**Part 96 - How to resolve a deadlock in a multithreaded program**

In this video we will discuss, **resolving a deadlock in a multithreaded program**. There are several techniques to avoid and resolve deadlocks. For example  
**1.** Acquiring locks in a specific defined order  
**2.** Mutex class  
**3.** Monitor.TryEnter() method

In this video, we will discuss, **acquiring locks in a specific defined order to resolve a deadlock**. We will be working with the same example that we worked with in [Part 95](http://csharp-video-tutorials.blogspot.com/2014/03/part-95-deadlock-in-multithreaded_19.html).   
  
**Sample program code used in the demo.**

usingSystem**;**

usingSystem.Threading**;**

publicclassProgram

**{**

publicstaticvoidMain**()**

**{**

Console.WriteLine**(**"Main Started"**);**

AccountaccountA=newAccount**(**101**,**5000**);**

AccountaccountB=newAccount**(**102**,**3000**);**

AccountManageraccountManagerA=

newAccountManager**(**accountA**,**accountB**,**1000**);**

ThreadT1=newThread**(**accountManagerA.Transfer**);**

T1.Name="T1"**;**

AccountManageraccountManagerB=

newAccountManager**(**accountB**,**accountA**,**2000**);**

ThreadT2=newThread**(**accountManagerB.Transfer**);**

T2.Name="T2"**;**

T1.Start**();**

T2.Start**();**

T1.Join**();**

T2.Join**();**

Console.WriteLine**(**"Main Completed"**);**

**}**

**}**

publicclassAccount

**{**

double\_balance**;**

int\_id**;**

publicAccount**(**intid**,**doublebalance**)**

**{**

this.\_id=id**;**

this.\_balance=balance**;**

**}**

publicintID

**{**

get

**{**

return\_id**;**

**}**

**}**

publicvoidWithdraw**(**doubleamount**)**

**{**

\_balance-=amount**;**

**}**

publicvoidDeposit**(**doubleamount**)**

**{**

\_balance+=amount**;**

**}**

**}**

publicclassAccountManager

**{**

Account\_fromAccount**;**

Account\_toAccount**;**

double\_amountToTransfer**;**

publicAccountManager**(**AccountfromAccount**,**AccounttoAccount**,**doubleamountToTransfer**)**

**{**

this.\_fromAccount=fromAccount**;**

this.\_toAccount=toAccount**;**

this.\_amountToTransfer=amountToTransfer**;**

**}**

publicvoidTransfer**()**

**{**

object\_lock1**,**\_lock2**;**

if**(**\_fromAccount.ID<\_toAccount.ID**)**

**{**

\_lock1=\_fromAccount**;**

\_lock2=\_toAccount**;**

**}**

else

**{**

\_lock1=\_toAccount**;**

\_lock2=\_fromAccount**;**

**}**

Console.WriteLine**(**Thread.CurrentThread.Name

+" trying to acquire lock on "

+**((**Account**)**\_lock1**)**.ID.ToString**());**

lock**(**\_lock1**)**

**{**

Console.WriteLine**(**Thread.CurrentThread.Name

+" acquired lock on "

+**((**Account**)**\_lock1**)**.ID.ToString**());**

Console.WriteLine**(**Thread.CurrentThread.Name

+" suspended for 1 second"**);**

Thread.Sleep**(**1000**);**

Console.WriteLine**(**Thread.CurrentThread.Name

+" back in action and trying to acquire lock on "

+**((**Account**)**\_lock2**)**.ID.ToString**());**

lock**(**\_lock2**)**

**{**

Console.WriteLine**(**Thread.CurrentThread.Name

+" acquired lock on "

+**((**Account**)**\_lock2**)**.ID.ToString**());**

\_fromAccount.Withdraw**(**\_amountToTransfer**);**

\_toAccount.Deposit**(**\_amountToTransfer**);**

Console.WriteLine**(**Thread.CurrentThread.Name+" Transfered "

+\_amountToTransfer.ToString**()**+" from "

+\_fromAccount.ID.ToString**()**+" to "

+\_toAccount.ID.ToString**());**

**}**

**}**

**}**

**}**