Worksheet: Dynamic Order with Synchronized Statements

Consider a method to transfer a balance from one account to another. Could this result in a deadlock? If not, please explain why not. If so, explain why it can and if there's a solution to prevent it.

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


Worksheet solution: Dynamic Order with Synchronized Statements

Consider a method to transfer a balance from one account to another. Could this result in a deadlock? If not, please explain why not. If so, explain why it can and if there's a solution to prevent it.

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

— What if one thread tries to transfer from A to B while another tries to transfer from B to A ?

Inconsistent lock order again – Deadlock!

Worksheet solution: Avoiding Dynamic Order Deadlocks

- The solution is to induce a lock ordering
  - Here, uses an existing unique numeric key, acctId, to establish an order

```
public class SafeTransfer {
 public void transferFunds(Account from, Account to, int amount) {
   Account firstLock, secondLock;
      if (fromAccount.acctId == toAccount.acctId)
         throw new Exception("Cannot self-transfer");
      else if (fromAccount.acctId < toAccount.acctId) {
         firstLock = fromAccount;
         secondLock = toAccount;
      }
      else {
         firstLock = toAccount;
         secondLock = fromAccount;
      }
      synchronized (firstLock) {
         synchronized (secondLock) {
            from.subtractFromBalance(amount);
            to.addToBalance(amount);
         }
      }
   }
}
```