Writer-3 has acquired the lock ...
Writer-3 is writing 84 to sharedList
Writer-3 has finsihed writing
Writer-3 has released the lock ...

Writer-2 has acquired the lock ...
Writer-2 is writing 1 to sharedList
Writer-2 has finsihed writing
Writer-2 has released the lock ...

Writer-1 has acquired the lock ...
Writer-1 is writing 65 to sharedList
Writer-1 has finsihed writing
Writer-1 has released the lock ...

Writer-4 has acquired the lock ...
Writer-4 is writing 33 to sharedList
Writer-4 has finsihed writing
Writer-4 has released the lock ...

Reader-1 is reading 65 from sharedList
Reader-7 is reading 77 from sharedList
Reader-7 has finsihed reading

Reader-8 is reading 77 from sharedList
Reader-2 is reading 33 from sharedList
Reader-1 has finsihed reading

Reader-4 is reading 84 from sharedList
Reader-5 is reading 1 from sharedList
Reader-6 is reading 65 from sharedList
Reader-3 is reading 65 from sharedList
Reader-2 has finsihed reading

Reader-8 has finsihed reading

Reader-10 is reading 33 from sharedList
Reader-10 has finsihed reading

Reader-9 is reading 1 from sharedList
Reader-9 has finsihed reading

Reader-3 has finsihed reading

Reader-6 has finsihed reading

Reader-5 has finsihed reading

Reader-4 has finsihed reading

BUILD SUCCESSFUL (total time: 0 seconds)
```

```
run:
********* READER AND WRITER PROBLEM ********

Writer-4 has acquired the lock ...
Writer-4 is writing 96 to sharedList
Writer-4 has finsihed writing
Writer-4 has released the lock ...

Writer-1 has acquired the lock ...
Writer-1 is writing 33 to sharedList
Writer-1 has finsihed writing
Writer-1 has released the lock ...

Writer-5 has acquired the lock ...
Writer-5 is writing 5 to sharedList
Writer-5 has finsihed writing
Writer-5 has released the lock ...

Writer-3 has acquired the lock ...
Writer-3 is writing 13 to sharedList
Writer-3 has finsihed writing
Writer-3 has released the lock ...

Writer-2 has acquired the lock ...
Writer-2 is writing 74 to sharedList
Writer-2 has finsihed writing
Writer-2 has released the lock ...

Reader-1 is reading 74 from sharedList
Reader-3 is reading 13 from sharedList
Reader-4 is reading 5 from sharedList
Reader-4 has finsihed reading

Reader-2 is reading 13 from sharedList
Reader-3 has finsihed reading

Reader-5 is reading 74 from sharedList
Reader-5 has finsihed reading

Reader-1 has finsihed reading

Reader-2 has finsihed reading

Reader-6 is reading 74 from sharedList
Reader-6 has finsihed reading

Reader-9 is reading 5 from sharedList
Reader-9 has finsihed reading

Reader-10 is reading 13 from sharedList
Reader-10 has finsihed reading

Reader-7 is reading 96 from sharedList
Reader-7 has finsihed reading

Reader-8 is reading 74 from sharedList
Reader-8 has finsihed reading

BUILD SUCCESSFUL (total time: 0 seconds)
```

7. 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 that 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.**

4. Recommendations

None