(T28)討論 MultiThread(多執行緒)、DeadLock(死鎖定)、Mutex(互斥鎖)
CourseGUID: 29f1196a-1950-41a4-b9c1-dd13a9e92d92
==========
(T28)討論 MultiThread(多執行緒)、DeadLock(死鎖定)、Mutex(互斥鎖)
(T28-1)討論 DeadLock(死鎖定)
(T28-2)討論 LockOrder，解決 DeadLock(死鎖定)
(T28-3)討論 Mutex(互斥鎖)，解決 DeadLock(死鎖定)解法 1
(T28-4)討論 Mutex(互斥鎖)，解決 DeadLock(死鎖定)解法 2
==========
1. New Project
1.1. Create New Project : Sample
2. Sample : Program.cs
==========

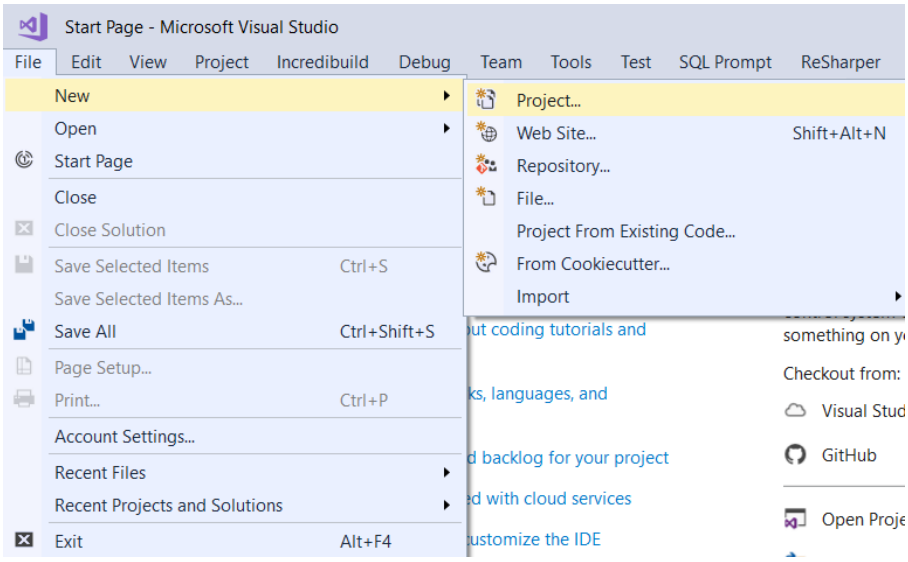
1. New Project

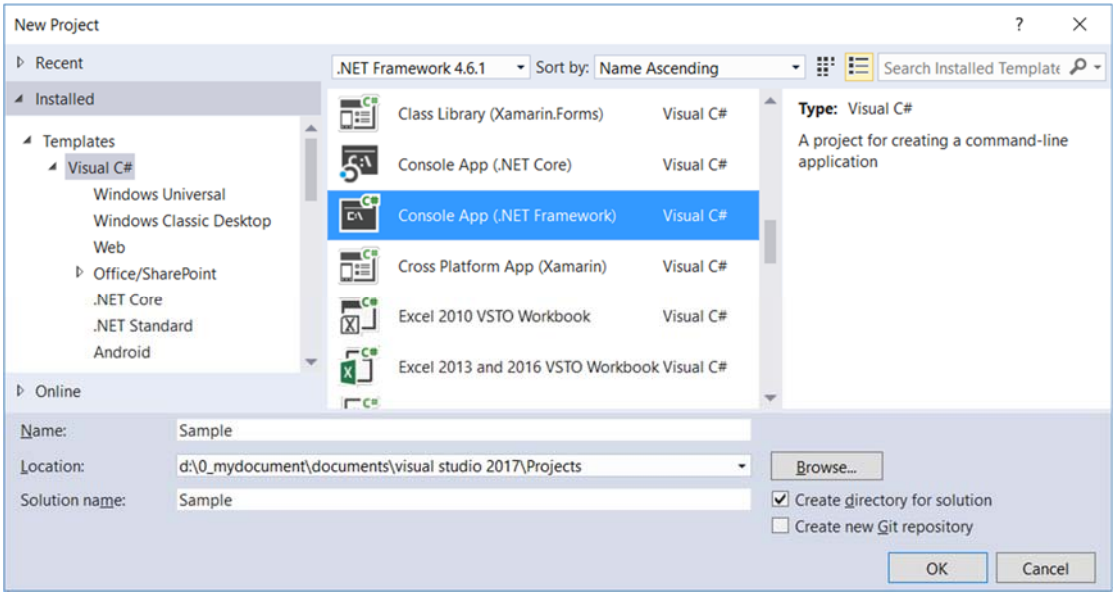
1.1. Create New Project : Sample

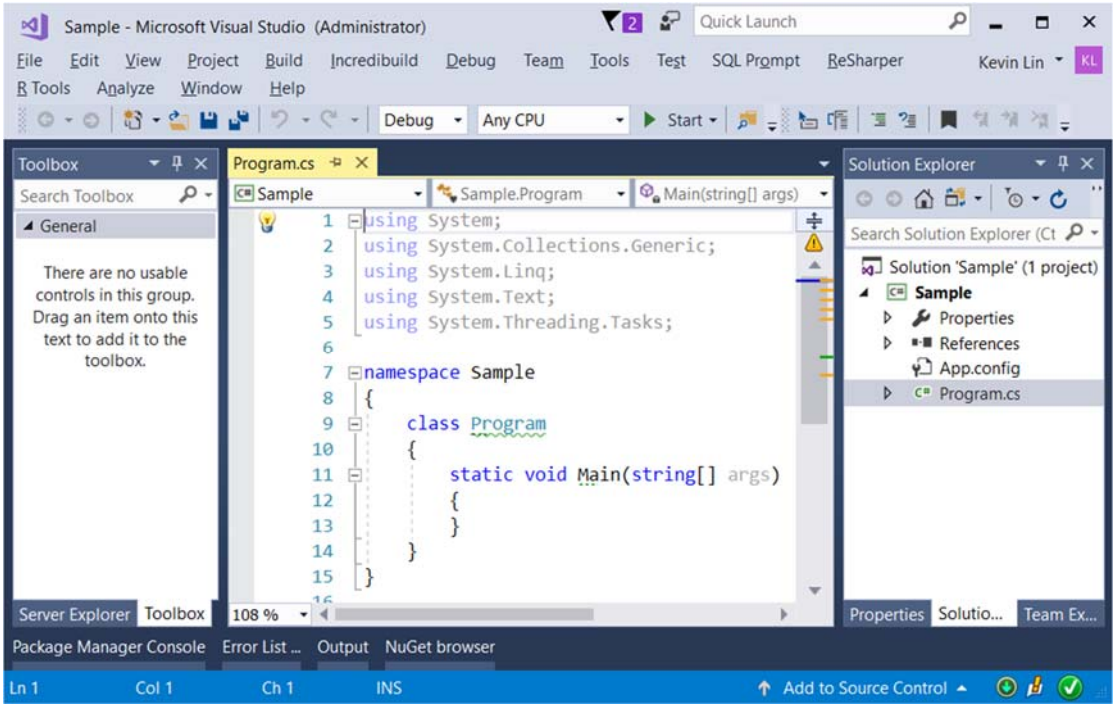
File --> New --> Project... -->

Visual C# --> **Console App (.Net Framework)** -->

Name: **Sample**

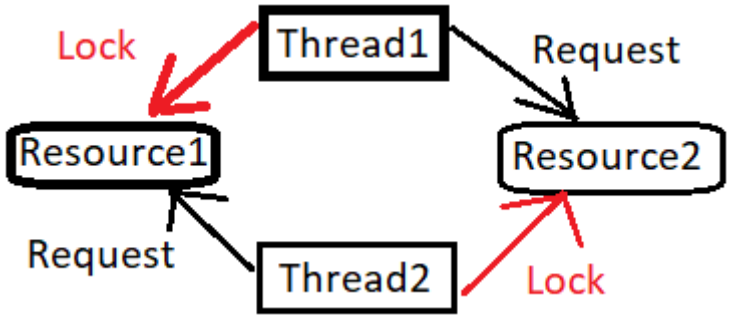






==========

2. Sample：Program.cs



```
using System;
using System.Threading;
using OnLineBanking;



namespace Sample
{
    class Program
    {
        static void Main(string[] args)
        {
            ////1 ========================================
            ////DeadLock Sample
            //Console.WriteLine("1. DeadLockSample() ============================ ");
            //DeadLockSample();


            //2 ========================================
            //We can fix the deadlock issue by the specific lock order.
            Console.WriteLine("2. specific defined lock accuire order can fix DeadLock ================
");
            SpecificDefinedLockAccuireOrder();


            //3 ========================================
            //MutexSample can fix DeadLock V1
            Console.WriteLine("3. Mutex can fix DeadLock V1 ================ ");
            MutexSample();


            //4 ========================================
            //MutexSample can fix DeadLock V2
            Console.WriteLine("4. Mutex can fix DeadLock V2 ================ ");
            MutexSampleV2();


            Console.ReadLine();
        }




        //1 ========================================
        //DeadLock Sample
        static void DeadLockSample()
        {
            Console.WriteLine("Beginning of DeadLockSample() ----------------- ");
            Account accA = new Account(1, 4000);
            Account accB = new Account(2, 2000);


            TransferHelper transferHelperA =
                new TransferHelper(accA, accB, 1000);
            Thread t1 = new Thread(transferHelperA.Transfer)
```

```
            {
                Name = "t1"
            };


            TransferHelper transferHelperB = new
                TransferHelper(accB, accA, 500);
            Thread t2 = new Thread(transferHelperB.Transfer)
            {
                Name = "t2"
            };


            t1.Start();
            t2.Start();
            t1.Join();
            t2.Join();


            Console.WriteLine("End of DeadLockSample() ------------------ ");
        }




        //2 ========================================
        //specific defined lock accuire order can fix DeadLock,
        //but it might get some unexpect result without care.
        static void SpecificDefinedLockAccuireOrder()
        {
            Console.WriteLine("Beginning of SpecificDefinedLockAccuireOrder() ------------------ ");
            Account accA = new Account(1, 4000);
            Account accB = new Account(2, 2000);


            TransferHelper2 transferHelperA =
                new TransferHelper2(accA, accB, 1000);
            Thread t1 = new Thread(transferHelperA.Transfer)
            {
                Name = "t1"
            };


            TransferHelper2 transferHelperB = new
                TransferHelper2(accB, accA, 500);
            Thread t2 = new Thread(transferHelperB.Transfer)
            {
                Name = "t2"
            };


            t1.Start();
            t2.Start();
            t1.Join();
            t2.Join();


            Console.WriteLine($"accA.id=={accA.Id},accA.Balance=={accA.Balance} ;
accB.id=={accB.Id},accB.Balance=={accB.Balance}");
            Console.WriteLine("End of SpecificDefinedLockAccuireOrder() ------------------ ");
```

```
        }



        //3 ========================================
        //MutexSample can fix DeadLock V1
        static void MutexSample()
        {
            Console.WriteLine("Beginning of MutexSample() ------------------ ");
            AccountA accA = new AccountA(1, 4000);
            AccountA accB = new AccountA(2, 2000);


            Thread t1 = new Thread(() => accB.TransferFrom(accA, 1000))
            {
                Name = "t1"
            };


            Thread t2 = new Thread(() => accA.TransferFrom(accB, 500))
            {
                Name = "t2"
            };


            t1.Start();
            t2.Start();
            t1.Join();
            t2.Join();


            Console.WriteLine($"accA.id=={accA.Id},accA.Balance=={accA.Balance} ;
accB.id=={accB.Id},accB.Balance=={accB.Balance}");
            Console.WriteLine("End of MutexSample() ------------------ ");
        }




        //4 ========================================
        //MutexSample can fix DeadLock V2
        static void MutexSampleV2()
        {
            Console.WriteLine("Beginning of SpecificDefinedLockAccuireOrder() ------------------ ");
            AccountB accA = new AccountB(1, 4000);
            AccountB accB = new AccountB(2, 2000);


            TransferHelper4 transferHelperA =
                new TransferHelper4(accA, accB, 1000);
            Thread t1 = new Thread(transferHelperA.Transfer)
            {
                Name = "t1"
            };


            TransferHelper4 transferHelperB = new
                TransferHelper4(accB, accA, 500);
```

```
            Thread t2 = new Thread(transferHelperB.Transfer)
            {
                Name = "t2"
            };


            t1.Start();
            t2.Start();
            t1.Join();
            t2.Join();


            Console.WriteLine($"accA.id=={accA.Id},accA.Balance=={accA.Balance} ;
accB.id=={accB.Id},accB.Balance=={accB.Balance}");
            Console.WriteLine("End of SpecificDefinedLockAccuireOrder() ------------------ ");
        }
    }
}



namespace OnLineBanking
{
    //1 ========================================
    //DeadLock Sample
    public class Account
    {
        public double Balance { get; set; }
        public int Id { get; }
        public Account(int id, double balance)
        {
            Id = id;
            Balance = balance;
        }
        public void Withdraw(double amount)
        {
            Balance -= amount;
        }
        public void Deposit(double amount)
        {
            Balance += amount;
        }
    }




    //============================
    //1.
    //這部分我們將討論如何產生 DeadLock
    public class TransferHelper
    {
        Account _fromAccount;
        Account _toAccount;
        double _amount;




        public TransferHelper(Account fromAccount,
```

```
            Account toAccount, double amount)
        {
            _fromAccount = fromAccount;
            _toAccount = toAccount;
            _amount = amount;
        }



        public void Transfer()
        {
            Console.WriteLine(
                $"Beginning of Transfer, Thread.CurrentThread.Name=={Thread.CurrentThread.Name},
_fromAccount.Id=={_fromAccount.Id} , _toAccount.Id=={_toAccount.Id}, _amount=={_amount}");


            Console.WriteLine($"Thread.CurrentThread.Name=={Thread.CurrentThread.Name}, is about to
lock(_fromAccount), and _fromAccount.Id=={_fromAccount.Id}");
           lock (_fromAccount)
            {
                Console.WriteLine($"Thread.CurrentThread.Name=={Thread.CurrentThread.Name},
lock(_fromAccount) acquired lock on _fromAccount.Id=={_fromAccount.Id}");
                Thread.Sleep(1000); // wait for 1000 milliseconds

                Console.WriteLine($"Thread.CurrentThread.Name=={Thread.CurrentThread.Name}, is about to
lock(_toAccount), and _toAccount.Id=={_toAccount.Id}");
               lock (_toAccount)
                {
                    Console.WriteLine($"Thread.CurrentThread.Name=={Thread.CurrentThread.Name},
lock(_toAccount) acquired lock on _toAccount.Id=={_toAccount.Id}");
                    _fromAccount.Withdraw(_amount);
                    _toAccount.Deposit(_amount);
                }
            }



            Console.WriteLine($"End of Transfer, Thread.CurrentThread.Name=={Thread.CurrentThread.Name},
_fromAccount.Id=={_fromAccount.Id} , _toAccount.Id=={_toAccount.Id}, _amount=={_amount}");
        }
    }
    /*
    1. DeadLockSample() ============================
    Beginning of DeadLockSample() ------------------
    Beginning of Transfer, Thread.CurrentThread.Name==t1, _fromAccount.Id==1 , _toAccount.Id==2,
_amount==1000
    Thread.CurrentThread.Name==t1, is about to lock(_fromAccount), and _fromAccount.Id==1
    Thread.CurrentThread.Name==t1, lock(_fromAccount) acquired lock on _fromAccount.Id==1
    Beginning of Transfer, Thread.CurrentThread.Name==t2, _fromAccount.Id==2 , _toAccount.Id==1,
_amount==500
    Thread.CurrentThread.Name==t2, is about to lock(_fromAccount), and _fromAccount.Id==2
    Thread.CurrentThread.Name==t2, lock(_fromAccount) acquired lock on _fromAccount.Id==2
    Thread.CurrentThread.Name==t1, is about to lock(_toAccount), and _toAccount.Id==2
    Thread.CurrentThread.Name==t2, is about to lock(_toAccount), and _toAccount.Id==1
    -------------------------
    A.
    這邊我們將討論, 如何產生 DeadLock
    A.1.
    thread t1 一開始會鎖住 fromAccount, 也就是鎖住 accA
    ---------
    A.2.
```

```
    同一時間, thread t2 也開始跑了
    thread t2 一開始會鎖住 fromAccount, 也就是鎖住 accB
    ---------
    A.3.
    然後, thread t1 繼續跑
    thread t1 準備要鎖住 toAccount, 也就是鎖住 accB
    結果 thread t1 發現 accB 早已經被 thread t2 鎖住了
    ---------
    A.4.
    然後, thread t2 繼續跑
    thread t2 準備要鎖住 toAccount, 也就是鎖住 accA
    結果 thread t2 發現 accA 早已經被 thread t1 鎖住了
    ---------
    A.5.
    這下尷尬了, 因為這形成了 DeadLock
    那我們接下來討論怎麼解決 DeadLock
    */




    //2 ========================================
    //We can fix the deadlock issue by the specific lock order.
    public class TransferHelper2
    {
        Account _fromAccount;
        Account _toAccount;
        double _amount;



        public TransferHelper2(Account fromAccount,
            Account toAccount, double amount)
        {
            _fromAccount = fromAccount;
            _toAccount = toAccount;
            _amount = amount;
        }



        public void Transfer()
        {
            Console.WriteLine(
                $"Beginning of Transfer, Thread.CurrentThread.Name=={Thread.CurrentThread.Name},
_fromAccount.Id=={_fromAccount.Id} , _fromAccount.Balance=={_fromAccount.Balance},
_toAccount.Id=={_toAccount.Id}, _toAccount.Balance=={_toAccount.Balance}, _amount=={_amount}");


            //***check the comment
            object _lockA =
                _fromAccount.Id < _toAccount.Id ?
                _fromAccount :
                _toAccount;
            object _lockB =
                _fromAccount.Id > _toAccount.Id ?
                _fromAccount :
                _toAccount;
```

```
            Console.WriteLine($"Thread.CurrentThread.Name=={Thread.CurrentThread.Name}, is about to
lock(_lockA), and ((Account)_lockA).Id=={((Account)_lockA).Id}");
            lock (_lockA)
            {
                Console.WriteLine($"Thread.CurrentThread.Name=={Thread.CurrentThread.Name}, lock(_lockA)
acquired lock on ((Account)_lockA).Id=={((Account)_lockA).Id}");
                Thread.Sleep(1000); // wait for 1000 milliseconds


                Console.WriteLine($"Thread.CurrentThread.Name=={Thread.CurrentThread.Name}, is about to
lock(_lockB), and ((Account)_lockB).Id=={((Account)_lockB).Id}");
                lock (_lockB)
                {
                    Console.WriteLine($"Thread.CurrentThread.Name=={Thread.CurrentThread.Name},
lock(_lockB) acquired lock on ((Account)_lockB).Id=={((Account)_lockB).Id}");
                    _fromAccount.Withdraw(_amount);
                    _toAccount.Deposit(_amount);
                }
            }


            Console.WriteLine(
                $"End of Transfer, Thread.CurrentThread.Name=={Thread.CurrentThread.Name},
_fromAccount.Id=={_fromAccount.Id} , _fromAccount.Balance=={_fromAccount.Balance},
_toAccount.Id=={_toAccount.Id}, _toAccount.Balance=={_toAccount.Balance}, _amount=={_amount}");
        }
    }
    /*
    2. specific defined lock accuire order can fix DeadLock ================
    Beginning of SpecificDefinedLockAccuireOrder() ------------------
    Beginning of Transfer, Thread.CurrentThread.Name==t2, _fromAccount.Id==2 ,
_fromAccount.Balance==2000, _toAccount.Id==1, _toAccount.Balance==4000, _amount==500
    Thread.CurrentThread.Name==t2, is about to lock(_lockA), and ((Account)_lockA).Id==1
    Thread.CurrentThread.Name==t2, lock(_lockA) acquired lock on ((Account)_lockA).Id==1
    Beginning of Transfer, Thread.CurrentThread.Name==t1, _fromAccount.Id==1 ,
_fromAccount.Balance==4000, _toAccount.Id==2, _toAccount.Balance==2000, _amount==1000
    Thread.CurrentThread.Name==t1, is about to lock(_lockA), and ((Account)_lockA).Id==1
    Thread.CurrentThread.Name==t2, is about to lock(_lockB), and ((Account)_lockB).Id==2
    Thread.CurrentThread.Name==t2, lock(_lockB) acquired lock on ((Account)_lockB).Id==2
    End of Transfer, Thread.CurrentThread.Name==t2, _fromAccount.Id==2 , _fromAccount.Balance==1500,
_toAccount.Id==1, _toAccount.Balance==4500, _amount==500
    Thread.CurrentThread.Name==t1, lock(_lockA) acquired lock on ((Account)_lockA).Id==1
    Thread.CurrentThread.Name==t1, is about to lock(_lockB), and ((Account)_lockB).Id==2
    Thread.CurrentThread.Name==t1, lock(_lockB) acquired lock on ((Account)_lockB).Id==2
    End of Transfer, Thread.CurrentThread.Name==t1, _fromAccount.Id==1 , _fromAccount.Balance==3500,
_toAccount.Id==2, _toAccount.Balance==2500, _amount==1000
    accA.id==1,accA.Balance==3500 ; accB.id==2,accB.Balance==2500
    End of SpecificDefinedLockAccuireOrder() ------------------
    -------------------------------
    A.
    4000-1000 = 3000
    3000 + 500 = 3500
    B.
    2000+1000=3000
    3000-500=2500
    -------------------------------
    A.
    A.1.
    我們可以透過
    設定特定的 lock 順序
    來解決 deadlock 問題
    ------------------
```

A.2.
我們知道，我們需要 lock 兩個 account
之前會產生 deadlock 的例子
的原因是因為
我們先 lock FromAccount 才 lock ToAccount
可是
不同的 thread 會有不同的 FromAccount 和 ToAccount
就會有可能產生 deadlock 問題
所以之前的範例才會產生 deadlock

A.3.
現在，為了要解決 deadlock 問題
我們打算
不管什麼 thread
都是先 lock Account Id 比較小的 account
才 lock Account Id 比較大的 account
這樣 deadlock 問題就可以解決了

A.4.
我們舉個例子
假設系統現在只有 2 個 account
分別為 accountId==1 和 accountId==2 這 2 個 account
那麼，我們打算
不管什麼 thread
都是先 lock Account Id 比較小的 account
才 lock Account Id 比較大的 account
所以

Thread t1 "先" 把 accountId==1 的 account 鎖起來
接著，Thread t2 也開始跑了
Thread t2 "先嘗試" 把 accountId==1 的 account 鎖起來
結果 Thread t2 發現發現 accountId==1 的 account
已經被其他 thread 鎖住了 (其實是被 Thread t1 鎖住)
所以 Thread t2 只好等待

接著，Thread t1 "再" 把 accountId==2 的 account 鎖起來
所以目前
Thread t1 已經 "同時鎖住" accountId==1 和 accountId==2 這 2 個 account
等 Thread t1 執行完 Transfer 的動作後
這 2 個 account 被解鎖了

Thread t2 發現 accountId==1 被解鎖了
所以 Thread t2 "馬上先" 把 accountId==1 的 account 鎖起來
接著，Thread t2 "再" 把 accountId==2 的 account 鎖起來
所以目前
Thread t2 已經 "同時鎖住" accountId==1 和 accountId==2 這 2 個 account
等 Thread t2 執行完 Transfer 的動作後
這 2 個 account 被解鎖了

A.5.
以上的例子，每次跑都會不一樣
有可能 Thread t2 先跑
然後 Thread t2 "先同時鎖住" accountId==1 和 accountId==2 這 2 個 account
或是
有可能 Thread t1 先跑
然後 Thread t1 "先同時鎖住" accountId==1 和 accountId==2 這 2 個 account
----
We can fix the deadlock issue by the specific lock order.
For example, in this case,
we can always lock the account which id is smaller first,
then lock the account which id is bigger.
In this sample, we can fix the deadlock issue by the specific lock order easily.

```
However, the real world sample is always more complex,
the specific lock order still might get some un-expect result.
*/



//3 ========================================
//MutexSample can fix DeadLock V1
public class AccountA
{
    public int Id { get; set; }
    public double Balance { get; set; }
    Mutex _mutexLock = new Mutex();



    public AccountA(int id, double balance)
    {
        Id = id;
        Balance = balance;
    }



    //check the comment
    public void Withdraw(double amount)
    {
        if (!_mutexLock.WaitOne()) return;
        try
        {
            Balance -= amount;
        }
        finally
        {
            _mutexLock.ReleaseMutex();
        }
    }



    public void Deposit(double amount)
    {
        if (!_mutexLock.WaitOne()) return;
        try
        {
            Balance += amount;
        }
        finally
        {
            _mutexLock.ReleaseMutex();
        }
    }
```

```
        //Transfer the "amount" from "fromAcc" into current object, which is "toAcc"
        public void TransferFrom(AccountA fromAcc, double amount)
        {
            Console.WriteLine(
                $"Beginning of TransferFrom, Thread.CurrentThread.Name=={Thread.CurrentThread.Name},
fromAcc.Id=={fromAcc.Id} , fromAcc.Balance=={fromAcc.Balance}, _toAccount.Id==this.Id=={Id},
_toAccount.Balance==this.Balance=={Balance}, amount=={amount}");


            //***check the comment
            Mutex[] mutexlocks = { _mutexLock, fromAcc._mutexLock };


            if (WaitHandle.WaitAll(mutexlocks))
            {
                //***check the comment
                try
                {
                    fromAcc.Withdraw(amount); //從 fromAcc 取出錢來
                    Deposit(amount); //把錢放進 toAcc
                }
                finally
                {
                    foreach (Mutex mutexlockItem in mutexlocks)
                    {
                        //***check the comment
                        mutexlockItem.ReleaseMutex();
                    }
                }
            }


            Console.WriteLine(
                $"End of TransferFrom, Thread.CurrentThread.Name=={Thread.CurrentThread.Name},
fromAcc.Id=={fromAcc.Id} , fromAcc.Balance=={fromAcc.Balance}, _toAccount.Id==this.Id=={Id},
_toAccount.Balance==this.Balance=={Balance}, amount=={amount}");
        }
    }
    /*
    3. Mutex can fix DeadLock V1 =================
    Beginning of MutexSample() ------------------
    Beginning of TransferFrom, Thread.CurrentThread.Name==t2, fromAcc.Id==2 , fromAcc.Balance==2000,
_toAccount.Id==this.Id==1, _toAccount.Balance==this.Balance==4000, amount==500
    Beginning of TransferFrom, Thread.CurrentThread.Name==t1, fromAcc.Id==1 , fromAcc.Balance==4000,
_toAccount.Id==this.Id==2, _toAccount.Balance==this.Balance==2000, amount==1000
    End of TransferFrom, Thread.CurrentThread.Name==t2, fromAcc.Id==2 , fromAcc.Balance==1500,
_toAccount.Id==this.Id==1, _toAccount.Balance==this.Balance==4500, amount==500
    End of TransferFrom, Thread.CurrentThread.Name==t1, fromAcc.Id==1 , fromAcc.Balance==3500,
_toAccount.Id==this.Id==2, _toAccount.Balance==this.Balance==2500, amount==1000
    accA.id==1,accA.Balance==3500 ; accB.id==2,accB.Balance==2500
    End of MutexSample() ------------------
    -------------------------
    A.
    //public void Withdraw(double amount)
    //{
    //    if (!_mutexLock.WaitOne()) return;
    //    try
    //    {
    //        Balance -= amount;
    //    }
    //    finally
    //    {
```

```
//        _mutexLock.ReleaseMutex();
//    }
//}
-----------------
A.1.
--------------
A.1.1.
Threading.WaitHandle.WaitOne()
Blocks the current thread until the current
System.Threading.WaitHandle receives a signal.
Returns true if the current instance receives a signal.
Returns false if the current instance is never signaled,
System.Threading.WaitHandle.WaitOne(System.Int32, System.Boolean) never returns.
--------------
A.1.2.
Use mutex to lock the thread.
//if (!_mutexLock.WaitOne()) return;
_mutexLock.WaitOne() return false means
the current thread fails to lock,
so it has to go back to the previous thread.
-----
_mutexLock.WaitOne() return true means
the current thread is successfully locked,
so it may continue to work.
-----------------
A.2.
--------------
A.2.1.
Threading.WaitHandle.WaitOne()
這意思是把 current thread 暫時暫停
然後看看可不可以得到一個 signal
這邊的 signal 有點像是接力賽跑的"接力棒"
--------------
A.2.2.
//if (!_mutexLock.WaitOne()) return;
_mutexLock.WaitOne()
Returns false if the current thread does not receive a signal
回傳 false 如果 current thread 沒有  收到/搶到  signal(接力棒)
------
這邊的意思是
如果 "沒有" 接收到/搶到  signal(接力棒)
則代表 current thread  "沒有" 成功的把 該資源鎖住
所以就不能繼續跑下去
這裡沒收到 Signal 你可以想像成,
沒成功搶到"接力棒",
代表沒有 lock 成功,
就只能等待下回看看可不可以  收到/搶到 Signal (接力棒),
--------------
A.2.3.
_mutexLock.WaitOne()
Returns true if the current thread receive a signal
回傳 true 如果 current thread 有  收到/搶到  signal(接力棒)
------
如果有接收到 signal(接力棒)
代表有 current thread 有成功的把該資源鎖住
則可以繼續往下做
//    try
//    {
//        Balance -= amount;
//    }
然後做完後,
就可以把  這個 signal(接力棒)再重新傳出去
就可以讓 其他的 thread 來搶 這個 signal(接力棒)
//    finally
//    {
```

```
//        _mutexLock.ReleaseMutex();
//    }
-----------------------------------------------
B.
//public void TransferFrom(AccountA fromAcc, double amount)
//{
//    Mutex[] mutexlocks = { _mutexLock, fromAcc._mutexLock };
//    if (WaitHandle.WaitAll(mutexlocks))
//    {
//        try
//        {
//            fromAcc.Withdraw(amount); //從 fromAcc 取出錢來
//            Deposit(amount); //把錢放進 toAcc
//        }
//        finally
//        {
//            foreach (Mutex mutexlockItem in mutexlocks)
//            {
//                mutexlockItem.ReleaseMutex();
//            }
//        }
//    }
//}
-------------------------------------------
B.1.
_mutexLock
-------------------------------------
B.1.1.
//public void TransferFrom(AccountA fromAcc, double amount)
Transfer the "amount" from "fromAcc" into current object, which is "toAcc"
-------------------------------------
B.1.2.
//if (WaitHandle.WaitAll(mutexlocks))
Waits for all the elements
in the specified array to receive a signal.
-------------------------------------------
B.2.
-------------------------------------
B.2.1.
//public void TransferFrom(AccountA fromAcc, double amount)
這個 method 的目的是 將 amount 從 fromAcc 轉進 toAccount
amount 是要轉帳的金額
toAccount 是目前的 current object
-------------------------------------
B.2.2.
//Mutex[] mutexlocks = { _mutexLock, fromAcc._mutexLock };
fromAcc 和 toAcc 裡面都有 _mutexLock 也就是 Mutex 物件
我們需要把 fromAcc 和 toAcc 裡面的_mutexLock 都先取出來
-------------------------------------
B.2.3.
//if (WaitHandle.WaitAll(mutexlocks))
如果把 toAcc 的 _mutexLock 和  fromAcc 的 _mutexLock 都拿到手的話
有點像是
如果 toAcc 和 fromAcc 都鎖起來的話
就去做
//fromAcc.Withdraw(amount); //從 fromAcc 取出錢來
//Deposit(amount); //把錢放進 toAcc
然後最後
釋放出 fromAcc 的_mutexLock   和  toAcc 的_mutexLock
也就是解鎖 fromAcc 和 toAcc
//foreach (Mutex mutexlockItem in mutexlocks)
//{
//    mutexlockItem.ReleaseMutex();
//}
-------------------------------------------
```

```
B.3.
結論:
accA.
4000-1000 = 3000
3000 + 500 = 3500
accB.
2000+1000=3000
3000-500=2500
預期的結果, 和執行出來的結果相符合
*/




//4 ========================================
//MutexSample can fix DeadLock V2
public class AccountB
{
    public int Id { get; set; }
    public double Balance { get; set; }
    public Mutex _mutexLock = new Mutex();



    public AccountB(int id, double balance)
    {
        Id = id;
        Balance = balance;
    }



    public void Withdraw(double amount)
    {
        if (!_mutexLock.WaitOne()) return;
        try
        {
            Balance -= amount;
        }
        finally
        {
            _mutexLock.ReleaseMutex();
        }
    }



    public void Deposit(double amount)
    {
        if (!_mutexLock.WaitOne()) return;
        try
        {
            Balance += amount;
        }
        finally
        {
            _mutexLock.ReleaseMutex();
```

```
        }
    }
}
```

```
public class TransferHelper4
{
    AccountB _fromAccount;
    AccountB _toAccount;
    double _amount;

    public TransferHelper4(AccountB fromAccount,
        AccountB toAccount, double amount)
    {
        _fromAccount = fromAccount;
        _toAccount = toAccount;
        _amount = amount;
    }

    public void Transfer()
    {
        //***check the comment
        //Mutex[] mutexlocks = { _fromAccount._mutexLock, _toAccount._mutexLock };
        WaitHandle[] mutexlocks = { _fromAccount._mutexLock, _toAccount._mutexLock };

        if (WaitHandle.WaitAll(mutexlocks))
        {
            //***check the comment
            try
            {
                _fromAccount.Withdraw(_amount); //從 fromAcc 取出錢來
                _toAccount.Deposit(_amount); //把錢放進 toAcc
            }
            finally
            {
                foreach (var waitHandle in mutexlocks)
                {
                    Mutex mutexlockItem = (Mutex)waitHandle;
                    mutexlockItem.ReleaseMutex();
                }
            }
        }
    }
}
/*
4. Mutex can fix DeadLock ================
Beginning of SpecificDefinedLockAccuireOrder() ------------------
accA.id==1,accA.Balance==3500 ; accB.id==2,accB.Balance==2500
End of SpecificDefinedLockAccuireOrder() ------------------
---------------------------------
A.
//public void Transfer()
```

```
//{
//    //***check the comment
//    WaitHandle[] mutexlocks = { _fromAccount._mutexLock, _toAccount._mutexLock };
//    if (WaitHandle.WaitAll(mutexlocks))
//    {
//        //***check the comment
//        try
//        {
//            _fromAccount.Withdraw(_amount); //從 fromAcc 取出錢來
//            _toAccount.Deposit(_amount); //把錢放進 toAcc
//        }
//        finally
//        {
//            foreach (var waitHandle in mutexlocks)
//            {
//                Mutex mutexlockItem = (Mutex) waitHandle;
//                mutexlockItem.ReleaseMutex();
//            }
//        }
//    }
//}
----------------------------------------
A.1.
-----------------------------------
A.1.1.
//public void Transfer()
Transfer the "amount" from "fromAcc" into current object, which is "toAcc"
------
這個 method 的目的是 將 amount 從 fromAcc 轉進 toAccount
-----------------------------------
B.1.2.
//Mutex[] mutexlocks = { _fromAccount._mutexLock, _toAccount._mutexLock };
WaitHandle[] mutexlocks = { _fromAccount._mutexLock, _toAccount._mutexLock };
You may use Mutex[] or WaitHandle[],
it will not affect the result
This array contain fromAcc Mutex and toAcc Mutex
------
fromAcc 和 toAcc 裡面都有 _mutexLock 也就是 Mutex 物件
我們需要把 fromAcc 和 toAcc 裡面的_mutexLock 都先取出來
array 型別上使用 Mutex[] 或是 WaitHandle[] 都不會影響結果
-----------------------------------
A.1.3.
//if (WaitHandle.WaitAll(mutexlocks))
Waits for all the elements
in the specified array to receive a signal.
------
如果把 toAcc 的 _mutexLock 和  fromAcc 的 _mutexLock 都拿到手的話
有點像是
如果 toAcc 和 fromAcc 都鎖起來的話
就去做
//fromAcc.Withdraw(amount); //從 fromAcc 取出錢來
//Deposit(amount); //把錢放進 toAcc
然後最後
釋放出 fromAcc 的_mutexLock   和  toAcc 的_mutexLock
也就是解鎖 fromAcc 和 toAcc
//foreach (var waitHandle in mutexlocks)
//{
//    Mutex mutexlockItem = (Mutex) waitHandle;
//    mutexlockItem.ReleaseMutex();
//}
-------------------------------------------
B.
結論:
---------------------------------
B.1.
```

```
    accA.
    4000-1000 = 3000
    3000 + 500 = 3500
    accB.
    2000+1000=3000
    3000-500=2500
    這個 sample 的結果也是 accA==3500 , accB==2500
    沒有問題
    ---------------------------------
    B.2.
    這個 sample 跟上個 sample 的差別在於
    我們把 Transfer method
    從 Account class 裡面刪除
    並且新增一個 Helper class 來處理 transfer 的事情
    所以 Helper class 的 transfer method 算是第三方
    這種寫法，比較符合邏輯一點
    有點像是
    "我是第三方，我鎖住 accA 同時也鎖住 accB"
    可是，上一個範例的寫法有點像是
    "我自己鎖自己，同時 還要去 鎖別人"
    所以，我個人是比較喜歡這個範例的
    */
}
```

```
2. specific defined lock accuire order can fix DeadLock ================
Beginning of SpecificDefinedLockAccuireOrder() ------------------
Beginning of Transfer, Thread.CurrentThread.Name==t1, _fromAccount.Id==1 , _fromAccount.Balance==4000, _toAccount.Id==2,
_toAccount.Balance==2000, _amount==1000
Thread.CurrentThread.Name==t1, is about to lock(_lockA), and ((Account)_lockA).Id==1
Thread.CurrentThread.Name==t1, lock(_lockA) acquired lock on ((Account)_lockA).Id==1
Beginning of Transfer, Thread.CurrentThread.Name==t2, _fromAccount.Id==2 , _fromAccount.Balance==2000, _toAccount.Id==1,
_toAccount.Balance==4000, _amount==500
Thread.CurrentThread.Name==t2, is about to lock(_lockA), and ((Account)_lockA).Id==1
Thread.CurrentThread.Name==t1, is about to lock(_lockB), and ((Account)_lockB).Id==2
Thread.CurrentThread.Name==t1, lock(_lockB) acquired lock on ((Account)_lockB).Id==2
End of Transfer, Thread.CurrentThread.Name==t1, _fromAccount.Id==1 , _fromAccount.Balance==3000, _toAccount.Id==2, _toAc
count.Balance==3000, _amount==1000
Thread.CurrentThread.Name==t2, lock(_lockA) acquired lock on ((Account)_lockA).Id==1
Thread.CurrentThread.Name==t2, is about to lock(_lockB), and ((Account)_lockB).Id==2
Thread.CurrentThread.Name==t2, lock(_lockB) acquired lock on ((Account)_lockB).Id==2
End of Transfer, Thread.CurrentThread.Name==t2, _fromAccount.Id==2 , _fromAccount.Balance==2500, _toAccount.Id==1, _toAc
count.Balance==3500, _amount==500
accA.id==1,accA.Balance==3500 ; accB.id==2,accB.Balance==2500
End of SpecificDefinedLockAccuireOrder() ------------------
3. Mutex can fix DeadLock V1 ================
Beginning of MutexSample() ------------------
Beginning of TransferFrom, Thread.CurrentThread.Name==t1, fromAcc.Id==1 , fromAcc.Balance==4000, _toAccount.Id==this.Id=
=2, _toAccount.Balance==this.Balance==2000, amount==1000
Beginning of TransferFrom, Thread.CurrentThread.Name==t2, fromAcc.Id==2 , fromAcc.Balance==2000, _toAccount.Id==this.Id=
=1, _toAccount.Balance==this.Balance==4000, amount==500
End of TransferFrom, Thread.CurrentThread.Name==t1, fromAcc.Id==1 , fromAcc.Balance==3000, _toAccount.Id==this.Id==2, _t
oAccount.Balance==this.Balance==3000, amount==1000
End of TransferFrom, Thread.CurrentThread.Name==t2, fromAcc.Id==2 , fromAcc.Balance==2500, _toAccount.Id==this.Id==1, _t
oAccount.Balance==this.Balance==3500, amount==500
accA.id==1,accA.Balance==3500 ; accB.id==2,accB.Balance==2500
End of MutexSample() ------------------
4. Mutex can fix DeadLock V2 ================
Beginning of SpecificDefinedLockAccuireOrder() ------------------
accA.id==1,accA.Balance==3500 ; accB.id==2,accB.Balance==2500
End of SpecificDefinedLockAccuireOrder() ------------------
```