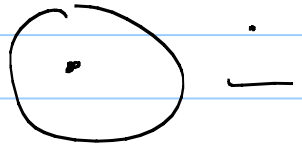
Lecture 15

Note Title 3/4/2008

Bound – v.



past bounded

opposite unbounded

bind – v

past tense bound

opposite unbound

```
public class DFVariable<E> {
   private E var; private boolean bound;
   public DFVariable() {
        var=null ; bound=false; }
   public synchronized E get () {
         while (! bound) wait();
         return var
    }
   public synchronized set (E var) {
        if (bound && var != this.var) raise ValueConflict;
        else {this.var = var; NotifyAll();}
    } }
```

RMI pattern of § 5.3.1

Bounded buffer

put while(full) {wait()}
     put new value NotifyAll

get while(empty) { wait() }
     return a value; NotifyAll();

recv
put [not full] →

Use of Notify() vs NotifyAll()

3. C-A-S vs T-A-S

CAS indivisibly {
r1, r2, a1
tmp ← c(a1)
if c(tmp) == c(r1) then a1 ← c(r2)
r2 ← c(tmp)
}

TAS r1, a1:

```
LI Rtmp, 0
LI r1, 1
CAS Rtmp, r1, a1
```

TAS r1, a1

r1 ← c(a1)

a1 ← 1

} c(a1) ∈ {0, 1} indivisibly

implement TAS using CAS

TAS behavior

old

c(a1)	0	1
r1	0	1
a1	1	1

```
synchronized (p) { this.set(p.x, p.y); }

int [] p2 = p.get();          x,y = p.get()
    this.set (p2[0], p2[1]);
```


5

Doug Lea

Java 5 locking library

Lock acquisition using synchronized is

1) not interruptable
2) not timeout able
3) only blockstructured - locks must be released in reverse order of acquisition

Furthermore: only one wait queue per object

Lock interface:

```
void lock();
void lockInterruptibly() throws InterruptedException;
bool tryLock();
bool tryLock(long timeout, TimeUnit unit) throws IE;
void unlock();
Condition newCondition(); // wait
```

ReentrantLock implements Lock.