Demo Deadlocks

Issue Description (Deadlock)

- Transfer() holds a lock on this.sync, while calling other.Deposit that effects a lock on other.sync.
- Transfer thus acquires nested locks on BankAccount this and therein on BankAccount other.
- Main() uses three threads calling Transfer:
  - First thread from account 1 to account2,
  - Second thread from account 2 to account 3,
  - Third thread from account 3 to account1.
- A cyclic wait dependency may occur:
  - First thread locks account1, awaiting account2
  - Second thread locks account2, awaiting account3
  - Third thread locks account3, awaiting account1
  - => A deadlock

```
public void Deposit(int amount)
{
   lock (sync)                    // NESTED LOCK
   {
      balance += amount;
   }
}
```

```
// CYCLIC LOCK ORDER

new Thread(() => account1.Transfer(account2, 100)).Start();
new Thread(() => account2.Transfer(account3, 100)).Start();
new Thread(() => account3.Transfer(account1, 100)).Start();
```

Checker Output (1 Issue, 3 Threads Involved)

```
Issue: #0 Deadlock with 3 threads
  caused by wait at "lock (sync) { balance += amount;..." in BankAccount.cs line 10
    caused by call Deposit at "other.Deposit(amount)" in BankAccount.cs line 21
      caused by call Transfer at "account2.Transfer(account3, 100)" in BankTest.cs line 13
        caused by thread or task at "() => account2.Transfer(account3..." in BankTest.cs line 13
          caused by call BankTest.BankTest.Main()
            caused by initial thread at "Main" in BankTest.cs line 7
  caused by wait at "lock (sync) { balance += amount;..." in BankAccount.cs line 10
    caused by thread or task at "() => account3.Transfer(account1..." in BankTest.cs line 14
      caused by call BankTest.BankTest.Main()
        caused by initial thread at "Main" in BankTest.cs line 7
  caused by wait at "lock (sync) { balance += amount;..." in BankAccount.cs line 10
    caused by thread or task at "() => account1.Transfer(account2..." in BankTest.cs line 12
      caused by call BankTest.BankTest.Main()
        caused by initial thread at "Main" in BankTest.cs line 7
```

Problem Fixing

Linear Lock Ordering

Break the cycle by introducing a linear order: Each thread may only acquire the lower account lock first, and then the higher lock. This can be achieved by inverting the transfer direction of one thread:

E.g. Instead of transferring from account 3 to account1, the third thread could transfer the negative amount in the opposite direction, from account1 to account3 (increasing object numbering).

```
new Thread(() => account1.Transfer(account2, 100)).Start();
new Thread(() => account2.Transfer(account3, 100)).Start();
new Thread(() => account3.Transfer(account1, -100)).Start();
```