```
import java.util.*;

/**
 * Silly demonstration of thread troubles. Putters add, Takers take.
 * In first version, nothing is synchronized, we'll change this.
 * to fix things -- (java ... 5 3 should cause problems)
 */

public class ThreadTrouble
{
    private LinkedList myList = new LinkedList();
    private Random     myRando = new Random();

    public ThreadTrouble(int putCount, int takeCount) {
        for(int k=0; k < putCount; k++){
            Putter put = new Putter(this);
            put.start();
        }
        for(int k=0; k < takeCount; k++){
            Taker take = new Taker(this);
            take.start();
        }
    }

    // Add an object to this collection

    public synchronized void put(Putter p, Object o) {
        System.out.println(p+" added "+o);
        myList.add(o);
    }

    /**
     * Take a random number of object out (if there are
     * any) and return the last one take. Returns null
     * if none to take. The loop can cause synchronization
     * issues, e.g., if two takers both try to remove 5 from
     * a list of 6 elements.
     */
    public synchronized Object take(Taker t) {
        if (myList.size() > 0){
            try{
                int index = myRando.nextInt(myList.size());
                System.out.println(t + " tries to take "+index);
                for(int k=0; k < index-1; k++){
                    myList.removeLast();
                }
                return myList.removeLast();
            }
            catch (Exception e){
                t.interrupt();
            }

        }
        return null;
    }

    public static void main(String[] args) {
        int putCount = 1;
        int takeCount = 1;
        if (args.length >= 1) {
            putCount = Integer.parseInt(args[0]);
        }
        if (args.length == 2) {
            takeCount = Integer.parseInt(args[1]);
        }

        ThreadTrouble ttrouble = new ThreadTrouble(putCount,takeCount);
    }
}
```



```
/**
 * A threaded putter that adds elements, sleeps,
 * and then adds some more.
 */
class Putter extends Thread
{
    ThreadTrouble mySource;

    public Putter(ThreadTrouble t) {
        mySource = t;
        myPutCount = ourTotalNumberOfPutters++;
    }

    public void run() {
        while (true){
            mySource.put(this,new Integer(ourCount++));
            try{
                Thread.sleep(500);
            }
            catch (InterruptedException e) {
                System.out.println("putter stopped");
                break;
            }
        }
    }

    public String toString()
    {
        return ""+myPutCount;
    }
    private int myPutCount;
    private static int ourCount = 0;
    private static int ourTotalNumberOfPutters = 0;
}

/**
 * A threaded taker that tries to take elements, sleeps,
 * then tries to take some more.
 */

class Taker extends Thread
{
    private ThreadTrouble mySource;
    private int myCount;
    private static int ourCount;

    public Taker(ThreadTrouble t) {
        mySource = t;
        myCount = ourCount++;
    }

    public void run() {
        while (true){
            Object o = mySource.take(this);
            if (o != null){
                System.out.println("\t took "+o);
            }

            try{
                Thread.sleep(500);
            }
            catch (InterruptedException e) {
                System.out.println("\t\t\ttaker " + myCount+ " stopped");
                ourCount--;
                if (ourCount == 1){
                    System.out.println("only one taker left, stopping");
                    System.exit(0);
                }
```

```
                break;
            }
        }
    }

    public String toString() {
        return ""+myCount;
    }
}
```

```
import java.util.*;

public class Timer implements Runnable
{
    public Timer()
    {
        myListeners = new ArrayList();
        myRunning = false;
        myThread = null;
        myTickPause = 1000;
    }

    public Timer(int duration, TimerListener tl)
    {
        this();
        myDuration = duration*1000;
        addListener(tl);
    }

    public void setPause(int pause)
    {
        myTickPause = pause;
    }

    public void addListener(TimerListener tl)
    {
        myListeners.add(tl);
    }

    public void removeListener(TimerListener tl)
    {
        myListeners.remove(tl);
    }

    public void start()
    {
        myThread = new Thread(this);
        myStartTime = new Date();
        myThread.setDaemon(true);
        myThread.start();
    }

    public void stop()
    {
        myRunning = false;
        myStopTime = new Date();
        for(int k=0; k < myListeners.size(); k++) {
            ( (TimerListener) myListeners.get(k)).timerStopped(this);
        }
    }

    public void run()
    {
        myRunning = true;

        while (myRunning) {

            try {
                pause();
            }
            catch (InterruptedException e) {
                myRunning = false;
                return;
            }

            handleTicks();

            if (isExpired()) {
                stop();
```



```
            }
        }
    }

    public boolean isExpired()
    {
        myStopTime = new Date();
        return timeRemaining() <= 0;
    }

    public long timeRemaining()
    {
        myStopTime = new Date();
        long secs = myStopTime.getTime() - myStartTime.getTime();
        return myDuration - secs;
    }

    public void pause() throws InterruptedException
    {
//      System.out.println("timer pause");
        myThread.sleep(myTickPause);
    }

    public void handleTicks()
    {
//      System.out.println("timer tick "+timeRemaining());
        for(int k=0; k < myListeners.size(); k++) {
            ( (TimerListener) myListeners.get(k)).timerTicked(this);
        }
    }

    private boolean myRunning;
    private long    myDuration;
    private Date    myStartTime;
    private Date    myStopTime;
    private Thread  myThread;
    private int     myTickPause;

    private List myListeners;
}
```

```
public interface TimerListener
{
    public void timerStarted(Timer timer);
    public void timerStopped(Timer timer);
    public void timerTicked(Timer timer);
}
```

```
import java.util.Date;
import java.io.*;

public class TimerUser implements TimerListener
{
    public TimerUser(int time)
    {
        myTimer = new Timer(time,this);
        myTimer.start();
        myRunning = true;
        stuff();
    }

    public void stuff()
    {
        BufferedReader reader = new BufferedReader(new
            InputStreamReader(System.in));

        String s=null;
        int count = 0;

        while (true) {
            try {
                if (reader.ready()) {
                    s = reader.readLine();
                    count++;
                    log(count + "\t" + s);
                }
            }
            catch (IOException ioe){ }

            if (! myRunning) {
                break;
            }
            try{
                Thread.sleep(50);
            }
            catch (InterruptedException ioe) { }
        }
    }

    public void timerStarted(Timer timer)
    {
        log("started "+timer.timeRemaining());
        myRunning = true;
    }

    public void timerStopped(Timer timer)
    {
        myRunning = false;
        log("stopped "+timer.timeRemaining());
    }

    public void timerTicked(Timer timer)
    {
        log ("ticking "+timer.timeRemaining());
    }

    protected void log(String s)
    {
        System.out.println(s);
    }

    Timer myTimer;
    private boolean myRunning;

    public static void main(String args[])
    {
        int time = 10;
```



```
        if (args.length > 0) {
            time = Integer.parseInt(args[0]);
        }
        TimerUser tu = new TimerUser(time);
    }
}
```