import java.util.LinkedList;

import java.util.Queue;

import java.util.concurrent.Semaphore;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

class ProducerConsumer {

private static final int BUFFER\_SIZE = 5;

private static final int MAX\_ITEMS = 10;

private Queue<Integer> buffer = new LinkedList<>();

private Semaphore empty = new Semaphore(BUFFER\_SIZE);

private Semaphore full = new Semaphore(0);

private Lock mutex = new ReentrantLock();

private int producedItems = 0;

private int consumedItems = 0;

class Producer extends Thread {

@Override

public void run() {

try {

while (true) {

mutex.lock();

if (producedItems >= MAX\_ITEMS) {

mutex.unlock();

break;

}

mutex.unlock();

int item = produceItem();

empty.acquire();

mutex.lock();

buffer.add(item);

producedItems++;

System.out.println("Producer produced: " + item + " (Total produced: " + producedItems + ")");

mutex.unlock();

full.release();

Thread.sleep((int) (Math.random() \* 1000));

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

private int produceItem() {

return (int) (Math.random() \* 100);

}

}

class Consumer extends Thread {

@Override

public void run() {

try {

while (true) {

mutex.lock();

if (consumedItems >= MAX\_ITEMS) {

mutex.unlock();

break;

}

mutex.unlock();

full.acquire();

mutex.lock();

int item = buffer.remove();

consumedItems++;

System.out.println("Consumer consumed: " + item + " (Total consumed: " + consumedItems + ")");

mutex.unlock();

empty.release();

Thread.sleep((int) (Math.random() \* 1500));

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public static void main(String[] args) {

ProducerConsumer pc = new ProducerConsumer();

Producer producer = pc.new Producer();

Consumer consumer = pc.new Consumer();

producer.start();

consumer.start();

try {

producer.join();

consumer.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}