Rewrite the transferFunds() method below to use j.u.c. locks with calls to tryLock 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 using pseudocode.

```
public void transferFunds (Account from, Account to, int amount) {
1.
2.
         synchronized (from) {
3.
             synchronized (to) {
4.
5.
                      to.addToBalance(amount);
6.
7.
8.
```

from.subtractFromBalance(amount);

COMP 322, Spring 2022 (M. Joyner, Z. Budimlić)





Rewrite the transferFunds() method below to use j.u.c. locks with calls to tryLock instead of synchronized.

from.lock() returns the lock for the from object. Sketch your answer using pseudocode.

1.	public void transferFunds (Account from, Ac
2.	while (true) {
3.	<pre>// assume that trylock() does not</pre>
4.	<pre>boolean fromFlag = from.lock.tryloc</pre>
5.	<pre>if (!fromFlag) continue;</pre>
6.	<pre>boolean toFlag = to.lock.trylock();</pre>
7.	<pre>if (!toFlag) { from.lock.unlock();</pre>
8.	<pre>try { from.subtractFromBalance(amou</pre>
9.	<pre>to.addToBalance(amount); brea</pre>
10.	<pre>finally { from.lock.unlock(); to.lock</pre>
11.	} // while
12.	}

Worksheet solution: Use of trylock()

- 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.,

```
ccount to, int amount) {
throw an exception
ck();
continue; }
unt);
ak; }
k.unlock(); }
```



