# 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. **Computations/Algorithms**
2. **Start**
3. Declare the number of readers and writers required.
4. Declare the ReadWriteList sharedList. In the class that creates the shared list (ReadWriteList), initialize the re-entrant lock, readlock and writelock.
5. Declare a ‘for’ loop where you call the ‘Writers’ class creating a thread each time, eventually creating 5 threads.
6. 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.
7. Declare a ‘for’ loop where you call the ‘Readers’ class creating a thread each time, eventually creating 10 threads.
8. 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.
9. **Stop**
10. **Presentation of Results**

**Java Program**

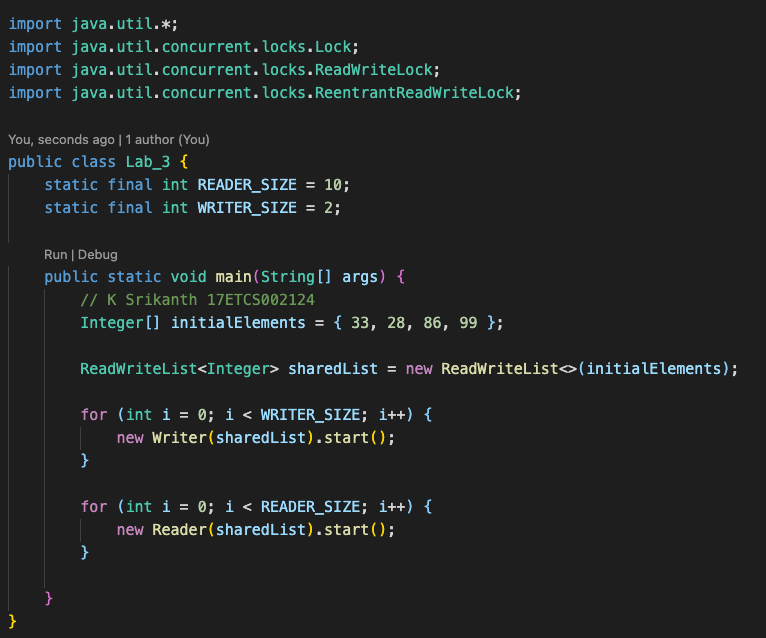
****

Figure 1 Java Program for given problem statement

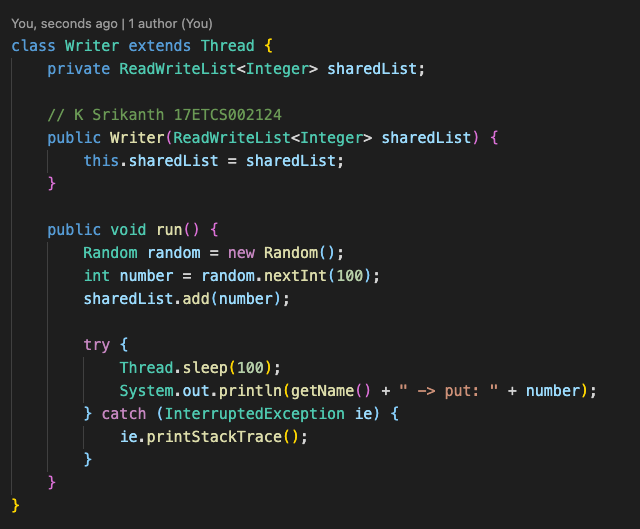
****

Figure 2 Java Program for given problem statement (Continued)

****

Figure 3 Java Program for given problem statement (Continued)

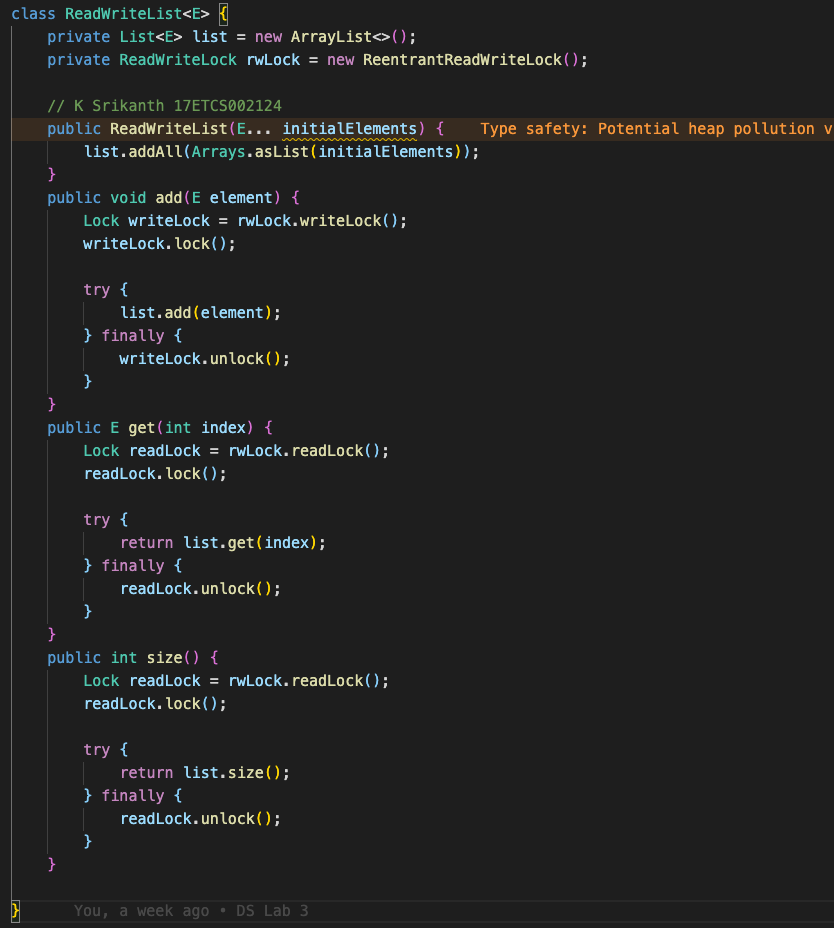
****

Figure 4 Java Program for given problem statement (Continued)

**Java Result**

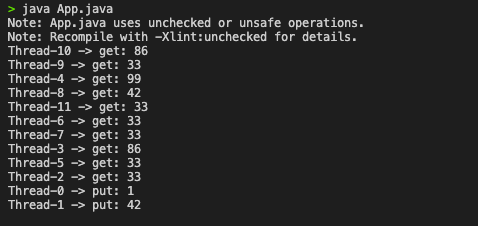
****

Figure 5 Java Program output for given problem statement

1. **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.