DAY 23:

ASSIGNMENT 3:

Task 3: Synchronization and Inter-thread Communication

Implement a producer-consumer problem using wait() and notify() methods to handle the correct processing sequence between threads.

ANSWER:

Producer-Consumer Problem:

- Producer: Generates data and puts it into a shared buffer.

- Consumer: Takes data from the shared buffer and processes it.

- The buffer has limited capacity, so the producer must wait if the buffer is full, and the consumer must wait if the buffer is empty.

Implementation:

java

```
import java.util.LinkedList;

import java.util.Queue;


class Buffer {

   private final Queue<Integer> queue = new LinkedList<>();

   private final int capacity;


   public Buffer(int capacity) {

      this.capacity = capacity;

   }


   public synchronized void produce(int value) throws InterruptedException {

      while (queue.size() == capacity) {

         wait(); // Wait if the buffer is full
```

```
        }
        queue.offer(value);
        System.out.println("Produced: " + value);
        notifyAll(); // Notify consumers that they can consume
    }

    public synchronized int consume() throws InterruptedException {
        while (queue.isEmpty()) {
            wait(); // Wait if the buffer is empty
        }
        int value = queue.poll();
        System.out.println("Consumed: " + value);
        notifyAll(); // Notify producers that they can produce
        return value;
    }
}

class Producer implements Runnable {
    private final Buffer buffer;

    public Producer(Buffer buffer) {
        this.buffer = buffer;
    }

    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            try {
                buffer.produce(i);
                Thread.sleep(500); // Simulate time taken to produce an item
            } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
    }
  }
}

class Consumer implements Runnable {
  private final Buffer buffer;

  public Consumer(Buffer buffer) {
    this.buffer = buffer;
  }

  @Override
  public void run() {
    for (int i = 0; i < 10; i++) {
      try {
        buffer.consume();
        Thread.sleep(1000); // Simulate time taken to consume an item
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
}

public class ProducerConsumerDemo {
  public static void main(String[] args) {
    Buffer buffer = new Buffer(5);

    Thread producerThread = new Thread(new Producer(buffer));
```

```
        Thread consumerThread = new Thread(new Consumer(buffer));

        producerThread.start();
        consumerThread.start();

        try {
            producerThread.join();
            consumerThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Producer and Consumer have finished their tasks.");
    }
}
```

Explanation:

1. Buffer Class:

- A shared buffer with a fixed capacity is implemented using a Queue.
- The produce method adds items to the buffer. If the buffer is full, the producer thread waits.
- The consume method removes items from the buffer. If the buffer is empty, the consumer thread waits.
- Both methods use synchronized to ensure mutual exclusion and wait()/notifyAll() to coordinate the producer and consumer threads.

2. Producer Class:

- Implements the Runnable interface.
- Produces items and adds them to the buffer. It simulates the time taken to produce an item using Thread.sleep(500).

3. Consumer Class:

- Implements the Runnable interface.

- Consumes items from the buffer. It simulates the time taken to consume an item using Thread.sleep(1000).

4. ProducerConsumerDemo Class:

- Creates instances of the Buffer, Producer, and Consumer classes.
- Starts the producer and consumer threads.
- Waits for both threads to finish using join().

Running the Program:

When you run this program, you will see the producer and consumer threads producing and consuming items in sequence. The wait() and notifyAll() methods ensure that the producer waits if the buffer is full and the consumer waits if the buffer is empty, maintaining the correct processing sequence.