Programming C# .Net
Warmup Exercise

1. Create a console application named **ThreadSafety**.

2. For this application use the following code:

Counter.cs

```
using System;
using System.Threading;

namespace ThreadSafety
{
	abstract class Counter
	{
		protected int count = 0;

		public abstract int Read(int threadNum);
		public abstract void Increment(int threadNum);
	}
}
```

CounterLock.cs

```
using System;
using System.Threading;

namespace ThreadSafety
{
	class CounterUsingLock : Counter
	{
		public override int Read(int threadNum)
		{
			lock(this)
			{
				Console.WriteLine(
					"Start Resource reading (Thread={0})count: {1}", threadNum, count);
				Thread.Sleep(250);
				Console.WriteLine(
					"Stop Resource reading (Thread={0}) count: {1}", threadNum, count);
				return count;
			}
		}

		public override void Increment(int threadNum)
		{
			lock(this)
			{
				Console.WriteLine(
					"Start Resource writing (Thread={0}) count: {1}", threadNum, count);
				int tempCount = count;
				Thread.Sleep(1000);
				tempCount++;
				count = tempCount;
				Console.WriteLine(
					"Stop Resource writing (Thread={0}) count: {1}", threadNum, count);
			}
		}
```

```
        }
}
```

CounterUnsafe.cs

```
using System;
using System.Threading;

namespace ThreadSafety
{
        class CounterUnsafe : Counter
        {
                public override int Read(int threadNum)
                {
                        try
                        {
                                Console.WriteLine(
                                        "Start Resource reading (Thread={0})count: {1}", threadNum, count);
                                Thread.Sleep(250);
                                Console.WriteLine(
                                        "Stop  Resource reading (Thread={0}) count: {1}", threadNum, count);
                                return count;
                        }
                        finally
                        {
                        }
                }

                public override void Increment(int threadNum)
                {
                        try
                        {
                                Console.WriteLine(
                                        "Start Resource writing (Thread={0}) count: {1}", threadNum, count);

                                int tempCount = count;
                                Thread.Sleep(1000);
                                tempCount++;
                                count = tempCount;
                                Console.WriteLine(
                                        "Stop  Resource writing (Thread={0}) count: {1}", threadNum, count);
                        }
                        finally
                        {
                        }
                }
        }
}
```

ThreadClient.cs

```
using System;
using System.Threading;

namespace ThreadSafety
{
        class UseThreads
        {
                static Counter counter = null;
                static int totalNumberOfAsyncOps = 10;
                static int numAsyncOps = totalNumberOfAsyncOps;
                static AutoResetEvent asyncOpsAreDone = new AutoResetEvent(false);

                public static void Main()
                {
```

```
                Console.WriteLine("\n\nUnsafe test:");
                asyncOpsAreDone.Reset();
                numAsyncOps = totalNumberOfAsyncOps;
                counter = new CounterUnsafe();
                for (int threadNum = 0; threadNum < numAsyncOps; threadNum++)
                {
                    ThreadPool.QueueUserWorkItem(new WaitCallback(UpdateResource), threadNum);
                }
                asyncOpsAreDone.WaitOne();
                Console.WriteLine("All Unsafe threads have completed.");

                Console.WriteLine("\n\nLock test:");
                asyncOpsAreDone.Reset();
                numAsyncOps = totalNumberOfAsyncOps;
                counter = new CounterUsingLock();
                for (int threadNum = 0; threadNum < numAsyncOps; threadNum++)
                {
                    ThreadPool.QueueUserWorkItem(new WaitCallback(UpdateResource), threadNum);
                }
                asyncOpsAreDone.WaitOne();
                Console.WriteLine("All Lock threads have completed.");
            }

            static void UpdateResource(Object state)
            {
                int threadNum = (int) state;

                if ((threadNum % 2) != 0)
                    counter.Read(threadNum);
                else
                    counter.Increment(threadNum);

                if (( Interlocked.Decrement(ref numAsyncOps)) == 0)
                    asyncOpsAreDone.Set();
            }
        }
}
```

3. Test your application and then save your work.