import java.util.Scanner;

import java.util.concurrent.\*;

import java.util.concurrent.locks.\*;

class Q {

int item;

private final Lock lock = new ReentrantLock();

private final Condition condition = lock.newCondition();

private boolean hasItem = false;

private volatile boolean terminate = false;

void get() {

lock.lock();

try {

while (!hasItem)

{

condition.await();

}

if (!terminate)

{

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

}

hasItem = false;

condition.signal();

} catch (InterruptedException e)

{

Thread.currentThread().interrupt();

} finally

{

lock.unlock();

}

}

void put(int item)

{

lock.lock();

try {

while (hasItem)

{

condition.await();

}

this.item = item;

if (!terminate)

{

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

}

hasItem = true;

condition.signal();

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

} finally {

lock.unlock();

}

}

public void terminate() {

terminate = true;

lock.lock();

try {

condition.signalAll();

} finally {

lock.unlock();

}

}

public boolean isTerminated() {

return terminate;

}

}

class MProd implements Runnable {

Q q;

MProd(Q q) {

this.q = q;

new Thread(this, "Mutex Producer").start();

}

public void run() {

while (!q.isTerminated()) {

int item = (int) (Math.random() \* 100); // Produce a random item

q.put(item);

try {

Thread.sleep(500); // Adding a delay to simulate production time

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

}

class MCon implements Runnable {

Q q;

MCon(Q q) {

this.q = q;

new Thread(this, "Mutex Consumer").start();

}

public void run() {

while (!q.isTerminated()) {

q.get();

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

}

class ProducerConsumer {

private static BlockingQueue<Integer> Buffer = new LinkedBlockingDeque<>();

private static Semaphore emptySlots = new Semaphore(10);

private static Semaphore fullSlots = new Semaphore(0);

private volatile boolean terminate = false;

class SProd implements Runnable {

@Override

public void run() {

try {

while (!terminate) {

int item = produceItem();

emptySlots.acquire();

Buffer.put(item);

fullSlots.release();

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

private int produceItem() {

int num = (int) (Math.random() \* 100);

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

return num;

}

}

class SCon implements Runnable {

@Override

public void run() {

try {

while (!terminate) {

fullSlots.acquire();

int item = Buffer.take();

emptySlots.release();

consumeItem(item);

Thread.sleep(1000);

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

private void consumeItem(int item) {

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

}

}

public void terminate() {

terminate = true;

}

public boolean isTerminated() {

return terminate;

}

}

public class B1

{

public static void main(String args[]) throws InterruptedException {

Scanner sc = new Scanner(System.in);

int choice = 0;

while (choice != 3) {

System.out.println("Select Operation\n1-Mutex\n2-Semaphore\n3-Exit");

choice = sc.nextInt();

if (choice == 1) {

Q q = new Q();

Thread consumerThread = new Thread(new MCon(q));

Thread producerThread = new Thread(new MProd(q));

consumerThread.start();

producerThread.start();

System.out.println("Press 'e' to exit back to main menu");

while (!q.isTerminated()) {

if (sc.next().equalsIgnoreCase("e")) {

q.terminate();

consumerThread.interrupt();

producerThread.interrupt();

break;

}

}

consumerThread.join();

producerThread.join();

} else if (choice == 2) {

ProducerConsumer pc = new ProducerConsumer();

Thread producerThread = new Thread(pc.new SProd());

Thread consumerThread = new Thread(pc.new SCon());

producerThread.start();

consumerThread.start();

System.out.println("Press 'e' to exit back to main menu");

while (!pc.isTerminated()) {

if (sc.next().equalsIgnoreCase("e")) {

pc.terminate();

producerThread.interrupt();

consumerThread.interrupt();

break;

}

}

producerThread.join();

consumerThread.join();

} else if (choice == 3) {

System.out.println("Exiting...");

break;

}

}

sc.close();

}

}