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.

```
public void transferFunds(Account from, Account to, int amount) {
        synchronized (from) {
            synchronized (to) {
                    from.subtractFromBalance(amount);
                    to.addToBalance(amount);
            }
        }
}
```

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.

```
public void transferFunds(Account from, Account to, int amount) {
    while (true) {
      // assume that trylock() does not throw an exception
      boolean fromFlag = from.lock.trylock();
      if (!fromFlag) continue;
      boolean toFlag = to.lock.trylock();
      if (!toFlag) { from.lock.unlock(); continue; }
      try { from.subtractFromBalance(amount);
              to.addToBalance(amount); break; }
      finally { from.lock.unlock(); to.lock.unlock(); }
    } // while
}
```