

Worksheet #26a: use of tryLock()

Name: _____

Netid: _____

Rewrite the `transferFunds()` method below to use j.u.c. locks with calls to `tryLock` (see slide 4) instead of `synchronized`. Your goal is to write a correct implementation that never deadlocks, unlike the buggy version below (which can deadlock). Assume that each `Account` object already contains a reference to a `ReentrantLock` object dedicated to that object e.g., `from.lock()` returns the lock for the `from` object. Sketch your answer below using pseudocode.

```
public void transferFunds(Account from, Account to, int amount) {  
1.     synchronized (from) {  
2.         synchronized (to) {  
3.             from.subtractFromBalance(amount);  
4.             to.addToBalance(amount);  
5.         }  
6.     }  
7. }
```



Worksheet #26b:

Linearizability of method calls on a concurrent object

Name: _____

Netid: _____

Is this a linearizable execution for a FIFO queue, q ? If so, why? If not, why not?

Time	Task A	Task B
0	Invoke $q.enq(x)$	
1	Return from $q.enq(x)$	
2		Invoke $q.enq(y)$
3	Invoke $q.deq()$	Work on $q.enq(y)$
4	Work on $q.deq()$	Return from $q.enq(y)$
5	Return y from $q.deq()$	

