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

1. **Program**

Algorithm

1. ***Start***
2. ***Initialize a thread class***
    - ***a.*** *Create an atomic variable say* ***"count_1" with atomicinteger class.***
    - ***b.*** *Create another atomic variable say* ***"count_2" with atomicinteger class.***
    - *c. Create a* ***Constructer*** *of the class.*
    - *d. Create* ***a Runnable metho****d to run the threads for our class*
    - *e. For Loop Begins:* ***condition (i < 10)***
        - *i. Count_1. addAndGet(1);*
        - *ii. Count_2. decrementAndGet();*
        - *iii. Exit*
3. ***Initialize a Main Class which throws an Expectation***
    - ***a. Make an instance of the thread class***
    - ***b. Define thread "1"***
    - ***c. Define thread "2"***
    - *d.* ***Thread_1.start()*** *// Which will start executing the thread 1*
    - *e.* ***Thread_2.start()*** *// Which will start executing the thread 2*
    - *f.* ***Thread_1.join()*** *//Now we join thread 1*
    - *g.* ***Thread_2.join()*** *//Now we join thread 2*
    - ***h.*** *Compare the absolute value of the both the* ***atomic integers using .get() method***
        - ***i.*** *If they are same then they have been updated correctly else they haven't been updated correctly.*
4. ***Stop***

2. Execution and Testing

Java Program

```java
import java.util.concurrent.atomic.AtomicInteger;

class Atomicity extends Thread {
    // K Srikanth 17ETCS002124
    AtomicInteger Variable_1;
    AtomicInteger Variable_2;

    Atomicity() {
        Variable_1 = new AtomicInteger();
        Variable_2 = new AtomicInteger();
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            Variable_1.addAndGet(1);
            Variable_2.decrementAndGet();
        }
    }
}
```

Figure 1 Java Program for the given problem statement

```java
public class App {
    Run | Debug
    public static void main(String[] args) throws InterruptedException {
        // K Srikanth 17ETCS002124
        System.out.println("");
        System.out.println("******** Lab Exam ********");
        System.out.println("******** K Srikanth 17ETCS002124 ********");
        System.out.println("");

        Atomicity obj = new Atomicity();
        Thread thread_1 = new Thread(obj, "First");
        Thread thread_2 = new Thread(obj, "Second");
        thread_1.start();
        thread_2.start();
        thread_1.join();
        thread_2.join();

        System.out.println("This is the First variable : " + obj.Variable_1);
        System.out.println("This is the Second variable : " + obj.Variable_2);

        if (Math.abs(obj.Variable_1.get()) == Math.abs(obj.Variable_2.get())) {
            System.out.println("Variables have been updated correctly");
        } else {
            System.out.println("Variables haven't been updated correctly");
        }
    }
}
```

Figure 2 Java Program for the given problem statement Continued

3. Results

Java Output

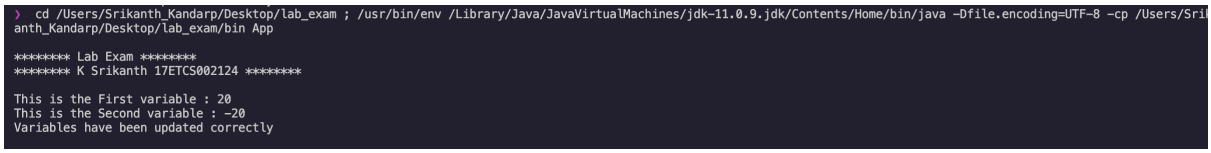

Figure 3 Java Program Output for the given problem statement

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.