Ask me anything

Mentimeter

2 questions
0 upvotes

How does a Nested Monitor fulfil the 4 necessary and sufficient conditions for deadlock to occur?

Mentimeter

(i) serially reusable resources is manifested in nested monitors

i) lock can be obtained again and again

(ii) host lock and guest lock

iii) no pre-emption is given, as long as we are in Java generally, because the language does not support pre-emption

iiii) Blocking waiting for the lock

i) java locks are reentrantii) to get the nested lock, we first need to have the mutexiii) no other thread can grab the lock we're holdingiv) the inner lock might wait for a release, which depends on releasing outer lock

(iv) waiting for notification but blocking for other to notify you

7

Why does BCCondition sync on notMin in the guard?

Mentimeter

Example solution

```
public class BoundedCounterCondition extends BoundedCounterAbstract {
public void dec() {                        // not synched!
      boolean wasMax = false;              // record notification condition
      synchronized(notMin) {               // synch on condition object
         while (true) {                    // new guard loop
            synchronized(this) {
               if (count > MIN) {          // check and act
                  wasMax = (count == MAX);
                  count--;
                  break;
               }
            }
            notMin.await();                // release host synch before wait
         }
      }
      if (wasMax) notMax.signal();         // release all syncs!
   } ...
}
```

Why must we sync on notMin?

16

so that we do not miss notMin.signal()s

because else we could end with being stuck in notMin.await()

We want to sync on the same object (guest) to avoid any nested monitor situation

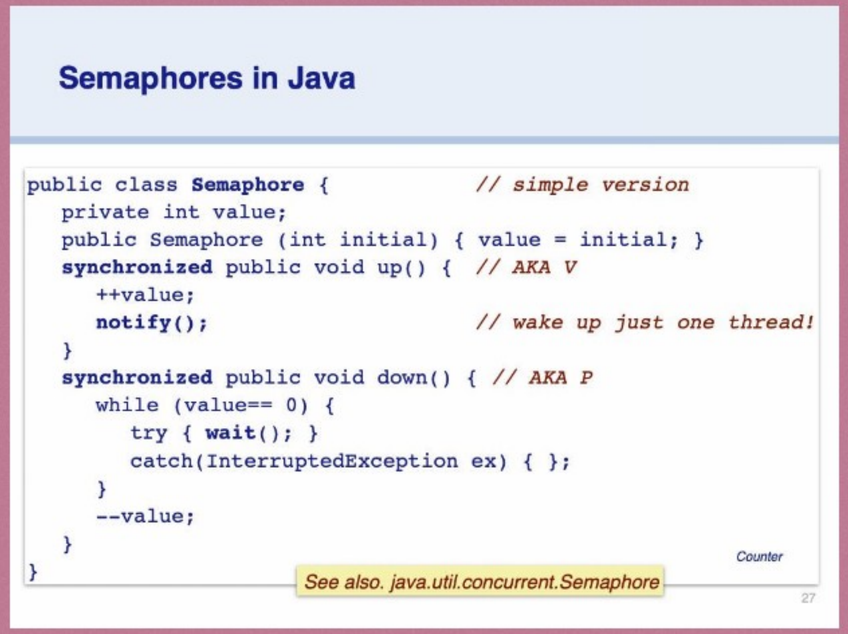
Because the condition in down is the same for all threads

Otherwise we could get blocked. We have to sync with respect to the conditional object

5

Why is it safe to perform notify() instead of notifyAll() in our Semaphore implementation?

Mentimeter



Because we do not care which one of the threads are waken up (basically everyone waits for the same condition becomes true)

Because every ++value will always allow exactly one thread to continue (and this section is synchronised).

because only one thread will be allowed to do a down()

Because any one thread waiting in down is ok to woke up

Because the condition in down() is the same for all threads

We know which thread to wake up

if there's threads sleeping at most one

7

How does a semaphore differ from a simple condition object?

We can choose to let n threads in with semaphore

semaphore doesn't have to be binary

Semaphore can be increased to any number and never falls below zero. However, conditions can be true or false (binary)

it owns a count variable which allows several threads to do sth before access is locked. With a simple condition object, we only provide wait and signal for a single lock

semaphores only lock on down if the counter reaches 0.

A condition object can implement a complex condition (boolean expression).

Mentimeter

Last chance for questions