// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// \*\*\*\*\*\*\*\*\* Demonstration of a FIFO queue operation \*\*\*\*\*\*\*\*\*\*//

// \*\*\*\*\*\*\*\*\* by displaying the current queue content \*\*\*\*\*\*\*\*\*\*\*//

// \*\*\*\*\*\*\* and a histogram of the queuing delay \*\*\*\*\*\*\*\*\*\*//

// \* Changes on the original FIFO code are indicated in yellow \*//

import java.util.\*;

import java.awt.\*;

import java.awt.event.\*;

class Packet {

 long time;

 int id, priority, length; // packet ID (serial number), priority and length (in transmission time units)

 // They are accessed through appropriate set and get functions.

 public void Packet() {} // empty constructor

 public long getTime() {

 return time; }

 public void setTime(long time) {

 this.time=time; }

 public int getId() {

 return id; }

 public void setId(int id) {

 this.id=id; }

 public int getPriority() {

 return priority; }

 public void setPriority(int priority) {

 this.priority=priority; }

 public int getLength() {

 return length; }

 public void setLength(int length) {

 this.length=length; }

 }

class Queue {

 int length, n;

 FifoQueue fifoq;

 DrawHistogram drawHistog;

 LinkedList ll = new LinkedList(); // the queue is implemented as a dyhnamic array (LinkedList)

 boolean full, empty;

 long startTime; // the start time (obtained from the system) is passed to the queue to serve as the time base.

 double delay;

 int[] histogram;

 int histogramSize;

 Queue(int length, long startTime, FifoQueue fifoq, DrawHistogram drawHistog) {

 this.length = length;

 this.startTime=startTime;

 this.fifoq = fifoq;

 this.drawHistog = drawHistog;

 histogramSize = 20;

 histogram = new int[histogramSize];

 for (int i=0;i<histogramSize;i++)

 histogram[i]=0;

 empty=true;

 full=false;

 }

 synchronized Packet get() {

 Packet pckt;

 long time;

 if(empty)

 try {

 wait();

 } catch(InterruptedException e) {

 System.out.println("InterruptedException caught");

 }

 pckt=(Packet)ll.removeFirst();

 time=System.currentTimeMillis( )-startTime;

 delay=time-pckt.getTime();

 double t=0;

 int flag1 = 0;

 for(int x=0; x<(histogramSize-1); x++, t += 50)

 if(delay < t) {

 histogram[x]++; flag1=1;

 break; }

 if(flag1==0) histogram[histogramSize-1]++;

// upodate histogram display

 drawHistog.updateHistog(this);

//

System.out.println("Got: " + pckt.getId() + " time: " + time + " delay: " + delay);

 n=ll.size();

 fifoq.updateFrame(n);

 if(n==0) empty=true;

 if(n==length-1) {

 full=false;

 notify();

 }

 return pckt;

 }

 synchronized void put(Packet pckt) {

 if(full)

 try {

 wait();

 } catch(InterruptedException e) {

 System.out.println("InterruptedException caught");

 }

 ll.addLast(pckt);

 System.out.println("Put: " + pckt.getId() + " time: " + (System.currentTimeMillis()-startTime));

 n=ll.size();

 fifoq.updateFrame(n);

 if(n==length) full=true;

 if(n==1) {

 empty=false;

 notify();

 }

 }

 }

class PacketProducer implements Runnable {

 Queue queue;

 Packet pckt;

 Random r1, r2; // Two Random objects will be used as pseudo-random number generators.

 // one for the packet length (transmission time) and the other fpor the interarrival time

 int timeScale;

 int pN; // number of packets to be produced

 long startTime;

 double load; // normalized load

 Thread t;

 PacketProducer (int packetNum, double load, int timeScale, Queue queue, long startTime) {

 this.load=load;

 this.timeScale = timeScale;

 pN = packetNum;

 this.startTime=startTime;

 this.queue=queue;

 r1 = new Random();

 r2 = new Random();

 t=new Thread(this, "Producer");

 t.start();

 }

 public void run() {

 int i;

 for(i=1;i<pN;i++) {

 pckt=new Packet();

 pckt.setTime(System.currentTimeMillis( )-startTime);

 pckt.setId(i);

 pckt.setPriority(0); // just a single priority

 pckt.setLength(r1.nextInt((int)(load\*2\*timeScale)));

// pckt.setLength((int)(load\*timeScale)); // try this to adjust the time scale

 queue.put(pckt);

 try {

 t.sleep(r2.nextInt(2\*timeScale));

// t.sleep(timeScale); // try this to adjust the time scale

 } catch(InterruptedException e) {

 System.out.println("Interrupted");

 }

 }

 }

 }

class PacketConsumer implements Runnable {

 int pN; // number of packets to be consumed

 Queue queue;

 Packet pckt;

 Thread t;

 PacketConsumer(int packetNum, Queue queue) {

 pN = packetNum;

 this.queue=queue;

 t=new Thread(this, "Consumer");

 t.start();

 }

 public void run() {

 int i;

 for(i=1;i<pN;i++) {

 pckt=queue.get();

 try {

 t.sleep(pckt.getLength());

 } catch(InterruptedException e) {

 System.out.println("Interrupted");

 }

 }

 }

 }

class DrawHistogram extends Frame {

 int[] h;

 int histSize;

 Queue q;

 public DrawHistogram () {

 histSize = 20;

 h = new int[histSize];

 for(int i=0; i<histSize; i++) {

 h[i]=0;

 }

 }

 public void paint(Graphics g) {

 for(int i=0; i<histSize; i++) {

 g.fillRect(i\*10, 600-h[i]\*5, 10, h[i]\*5);

 }

 }

 public void updateHistog(Queue q) {

 this.q = q;

 histSize = q.histogramSize;

 System.arraycopy(q.histogram, 0, h, 0, histSize);

 repaint();

 }

 }

 public class FifoQueue extends Frame {

 static long startTime=System.currentTimeMillis( );

 int bufferSize;

 int n;

 public FifoQueue(int buf, int len) {

 bufferSize = buf;

 n = len;

 addWindowListener(new MyWindowAdapter());

 }

 public void paint(Graphics g) {

 g.fillRect(20,20,10\*n,30);

 g.drawRect(20, 20, 10\*bufferSize,30);

 }

 public void updateFrame(int n) {

 this.n = n;

 repaint();

 }

 public static void main(String args[]) {

 int bufferSize = 10;

 int timeScale = 1000; // choose ia suitable value for a desired real-time accuracy with your system

 int packetNum = 501;

 double load = 0.8;

 int len=0;

 DrawHistogram drawHistog = new DrawHistogram();

 drawHistog.setSize(new Dimension(300,600));

 drawHistog.setTitle("Queueing delay histogram" );

 drawHistog.setVisible(true);

 FifoQueue fifoq = new FifoQueue(bufferSize, len);

 fifoq.setSize(new Dimension(300,200));

 fifoq.setTitle("Fifo Queue display window");

 fifoq.setVisible(true);

 Queue queue = new Queue(bufferSize, startTime, fifoq, drawHistog);

 PacketProducer p = new PacketProducer(packetNum, load, timeScale, queue, startTime);

 PacketConsumer c = new PacketConsumer(packetNum, queue);

 try {

 p.t.join();

 c.t.join();

 } catch(InterruptedException e) {

 System.out.println("Interrupted in join");

 }

 for(int i=0; i<20; i++) {

 for(int x=queue.histogram[i]; x>0; x--)

 System.out.print("\*");

 System.out.println();

 }

 }

 }

 class MyWindowAdapter extends WindowAdapter {

 public void windowClosing(WindowEvent we) {

 System.exit(0);

 }

 }