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

Laboratory 4

Title of the Laboratory Exercise: Producer Consumer problem

1. Introduction and Purpose of Experiment

In computing, the producer-consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue.

- The producer's job is to generate data, put it into the buffer, and start again.
- At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.

Aim and Objectives

Aim

- To develop programs using monitors in Java

2. 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

3. Question

Create a multithreaded program in Java

The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

4. Computations/Algorithms

Flow Chart

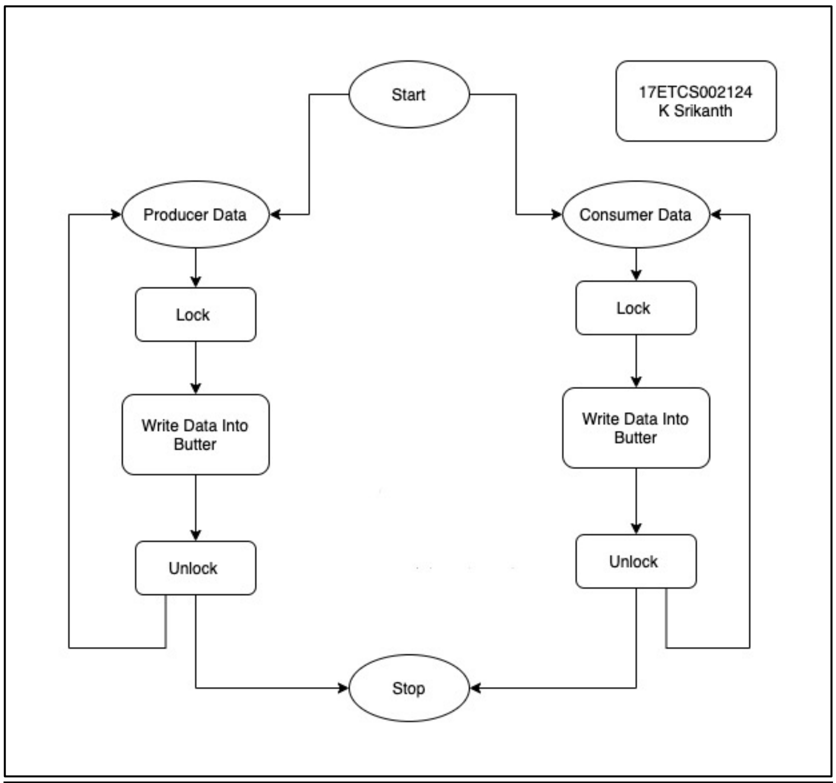
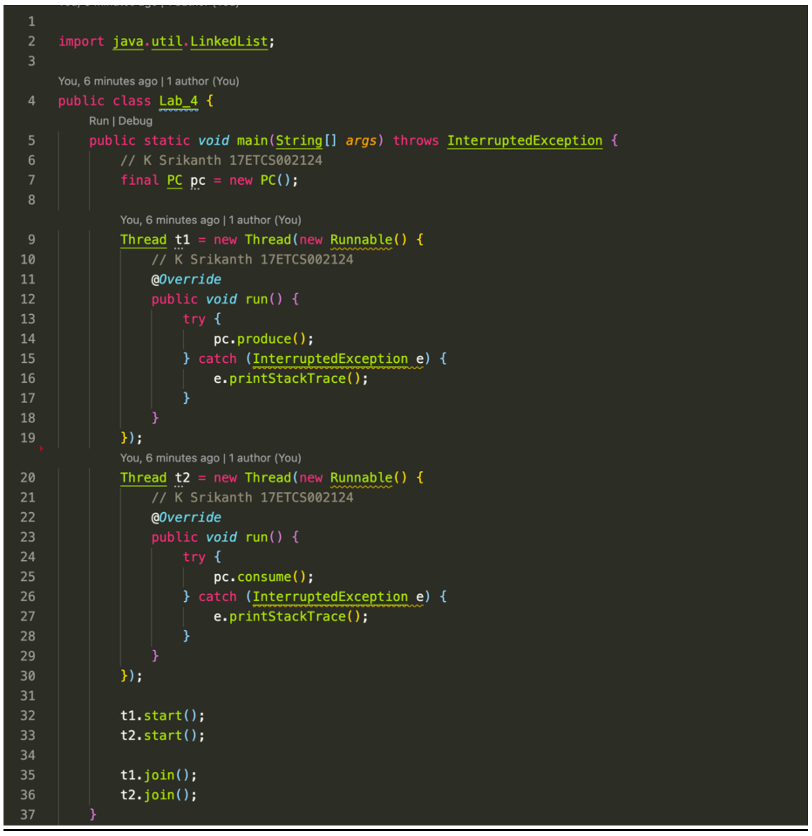


Figure 1 Flowchart of given problem statement

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

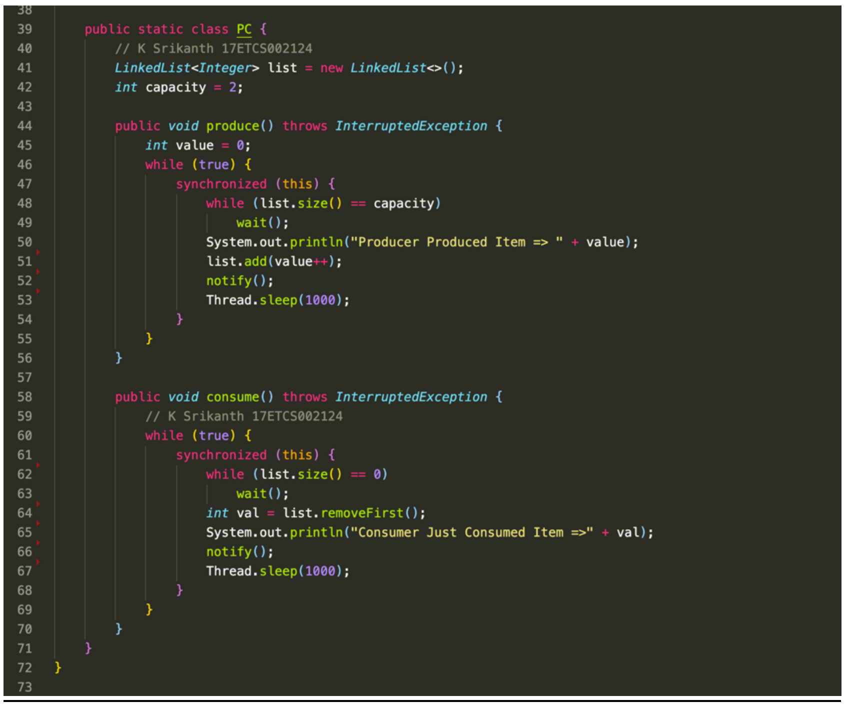
5. Presentation of Results

Code



```java

import java.util.LinkedList;

You, 6 minutes ago | 1 author (You)
public class Lab_4 {
    Run | Debug
    public static void main(String[] args) throws InterruptedException {
        // K Srikanth 17ETCS002124
        final PC pc = new PC();

        You, 6 minutes ago | 1 author (You)
        Thread t1 = new Thread(new Runnable() {
            // K Srikanth 17ETCS002124
            @Override
            public void run() {
                try {
                    pc.produce();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        You, 6 minutes ago | 1 author (You)
        Thread t2 = new Thread(new Runnable() {
            // K Srikanth 17ETCS002124
            @Override
            public void run() {
                try {
                    pc.consume();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        t1.start();
        t2.start();

        t1.join();
        t2.join();
    }
```

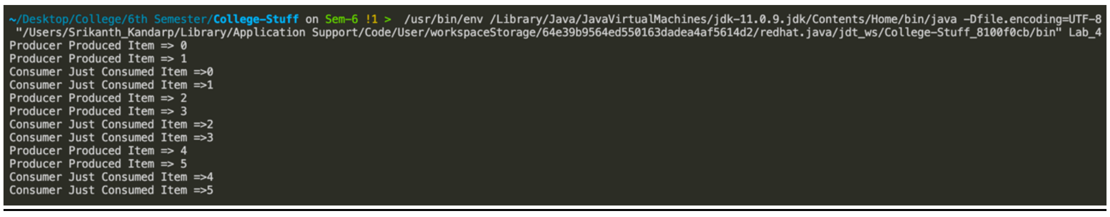
Figure 2 Java Program for the given problem statement



```java

    public static class PC {
        // K Srikanth 17ETCS002124
        LinkedList<Integer> list = new LinkedList<>();
        int capacity = 2;

        public void produce() throws InterruptedException {
            int value = 0;
            while (true) {
                synchronized (this) {
                    while (list.size() == capacity)
                        wait();
                    System.out.println("Producer Produced Item => " + value);
                    list.add(value++);
                    notify();
                    Thread.sleep(1000);
                }
            }
        }

        public void consume() throws InterruptedException {
            // K Srikanth 17ETCS002124
            while (true) {
                synchronized (this) {
                    while (list.size() == 0)
                        wait();
                    int val = list.removeFirst();
                    System.out.println("Consumer Just Consumed Item =>" + val);
                    notify();
                    Thread.sleep(1000);
                }
            }
        }
    }
}

```

Figure 3 Java Program for the given problem statement continued

Output



```
~/Desktop/College/6th Semester/College-Stuff on Sem-6 !1 >  /usr/bin/env /Library/Java/JavaVirtualMachines/jdk-11.0.9.jdk/Contents/Home/bin/java -Dfile.encoding=UTF-8
 "/Users/Srikanth_Kandarp/Library/Application Support/Code/User/workspaceStorage/64e39b9564ed550163dadea4af5614d2/redhat.java/jdt_ws/College-Stuff_8100f0cb/bin" Lab_4
Producer Produced Item => 0
Producer Produced Item => 1
Consumer Just Consumed Item =>0
Consumer Just Consumed Item =>1
Producer Produced Item => 2
Producer Produced Item => 3
Consumer Just Consumed Item =>2
Consumer Just Consumed Item =>3
Producer Produced Item => 4
Producer Produced Item => 5
Consumer Just Consumed Item =>4
Consumer Just Consumed Item =>5
```

Figure 4 Java Program output for the given problem statement

6. Analysis and Discussions

Multi-threaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results.
So it needs to be made sure by some synchronization method that only one thread can access the resource at a given point of time.
Java provides a way of creating threads and synchronizing their task by using synchronized blocks. Synchronized blocks in Java are marked with the synchronized keyword. A synchronized block in Java is synchronized on some object. All synchronized blocks synchronized on the same object can only have one thread executing inside them at a time. All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits the block.

Syntax

```
synchronized(sync_object)
{ // Access shared variables and other
  // shared resources }
```

1. Limitations of Experiments

The producer should produce data only when the buffer is not full. ... If the buffer is empty, then the consumer shouldn't be allowed to take any data from the buffer. The producer and consumer should not access the buffer at the same time.

2. Limitations of Results

- Increase the waiting time of the thread
- Create performance problem

3. Learning happened

- Learned about how synchronized methods work in Java