Code:-

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Semaphore;

class ProducerConsumer {
    private final Queue<Integer> queue = new LinkedList<>();
    private final int capacity;
    private final Semaphore empty; // Counts empty slots
    private final Semaphore full;  // Counts filled slots
    private final Object mutex = new Object(); // Mutex for critical section

    public ProducerConsumer(int capacity) {
        this.capacity = capacity;
        this.empty = new Semaphore(capacity); // Initially, all slots are empty
        this.full = new Semaphore(0); // Initially, no slots are filled
    }

    public void produce(int item) throws InterruptedException {
        empty.acquire(); // Wait for an empty slot
        synchronized (mutex) {
            queue.add(item);
            System.out.println("Produced: " + item);
        }
        full.release(); // Increase the count of filled slots
```

```
    }

    public int consume() throws InterruptedException {
        full.acquire(); // Wait for a filled slot
        int item;
        synchronized (mutex) {
            item = queue.poll();
            System.out.println("Consumed: " + item);
        }
        empty.release(); // Increase the count of empty slots
        return item;
    }
}

class Producer implements Runnable {
    private final ProducerConsumer pc;

    public Producer(ProducerConsumer pc) {
        this.pc = pc;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < 5; i++) {
                pc.produce(i);
                Thread.sleep(100); // Simulate time taken to produce
```

```
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}

class Consumer implements Runnable {
    private final ProducerConsumer pc;

    public Consumer(ProducerConsumer pc) {
        this.pc = pc;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < 5; i++) {
                pc.consume();
                Thread.sleep(150); // Simulate time taken to consume
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

```
public class ProducerConcumer {
    public static void main(String[] args) {
        int capacity = 5; // Capacity of the buffer
        ProducerConsumer pc = new ProducerConsumer(capacity);

        ExecutorService executor = Executors.newFixedThreadPool(2);
        executor.execute(new Producer(pc));
        executor.execute(new Consumer(pc));

        executor.shutdown();
    }
}
```

Output:-

```
Console ×
<terminated> ProducerConcumer [Java
Produced: 0
Consumed: 0
Produced: 1
Consumed: 1
Produced: 2
Produced: 3
Consumed: 2
Produced: 4
Consumed: 3
Consumed: 4
```