**M.S.Ramaiah University of Applied Sciences**

# Faculty of Engineering & Technology

# Lab Exam Answer Sheet – B. Tech.

**Department:** Computer Science and Engineering**Course:** B. Tech. in Computer Science and Engineering

**Subject Code:** 19CSL316A**Subject Title:** Distributed and Cloud Computing Lab

**Student Name**: K Srikanth

**Roll Number:** 17ETCS002124

**Section:** A Section

**Batch:** Batch-1

**Maximum Duration: 3 Hours Maximum Marks: 50**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No.** | **Item** | **Maximum Marks** | **Marks Obtained** |
| **a** | **Algorithm** | **5** |  |
| **b** | **Program + Results** | **7+3=10** |  |
| **c** | **Viva** | **10** |  |
|  | **Total** | **25** |  |

**Question:**

Develop a simple Java program to explain atomicity for two related variables

## Program Algorithm

* + - 1. **Start**
      2. **Initialize a thread class**
         1. Create an atomic variable say **“count\_1” with atomicinteger class.**
         2. Create another atomic variable say **“count\_2” with atomicinteger class.**
         3. Create a **Constructer** of the class.
         4. Create **a Runnable metho**d to run the threads for our class
         5. For Loop Begins: **condition ( i < 10)**

Count\_1. addAndGet(1);

Count\_2. decrementAndGet();

Exit

* + - 1. **Initialize a Main Class which throws an Expectation** 
         1. **Make an instance of the thread class**
         2. **Define thread “1”**
         3. **Define thread “2”**
         4. **Thread\_1.start()** // Which will start executing the thread 1
         5. **Thread\_2.start()** // Which will start executing the thread 2
         6. **Thread\_1.join()** //Now we join thread 1
         7. **Thread\_2.join()** //Now we join thread 2
         8. Compare the absolute value of the both the **atomic integers using .get() method**

If they are same then they have been updated correctly else they haven't been updated correctly.

* + - 1. **Stop**

## Execution and Testing

**Java Program**

****

Figure 1 Java Program for the given problem statement

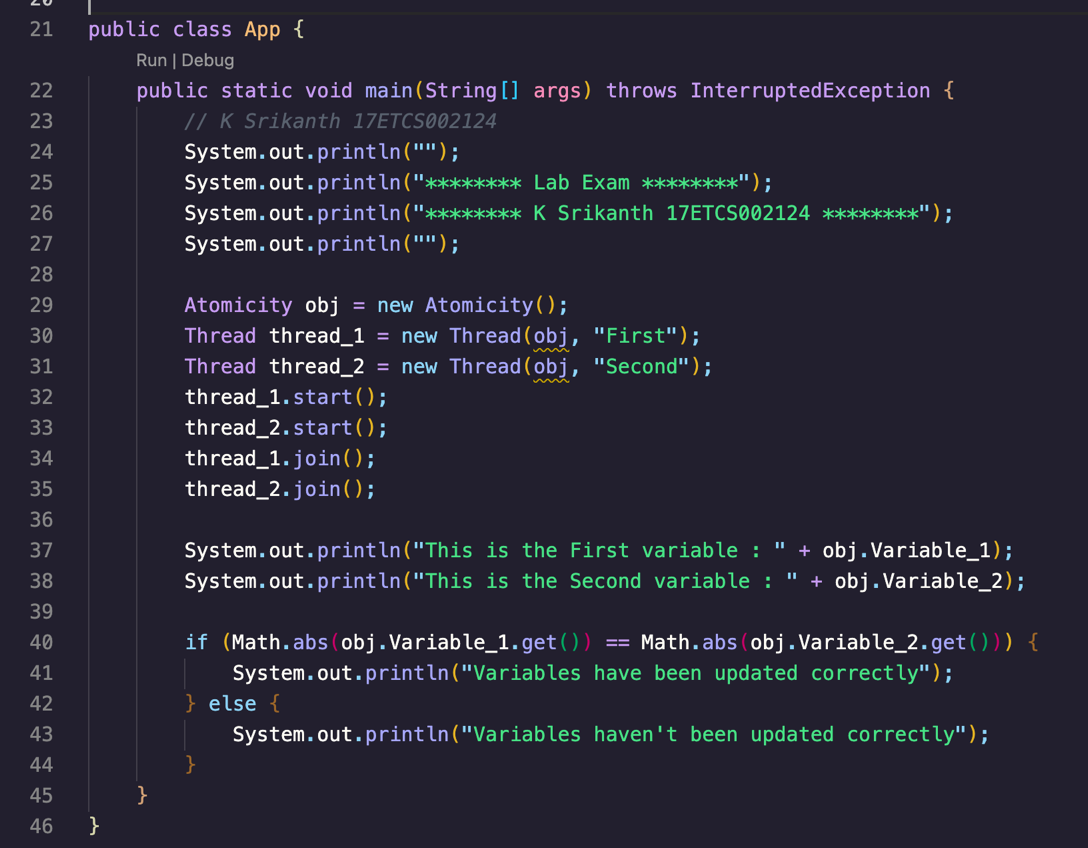
****

Figure 2 Java Program for the given problem statement Continued

1. **Results**

**Java Output**

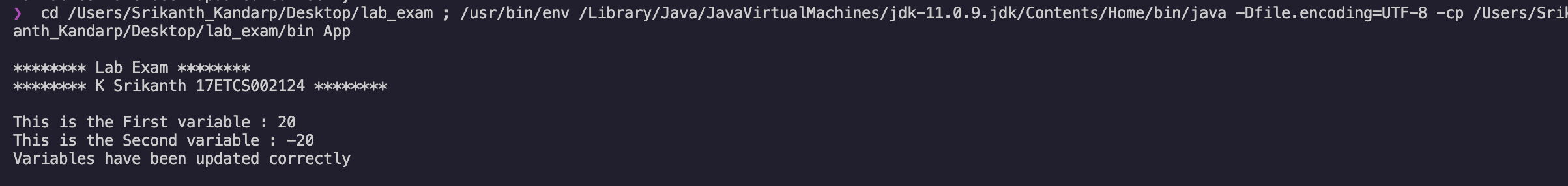
****

Figure 3 Java Program Output for the given problem statement

**Conclusion**

Successfully conducted the lab tasks without any errors. So when programming in a multi-threaded environment, we need to avoid situations in which concurrent execution of a set of operations may lead to incorrect or unexpected behaviour. So, we need to make these set of operations atomic. For operations on a single variable, we can achieve this by using the atomic variable classes, which offers us atomic operations on the variables, thus achieving correct behaviour in a multi-threaded environment.