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.

```
package com.wipro.model;

import java.util.LinkedList;

import java.util.Queue;

class Buffer {

private Queue<Integer> queue;

private int capacity;

public Buffer(int capacity) {

this.capacity = capacity;

this.queue = new LinkedList<>();

}

public synchronized void produce(int item) throws InterruptedException {

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

System.out.println("Buffer is full. Producer is waiting...");

wait();

}

queue.add(item);

System.out.println("Produced: " + item);

notifyAll();

}

public synchronized int consume() throws InterruptedException {

while (queue.isEmpty()) {

System.out.println("Buffer is empty. Consumer is waiting...");

wait();

}

int item = queue.remove();

System.out.println("Consumed: " + item);

notifyAll();

return item;

}
```

```
}
class Producer extends Thread {
private Buffer buffer;
private int number;
public Producer(Buffer buffer, int number) {
this.buffer = buffer;
this.number = number;
}
public void run() {
try {
for (int i = 1; i <= 5; i++) {
buffer.produce(i);
Thread.sleep(1000);
}
} catch (InterruptedException e) {
e.printStackTrace();
}
}
}
class Consumer extends Thread {
private Buffer buffer;
public Consumer(Buffer buffer) {
this.buffer = buffer;
}
public void run() {
try {
for (int i = 1; i <= 5; i++) {
int item = buffer.consume();
Thread.sleep(2000);
}
} catch (InterruptedException e) {
```

```
e.printStackTrace();
}
}
}
public class ProducerConsumerDemo {
public static void main(String[] args) {
Buffer buffer = new Buffer(2);
Producer producer = new Producer(buffer, 1);
Consumer consumer = new Consumer(buffer);
producer.start();
consumer.start();
}
}
```