Name: K Srikanth **Roll Number:** 17ETCS002124

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

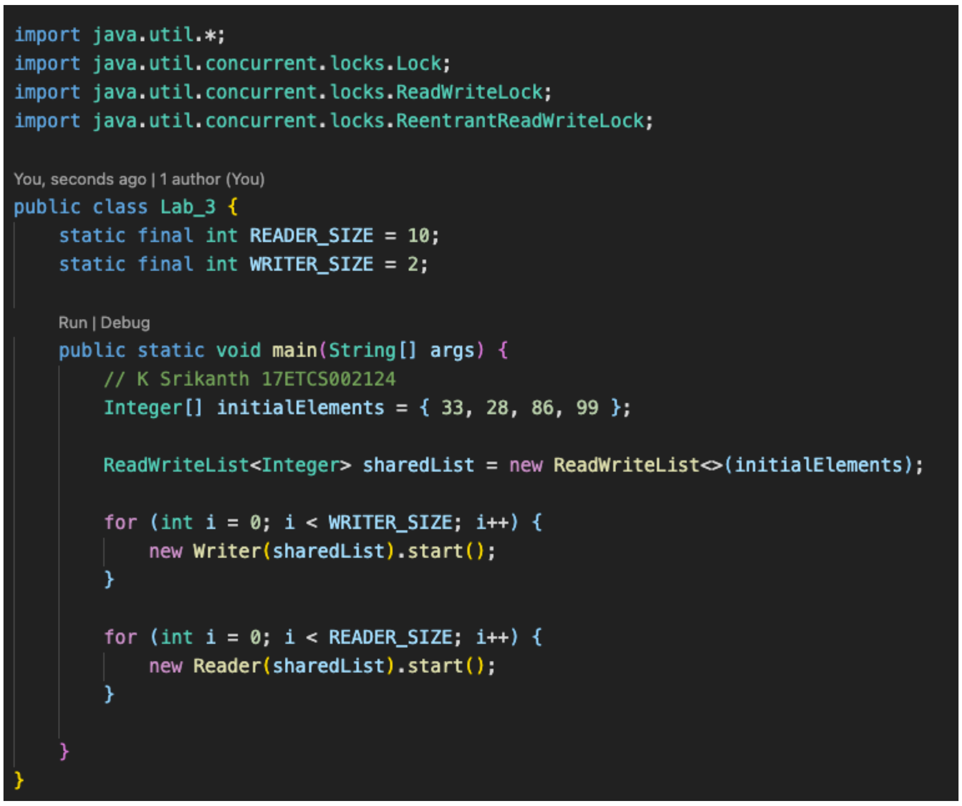
- A writer thread is responsible for randomly adds a number to the shared ReadWriteList list and reader thread is responsible for randomly gets an element from the shared list. The program creates and runs 10 reader threads and 5 writer threads that work on a shared ReadWriteList data structure.

5. Computations/Algorithms

1. **Start**
2. Declare the number of readers and writers required.
3. Declare the ReadWriteList sharedList. In the class that creates the shared list (ReadWriteList), initialize the re-entrant lock, readlock and writelock.
4. Declare a ‘for’ loop where you call the ‘Writers’ class creating a thread each time, eventually creating 5 threads.
5. In the ‘writers’ class, whenever the writer is writing an element into the shared list, the writelock is enabled and a random character is added to the list. When it is done, unlock the writelock and that thread is then put to sleep for some time.
6. Declare a ‘for’ loop where you call the ‘Readers’ class creating a thread each time, eventually creating 10 threads.
7. In the ‘readers’ class, whenever the reading is an element from the shared list, the readlock is enabled and a random character is read from the list. When it is done, the readlock is unlocked, giving access to other readers or writers and then, that thread is then put to sleep for some time.
8. **Stop**

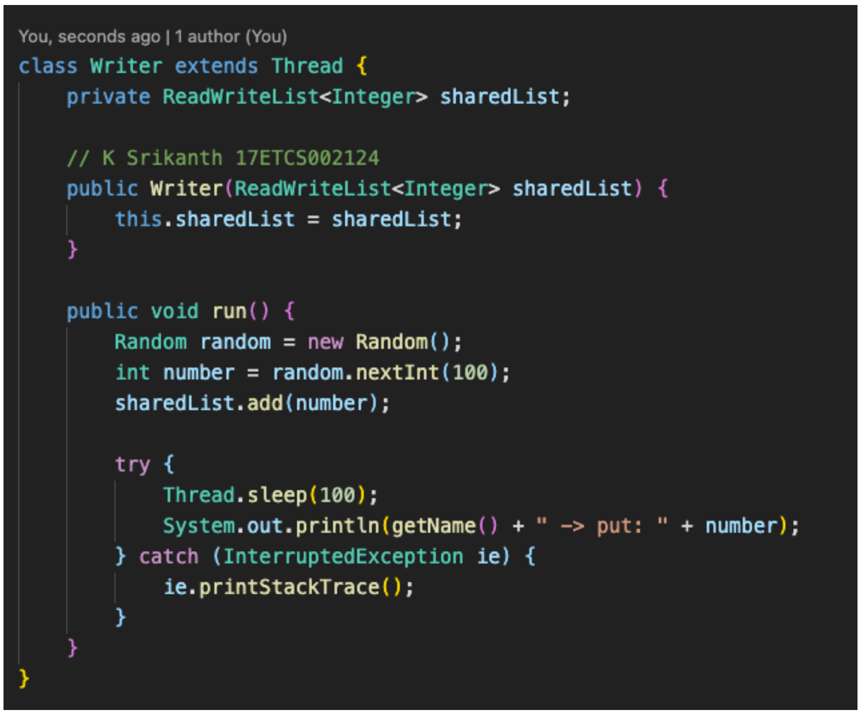
6. Presentation of Results

Java Program



```
import java.util.*;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReadWriteLock;
import java.util.concurrent.locks.ReentrantReadWriteLock;

You, seconds ago | 1 author (You)
public class Lab_3 {
    static final int READER_SIZE = 10;
    static final int WRITER_SIZE = 2;

    Run | Debug
    public static void main(String[] args) {
        // K Srikanth 17ETCS002124
        Integer[] initialElements = { 33, 28, 86, 99 };

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

        for (int i = 0; i < WRITER_SIZE; i++) {
            new Writer(sharedList).start();
        }

        for (int i = 0; i < READER_SIZE; i++) {
            new Reader(sharedList).start();
        }

    }
}
```

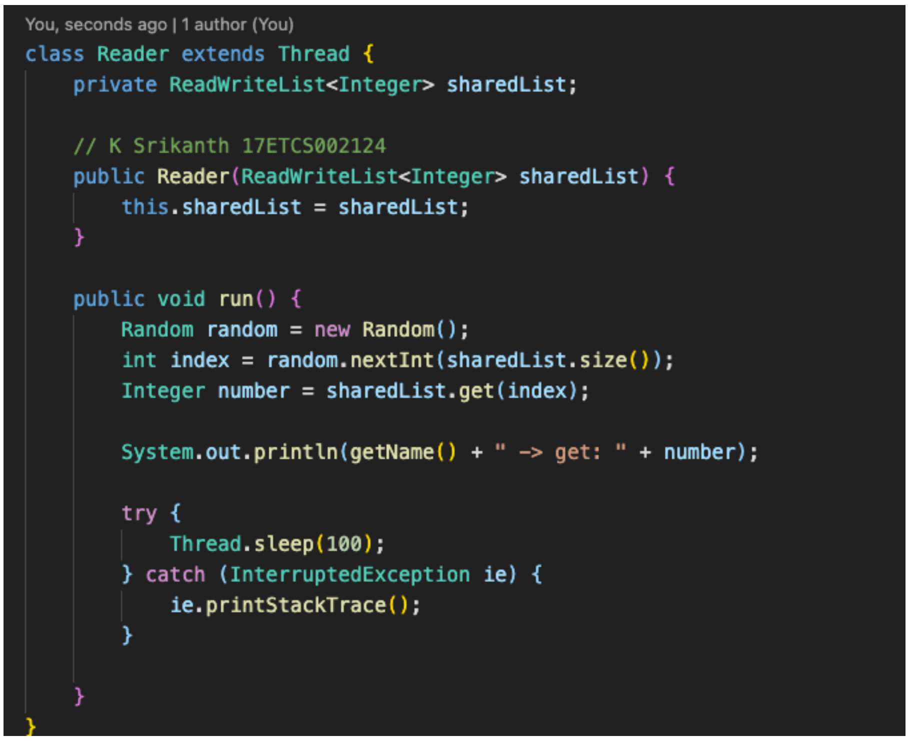
Figure 1 Java Program for given problem statement



```
You, seconds ago | 1 author (You)
class Writer extends Thread {
    private ReadWriteList<Integer> sharedList;

    // K Srikanth 17ETCS002124
    public Writer(ReadWriteList<Integer> sharedList) {
        this.sharedList = sharedList;
    }

    public void run() {
        Random random = new Random();
        int number = random.nextInt(100);
        sharedList.add(number);

        try {
            Thread.sleep(100);
            System.out.println(getName() + " -> put: " + number);
        } catch (InterruptedException ie) {
            ie.printStackTrace();
        }
    }
}
```

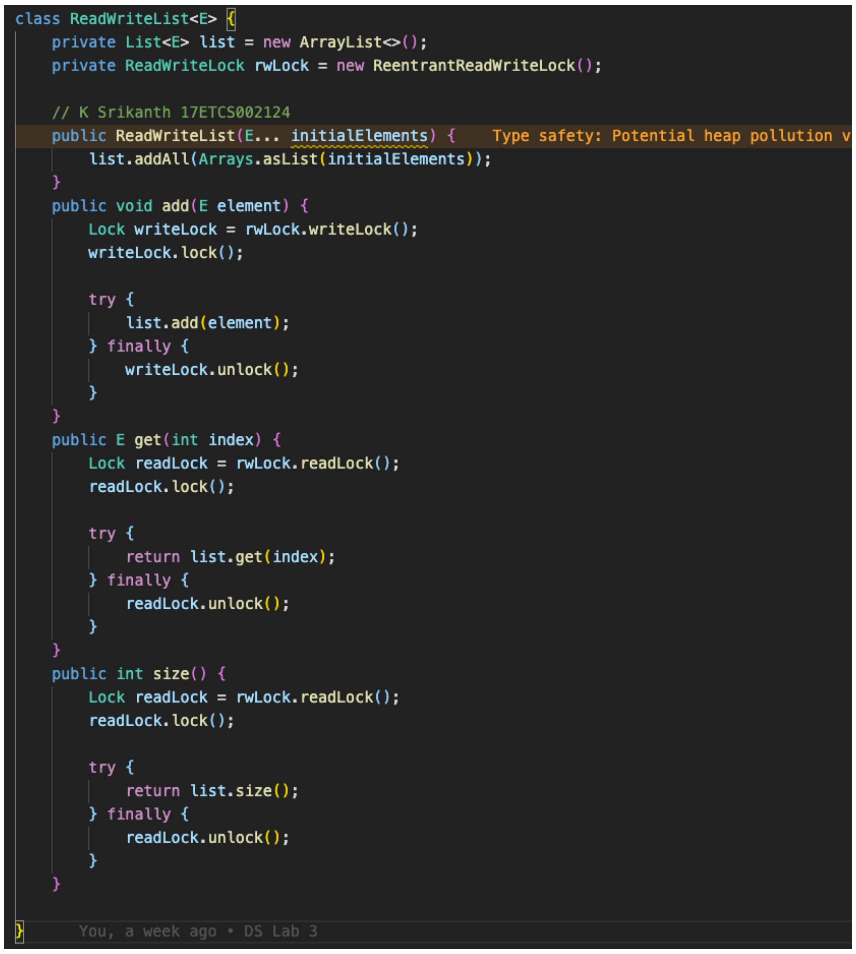
Figure 2 Java Program for given problem statement (Continued)

Name: K Srikanth **Roll Number:** 17ETCS002124



```
You, seconds ago | 1 author (You)
class Reader extends Thread {
    private ReadWriteList<Integer> sharedList;

    // K Srikanth 17ETCS002124
    public Reader(ReadWriteList<Integer> sharedList) {
        this.sharedList = sharedList;
    }

    public void run() {
        Random random = new Random();
        int index = random.nextInt(sharedList.size());
        Integer number = sharedList.get(index);

        System.out.println(getName() + " -> get: " + number);

        try {
            Thread.sleep(100);
        } catch (InterruptedException ie) {
            ie.printStackTrace();
        }

    }
}
```

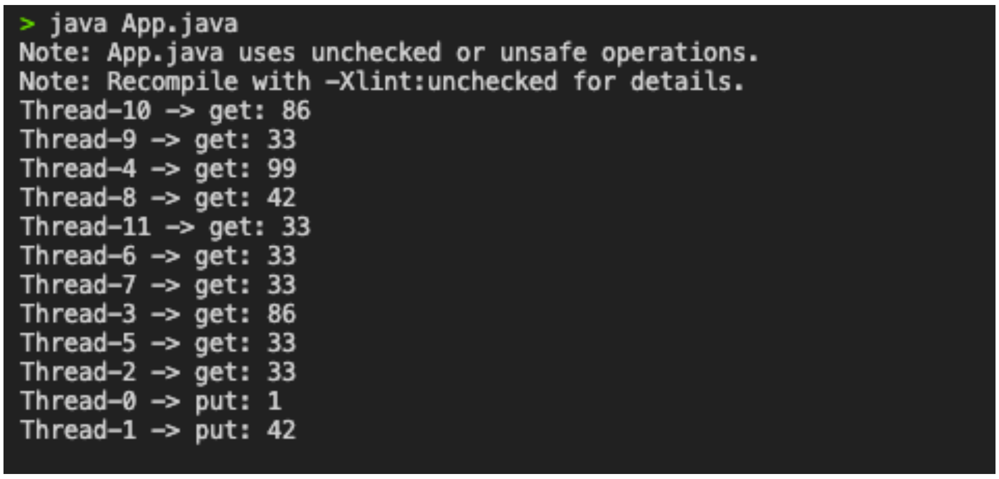
Figure 3 Java Program for given problem statement (Continued)



```
class ReadWriteList<E> {
    private List<E> list = new ArrayList<>();
    private ReadWriteLock rwLock = new ReentrantReadWriteLock();

    // K Srikanth 17ETCS002124
    public ReadWriteList(E... initialElements) {    Type safety: Potential heap pollution v
        list.addAll(Arrays.asList(initialElements));
    }
    public void add(E element) {
        Lock writeLock = rwLock.writeLock();
        writeLock.lock();

        try {
            list.add(element);
        } finally {
            writeLock.unlock();
        }
    }
    public E get(int index) {
        Lock readLock = rwLock.readLock();
        readLock.lock();

        try {
            return list.get(index);
        } finally {
            readLock.unlock();
        }
    }
    public int size() {
        Lock readLock = rwLock.readLock();
        readLock.lock();

        try {
            return list.size();
        } finally {
            readLock.unlock();
        }
    }
}       You, a week ago • DS Lab 3
```

Figure 4 Java Program for given problem statement (Continued)

Java Result



```
> java App.java
Note: App.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Thread-10 -> get: 86
Thread-9 -> get: 33
Thread-4 -> get: 99
Thread-8 -> get: 42
Thread-11 -> get: 33
Thread-6 -> get: 33
Thread-7 -> get: 33
Thread-3 -> get: 86
Thread-5 -> get: 33
Thread-2 -> get: 33
Thread-0 -> put: 1
Thread-1 -> put: 42
```

Figure 5 Java Program output for given problem statement

7. Analysis and Discussions

1. Limitations of Experiments

The given program will only have 10 reader and 5 writer threads and there is no generic order for the readers and writers to run (that is, it happens in a random order).

2. Limitations of Results

The writers can write only one element into the shared list.

3. Learning happened

We learnt about re-entrant locks.