Questions

Bookmark this page

Synchronizing with locks

2/2 points (graded)

If thread B tries to acquire a lock currently held by thread A:

What happens to thread A?

- ○ blocks until B acquires the lock
- ○ blocks until B releases the lock
- ● nothing

✓

What happens to thread B?

- ○ blocks until A acquires the lock
- ● blocks until A releases the lock
- ○ nothing

✓

Explanation
Thread B blocks until A releases the lock. Then B acquires the lock.

Submit

Show Answer

Answers are displayed within the problem

This list is mine, all mine

0/1 point (graded)

Suppose `list` is an instance of `ArrayList<String>`.

What is true while A is in a `synchronized (list) { ... }` block?

- [✓] it owns the lock on `list`
- [ ] it does not own the lock on `list`
- [ ] no other thread can use observers of `list`
- [ ] no other thread can use mutators of `list`
- [✓] no other thread can acquire the lock on `list`
- [ ] no other thread can acquire locks on elements in `list`

Explanation
Acquiring a lock means one thing and one thing only: no other thread can acquire that lock, until the lock is released.
The spec of `ArrayList` could advertise that acquiring the lock on an instance of `ArrayList` provides some synchronization benefit, but it does not.
Similarly, acquiring the lock on a list doesn't automatically acquire the locks of its elements.
If you wanted acquiring the lock on `list` to mean some of the other things above, all the threads would have to agree on that locking discipline.

Submit

Show Answer

OK fine but this synchronized List is totally mine

1/1 point (graded)

Suppose `sharedList` is a `List` returned by `Collections.synchronizedList`.

It is now safe to use `sharedList` from multiple threads without acquiring any locks... except! Which of the following would require a `synchronized(sharedList) { ... }` block?

- [ ] call `isEmpty`
- [ ] call `add`
- [✓] iterate over the list

[✓] call `isEmpty`, if it returns false, call `remove(0)`

✓

Explanation

The `sharedList` is safe for concurrency, so individual operations are safe to call without additional synchronization.
The spec of `Collections.synchronizedList` explicitly says that you must synchronize on the list before iterating. This prevents other clients from mutating the list during the iteration.
We must also synchronize on the list any time we need to maintain an invariant between calls to individual operations: in between our call to `isEmpty` and `remove`, someone else could have emptied the list!

Submit

Show Answer

Answers are displayed within the problem

Previous Next