NOTION OF A MONITOR

A monitor is a synchronization approach that allows threads to wait until a condition is satisfied while enforcing mutual exclusion. A classic example of this would be the producer-consumer problem where the consumer not only has mutual exclusive access to a list of items, but also needs to wait if there are insufficient items on the list. Monitors in Java are provided using the “synchronized” keyword along with the “wait” and “notify”, “notifyAll” methods for the object.

The example here defines an empty list (with just an integer) and create 4 producers and 4 consumers that produce/consume 1-4 items at a time. If there are insufficient items for a consumer, it waits until they are available.

CODE: https://repl.it/@wolfram77/monitor-example#Main.java

```
import java.util.concurrent.*;


class Main {
static int[] items;
static int N = 4;


static void produce(int n) throws InterruptedException {
String id = "produce"+n;
synchronized (items) {
log(id+": items="+items[0]);
items[0] += n;
items.notifyAll();
log(id+": new items="+items[0]);
}
}


static void consume(int n) throws InterruptedException {
String id = "consume"+n;
synchronized (items) {
log(id+": items="+items[0]);
while(items[0]-n < 0) {
log(id+": too few items="+items[0]);
items.wait();
}
items[0] -= n;
log(id+": new items="+items[0]);
```

```
}
}


static void producer(int n) {
new Thread(() -> {
try {
while(true) {
produce(n);
Thread.sleep(random());
}
}
catch(InterruptedException e) {}
}).start();
}


static void consumer(int n) {
new Thread(() -> {
try {
while(true) {
consume(n);
Thread.sleep(random());
}
}
catch(InterruptedException e) {}
}).start();
}


public static void main(String[] args) {
items = new int[] {0};
for(int i=1; i<=N; i++) {
producer(i);
consumer(i);
}
}
static long random() {
return (long) (Math.random()*1000);
}
static void log(String x) {
System.out.println(x);
}
}
```

OUTPUT: https://monitor-example.wolfram77.repl.run

```
produce2: items=0
produce2: new items=2
consume4: items=2
consume4: too few items=2
consume1: items=2
consume1: new items=1
produce4: items=1
produce4: new items=5
consume2: items=5
consume2: new items=3
produce3: items=3
produce3: new items=6
consume3: items=6
consume3: new items=3
produce1: items=3
produce1: new items=4
consume4: new items=0
consume2: items=0
consume2: too few items=0
consume1: items=0
consume1: too few items=0
produce1: items=0
produce1: new items=1
consume2: too few items=1
consume1: new items=0
produce4: items=0
produce4: new items=4
consume2: new items=2
produce2: items=2
produce2: new items=4
consume2: items=4
consume2: new items=2
produce2: items=2
produce2: new items=4
produce3: items=4
produce3: new items=7
consume4: items=7
...
```