

The Apprentice Challenge

J Strother Moore
Department of Computer Sciences
University of Texas at Austin

A Class Involving Multiple Threads

```
class Container {  
    public int counter; }  
  
class Job extends Thread {  
    Container objref;  
  
    public void setref(Container o) {  
        objref = o; }  
}
```

```
public Job incr () {  
    synchronized(objref) {  
        objref.counter = objref.counter + 1; }  
    return this; }  
  
public void run() {  
    for (;;) {incr(); } } }
```

```
class Apprentice {  
    public static void main(String[] args){  
        Container container = new Container();  
        for (;;) {Job job = new Job();  
            job.setref(container);  
            job.start(); } } }
```

Theorem The value of the counter never decreases.

This has to be formulated more carefully to account for Java's 32-bit `int` arithmetic.

Theorem

```
(let* ((s1 (run sched *a0*))
      (s2 (step th s1)))
  (or (equal (counter s1) nil)
      (equal (counter s1)
              (counter s2))
      (equal (int-fix (+ 1 (counter s1)))
              (counter s2))))
```

It's Obvious...

```
class Container {public int counter; }
class Job extends Thread {
    Container objref;
    public void setref(Container o) {objref = o; }
    public Job incr () {
        synchronized(objref){
            objref.counter = objref.counter + 1;}
        return this; }
    public void run() {
        for (;;) {incr(); } } }
class Apprentice {
    public static void main(String[] args){
        Container container = new Container();
        for (;;) {Job job = new Job();
            job.setref(container);
            job.start(); } } }
```

```

class Container {public int counter; }
class Job extends Thread {
    Container objref;
    public void setref(Container o) {objref = o; }
    public Job incr () {
        synchronized(objref) {objref.counter = objref.counter + 1; }
        return this; }
    public void run() {
        for (;;) {incr(); } } }
class Apprentice {
    public static void main(String[] args){
        Container container = new Container();
        Container bogus      = new Container();
        for (;;) {Job job = new Job();
            job.setref(bogus);
            job.start();
            job.setref(container); } } }

```


Thread Table

Thread 0

Heap

Class Table

```
Job incr()  
synchronized(objref)  
{objref.counter = objref.counter + 1;}
```

```
Apprentice main()  
  
Container container = new Container();  
Container bogus    = new Container()  
for (;;) { Job job = new Job();  
            job.setref(bogus);  
            job.start();  
            job.setref(container);}
```

Thread Table

Thread 0

Class Table

```
Job incr()  
synchronized(objref)  
{objref.counter = objref.counter + 1;}
```

Heap

container

counter:
0

monitor:

container

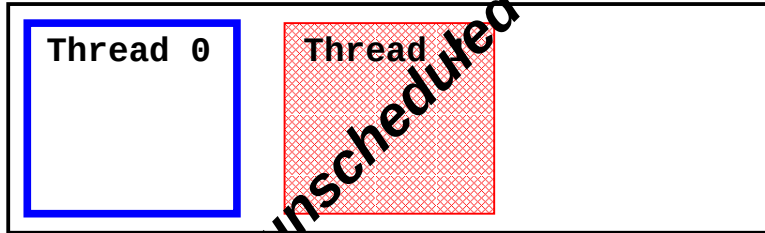
counter:
0

monitor:

Apprentice main()

```
Container container = new Container();  
Container bogus    = new Container()  
for (;;) { Job job = new Job();  
             job.setref(bogus);  
             job.start();  
             job.setref(container);}
```

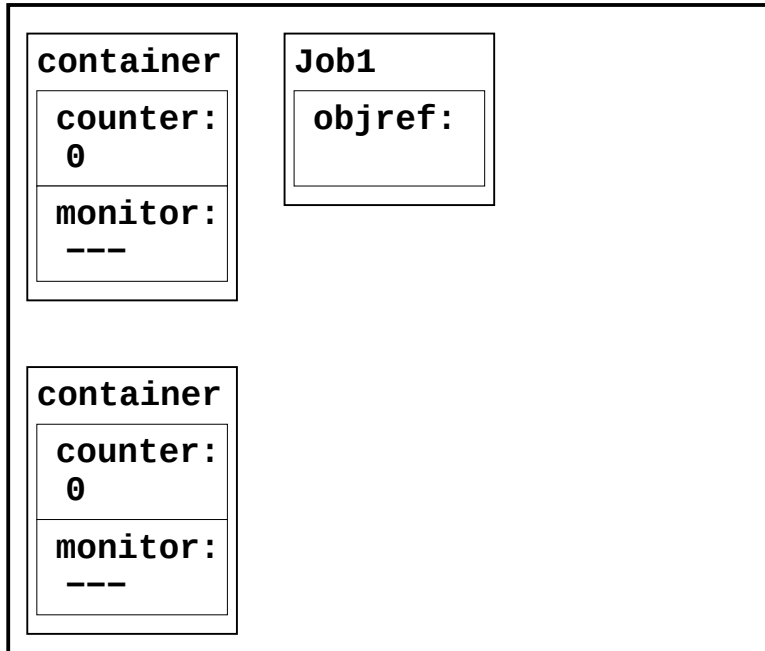
Thread Table



Class Table

```
Job incr()  
synchronized(objref)  
{objref.counter = objref.counter + 1;}
```

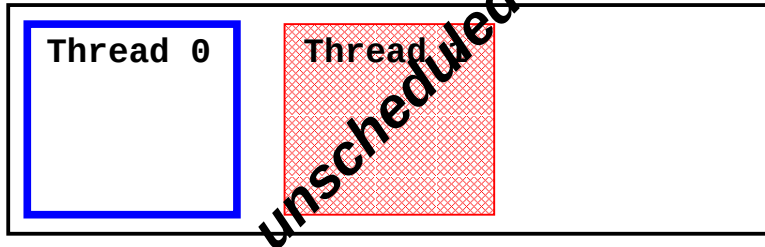
Heap



```
Apprentice main()
```

```
Container container = new Container();  
Container bogus     = new Container()  
for (;;) { Job job = new Job();  
           job.setref(bogus);  
           job.start();  
           job.setref(container);}
```

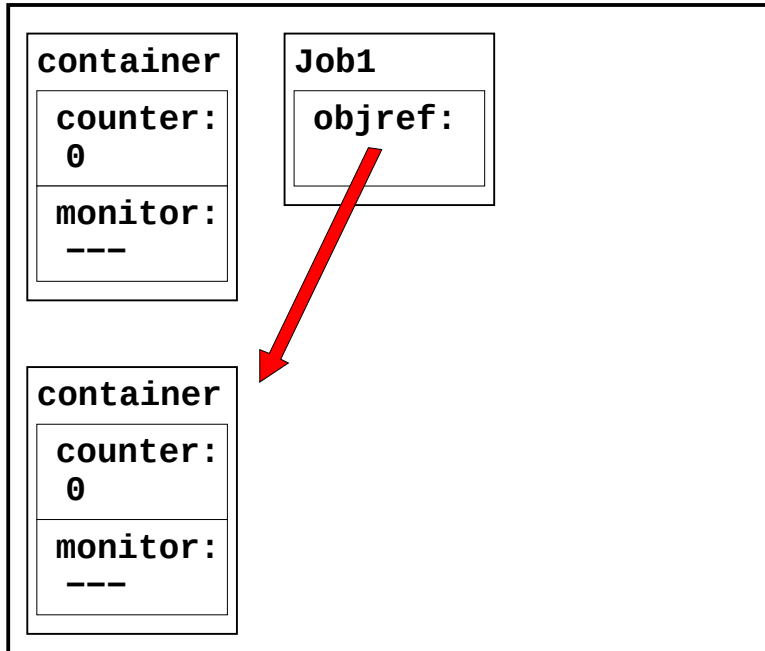
Thread Table



Class Table

```
Job incr()  
synchronized(objref)  
{objref.counter = objref.counter + 1;}
```

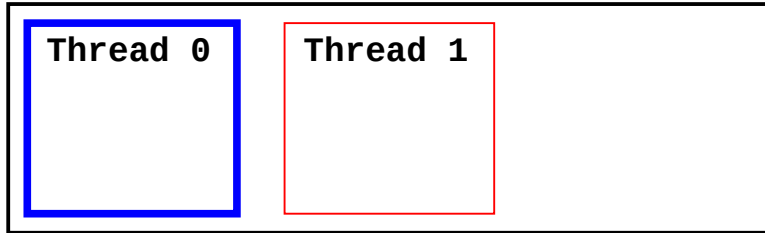
Heap



Apprentice main()

```
Container container = new Container();  
Container bogus     = new Container()  
for (;;) { Job job = new Job();  
            job.setref(bogus);  
            job.start();  
            job.setref(container);}
```

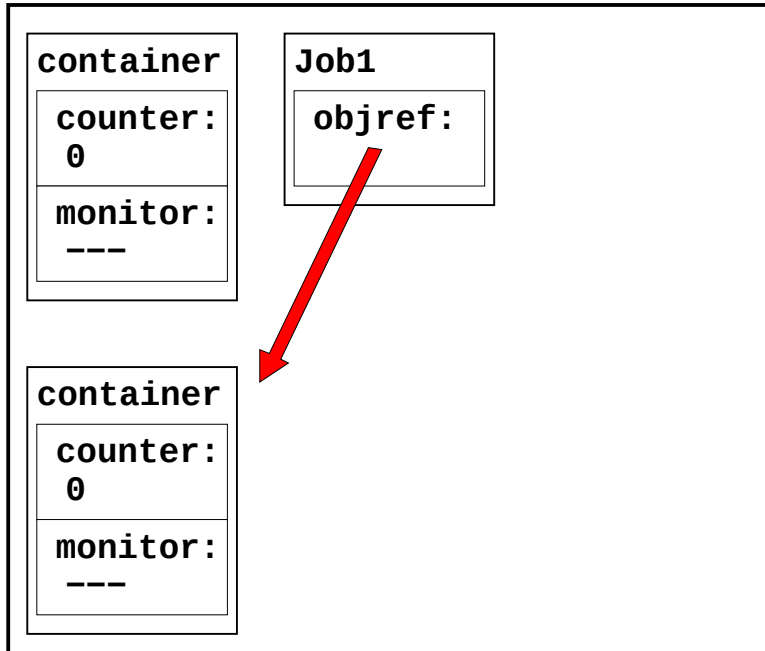
Thread Table



Class Table

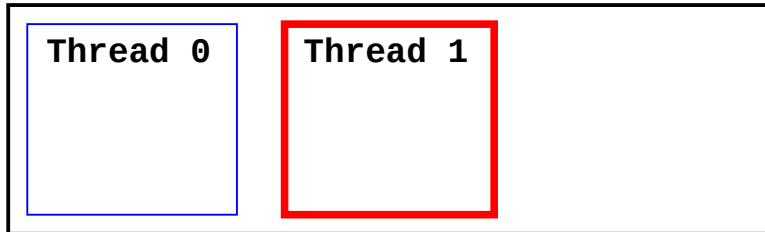
```
Job incr()  
synchronized(objref)  
{objref.counter = objref.counter + 1;}
```

Heap



```
Apprentice main()  
  
Container container = new Container();  
Container bogus    = new Container()  
for (;;) { Job job = new Job();  
            job.setref(bogus);  
            job.start();  
            job.setref(container);}
```

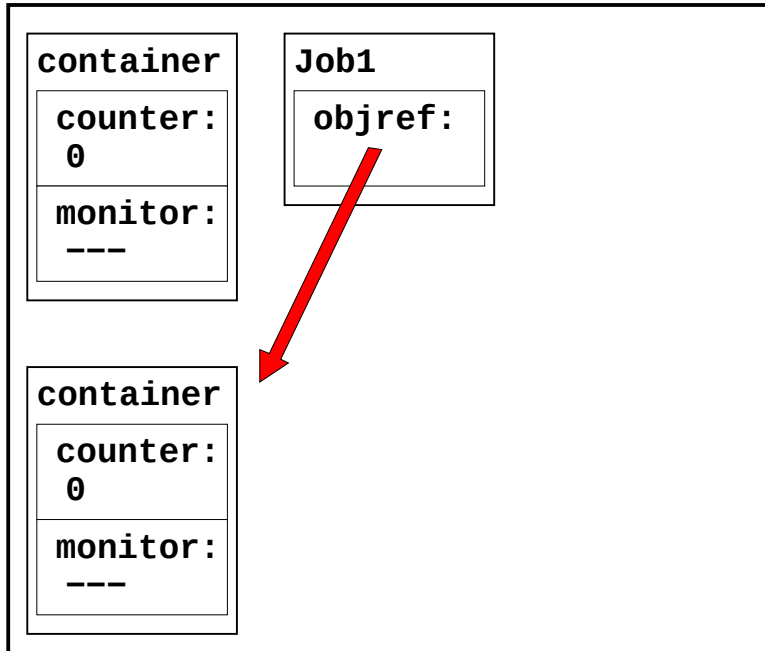
Thread Table



Class Table

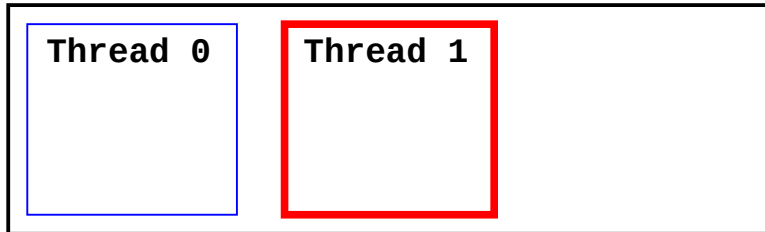
```
Job incr()  
synchronized(objref)  
    {objref.counter = objref.counter + 1;}
```

Heap



```
Apprentice main()  
  
Container container = new Container();  
Container bogus    = new Container()  
for (;;) { Job job = new Job();  
            job.setref(bogus);  
            job.start();  
            job.setref(container);}
```

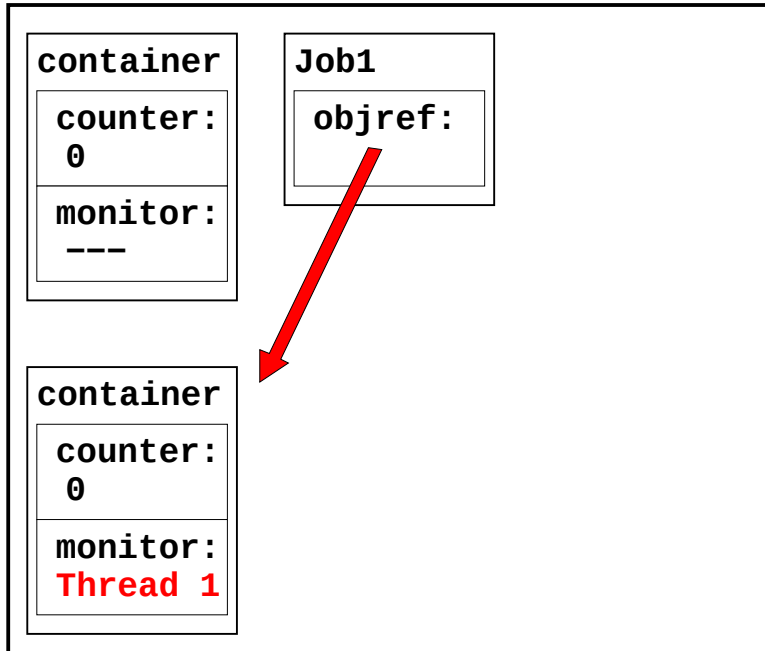
Thread Table



Class Table

```
Job incr()  
synchronized(objref)  
{objref.counter = objref.counter + 1;}
```

Heap



```
Apprentice main()  
  
Container container = new Container();  
Container bogus     = new Container();  
for (;;) { Job job = new Job();  
            job.setref(bogus);  
            job.start();  
            job.setref(container);}
```

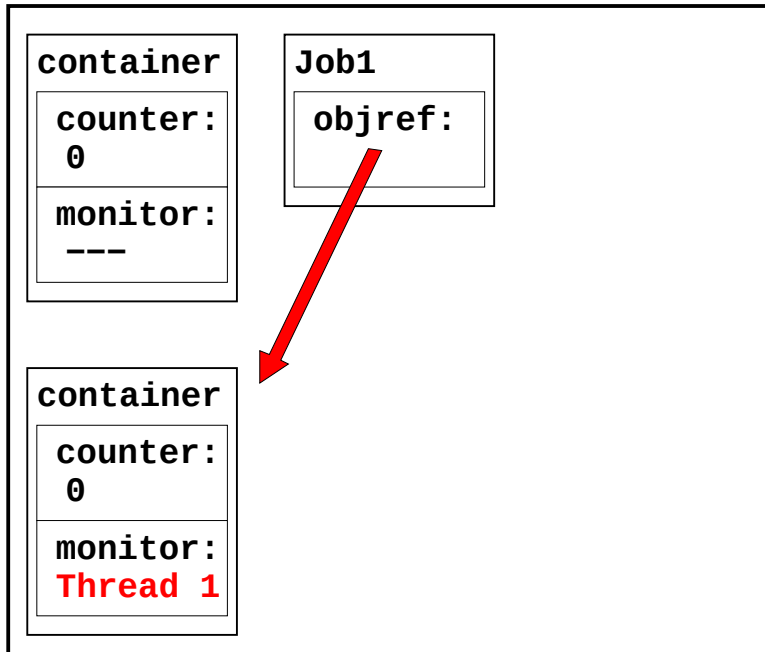
Thread Table

Thread 0	Thread 1
	1

Class Table

```
Job incr()  
synchronized(objref)  
{objref.counter = objref.counter + 1;}
```

Heap



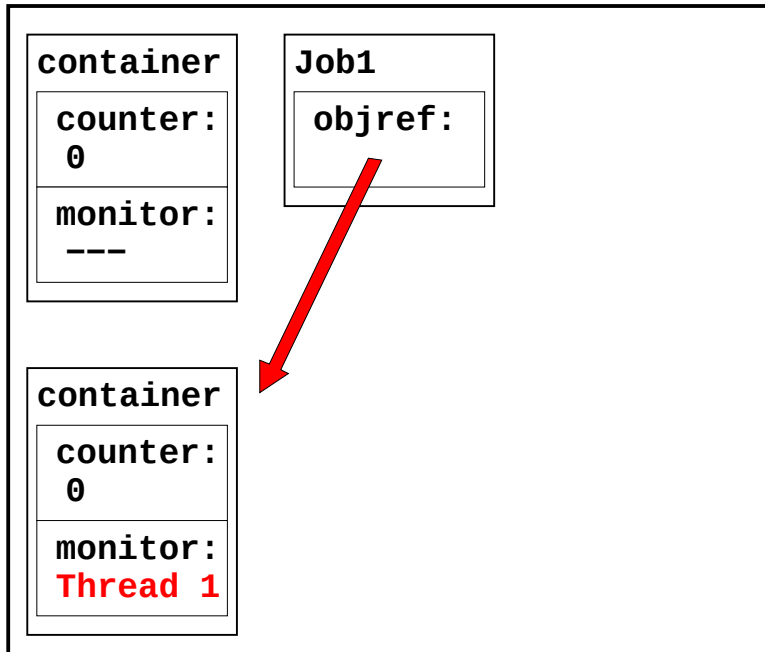
Apprentice main()

```
Container container = new Container();  
Container bogus    = new Container();  
for (;;) { Job job = new Job();  
            job.setref(bogus);  
            job.start();  
            job.setref(container);}
```


Thread Table

Thread 0	Thread 1
	1

Heap

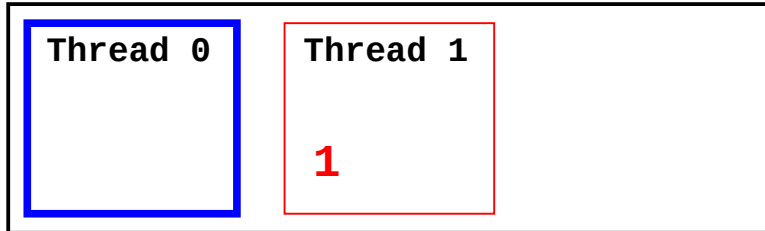


Class Table

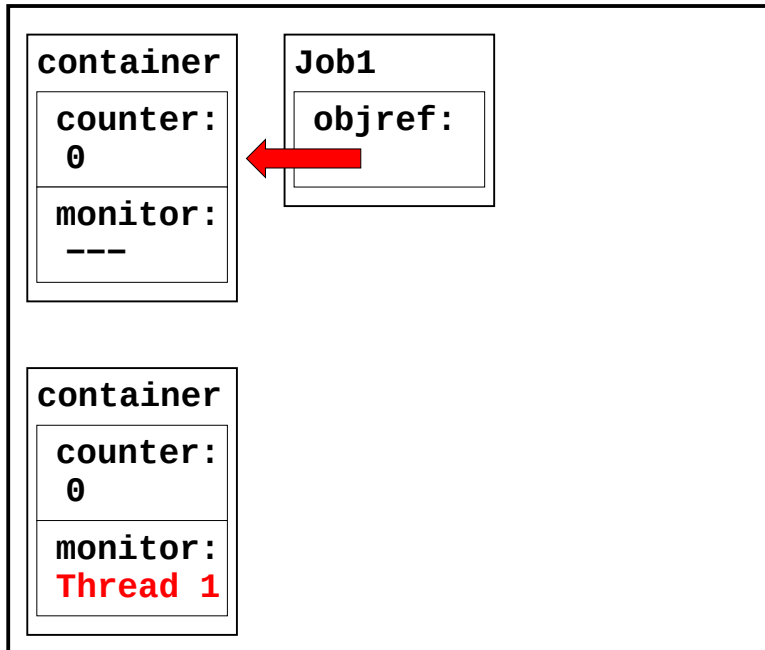
```
Job incr()  
synchronized(objref)  
{objref.counter = objref.counter + 1;}
```

```
Apprentice main()  
  
Container container = new Container();  
Container bogus     = new Container()  
for (;;) { Job job = new Job();  
            job.setref(bogus);  
            job.start();  
            job.setref(container);}
```

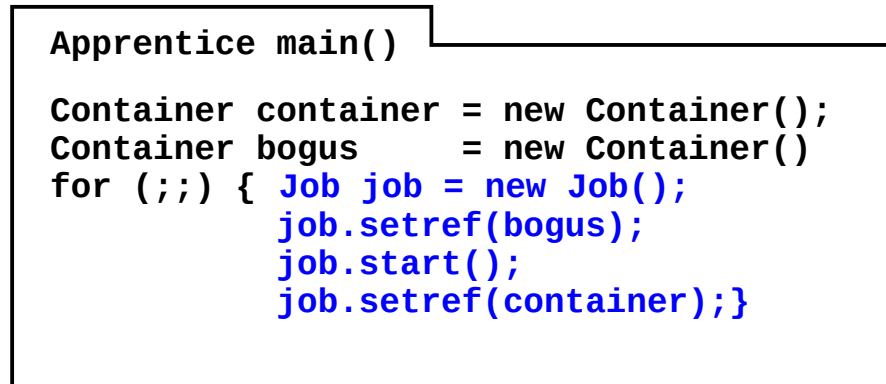
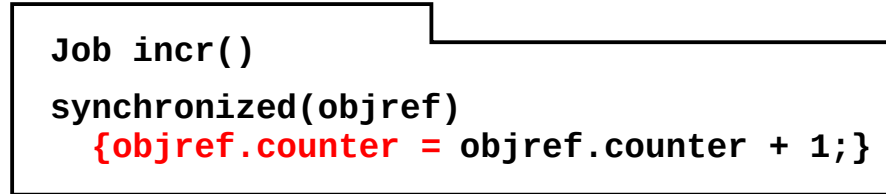
Thread Table



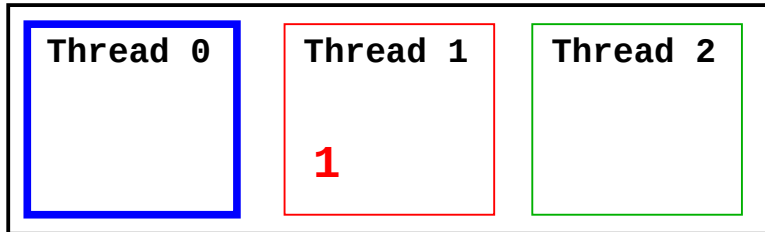
Heap



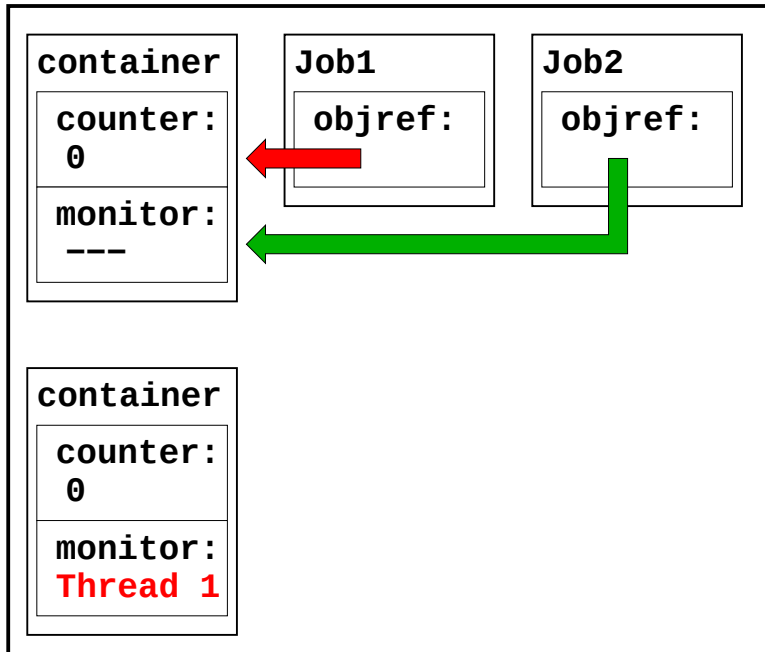
Class Table



Thread Table



Heap

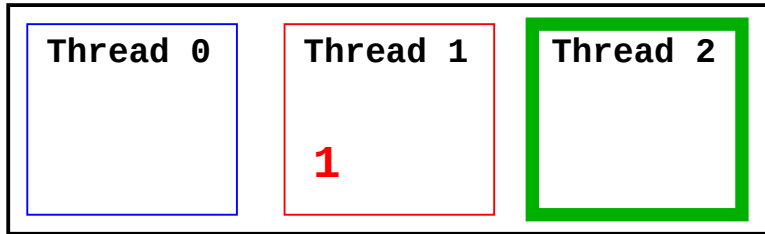


Class Table

```
Job incr()  
synchronized(objref)  
{objref.counter = objref.counter + 1;}
```

```
Apprentice main()  
  
Container container = new Container();  
Container bogus     = new Container();  
for (;;) { Job job = new Job();  
            job.setref(bogus);  
            job.start();  
            job.setref(container);}
```

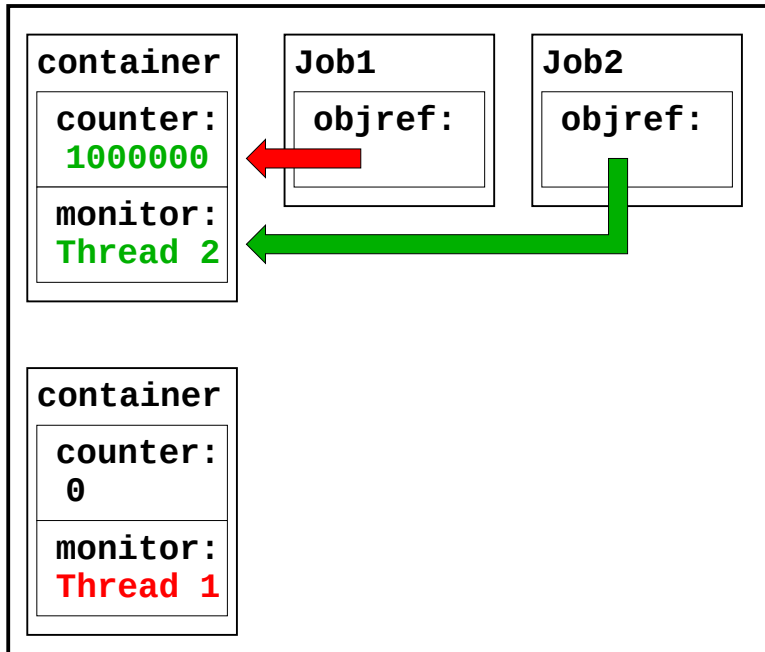
Thread Table



Class Table

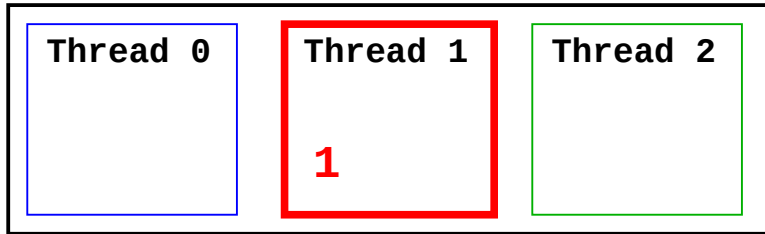
```
Job incr()  
synchronized(objref)  
{objref.counter = objref.counter + 1;}
```

Heap

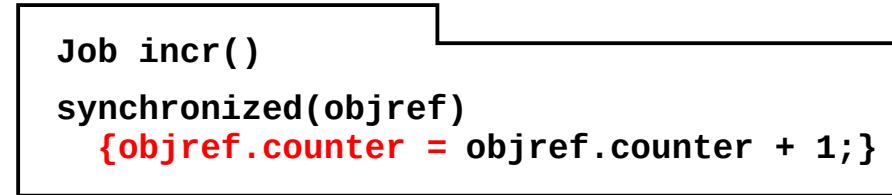


```
Job incr()  
synchronized(objref)  
{objref.counter = objref.counter + 1;}
```

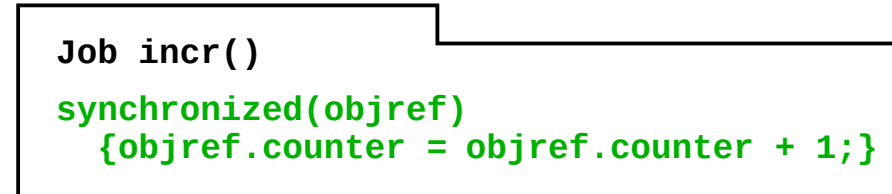
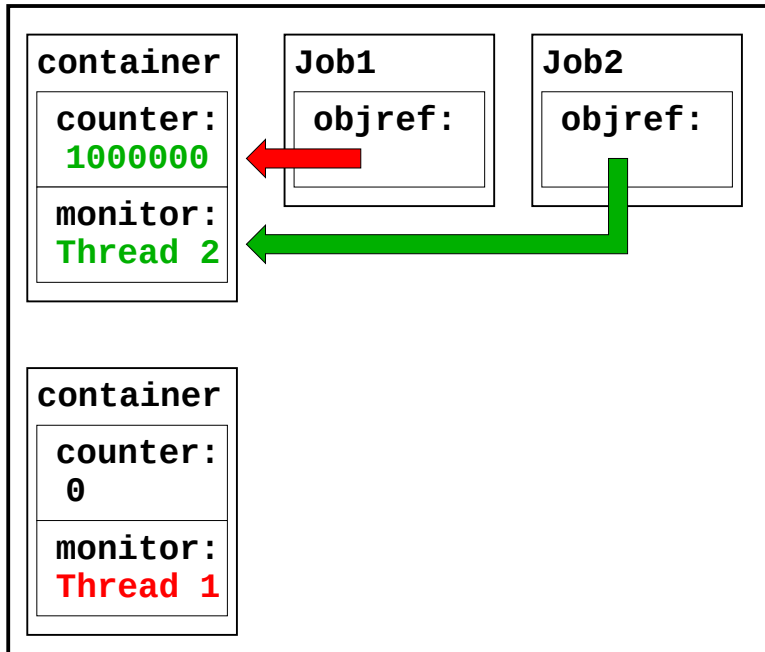
Thread Table



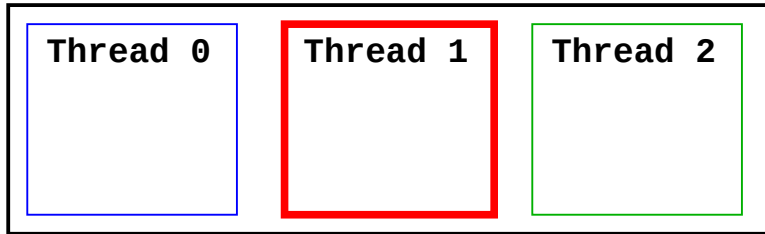
Class Table



Heap



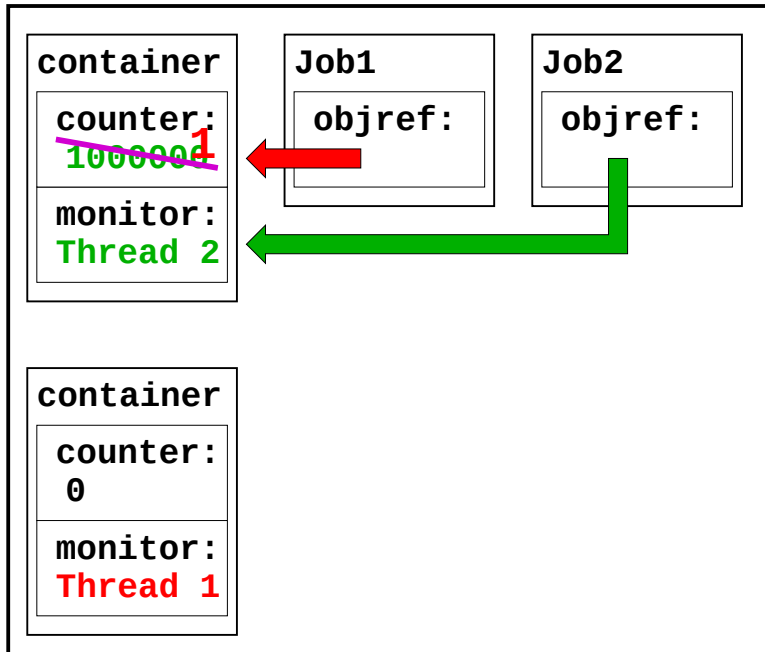
Thread Table



Class Table

```
Job incr()  
synchronized(objref)  
{objref.counter = objref.counter + 1;}
```

Heap



```
Job incr()  
synchronized(objref)  
{objref.counter = objref.counter + 1;}
```

```

class Container {
    public int counter; }
class Job extends Thread {
    Container objref;
    public void setref(Container o) {objref = o; }
    public Job incr () {
        synchronized(objref) {objref.counter = objref.counter + 1; }
        return this; }
    public void run() {
        for (;;) {incr(); } } }
class Apprentice {
    public static void main(String[] args){
        Container container = new Container();
        for (;;) {Job job = new Job();
            job.setref(container);
            job.start(); } } }

```

Theorem

```
(let* ((s1 (run sched *a0*))  
      (s2 (step th s1)))  
  (rel (counter s1) (counter s2)))
```



```
(defthm Lemma-1
  (good-state *a0*))
```

```
(defthm Lemma-2
  (implies (good-state s)
    (good-state (step th s))))
```

```
(defthm Lemma-3
  (implies (good-state s)
    (rel (counter s)
      (counter (step th s)))))
```