Worksheet: Use of trylock()

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.

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

Worksheet solution: Use of trylock()

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.

```
1.  public void transferFunds(Account from, Account to, int amount) {
2.      while (true) {
3.          // assume that trylock() does not throw an exception
4.          boolean fromFlag = from.lock.trylock();
5.          if (!fromFlag) continue;
6.          boolean toFlag = to.lock.trylock();
7.          if (!toFlag) { from.lock.unlock(); continue; }
8.          try { from.subtractFromBalance(amount);
9.                to.addToBalance(amount); break; }
10.         finally { from.lock.unlock(); to.lock.unlock(); }
11.     } // while
12.  }
```