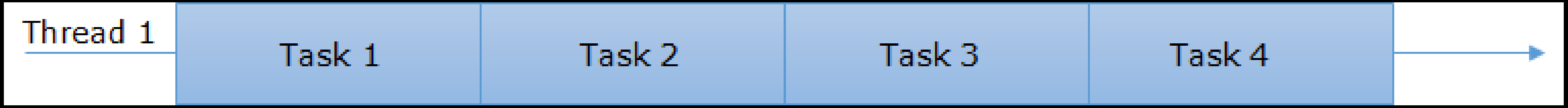
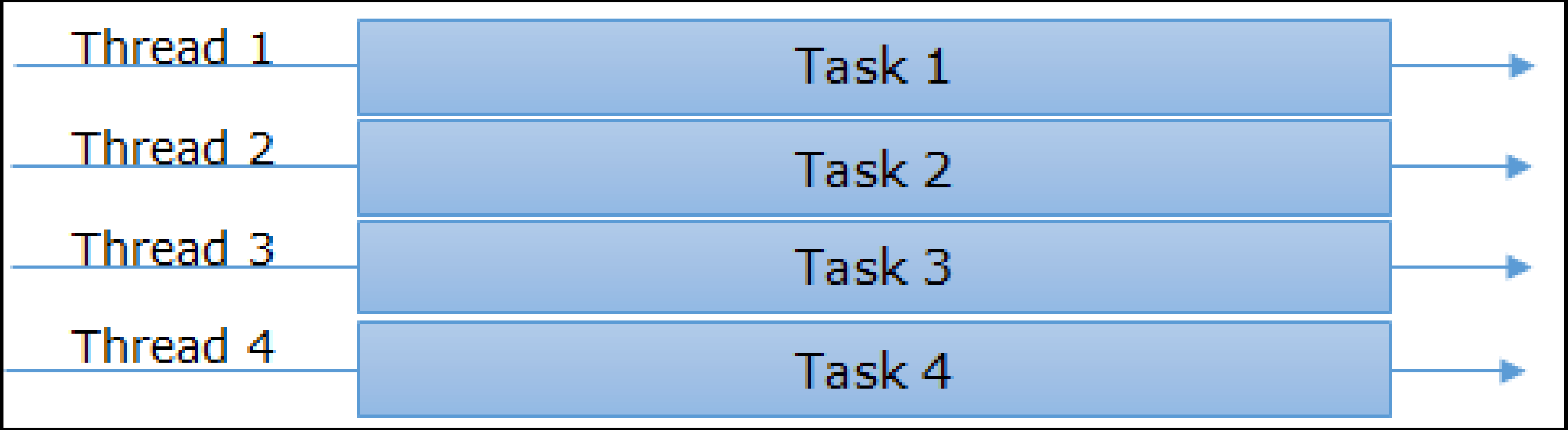
MultiThreading

acync

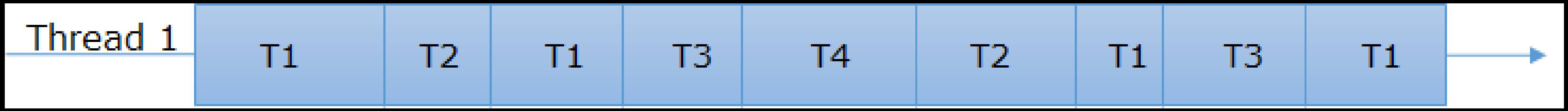
SyncModel.SingleThread



SyncModel.MultiThread



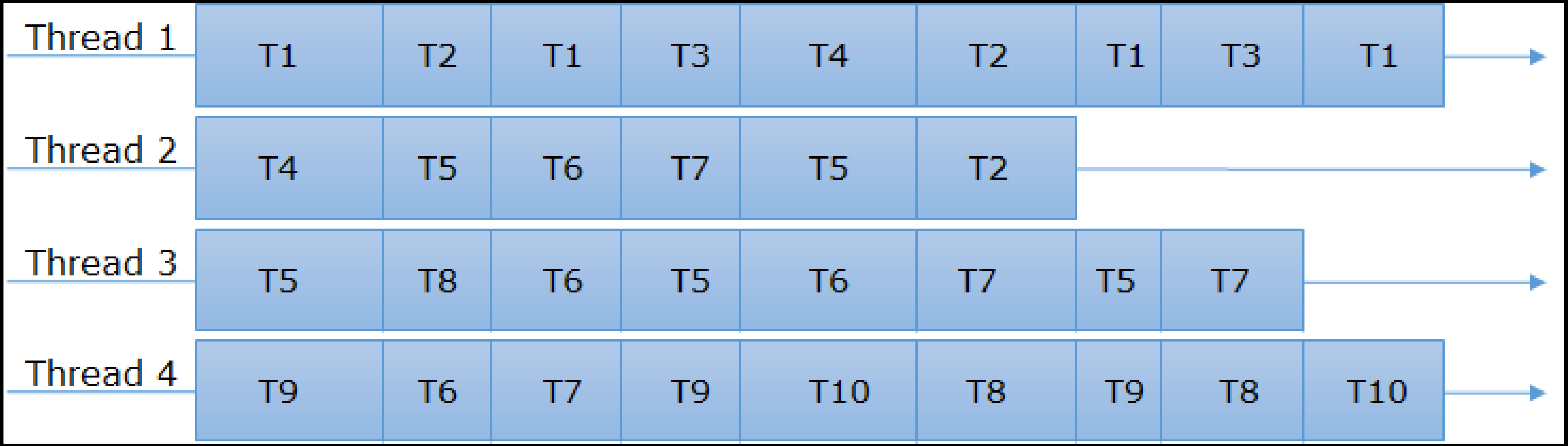
A-SyncModel.SingleThread



Thread 1

T1 | T2 | T1 | T3 | T4 | T2 | T1 | T3 | T1

A-SyncModel.MultiThread

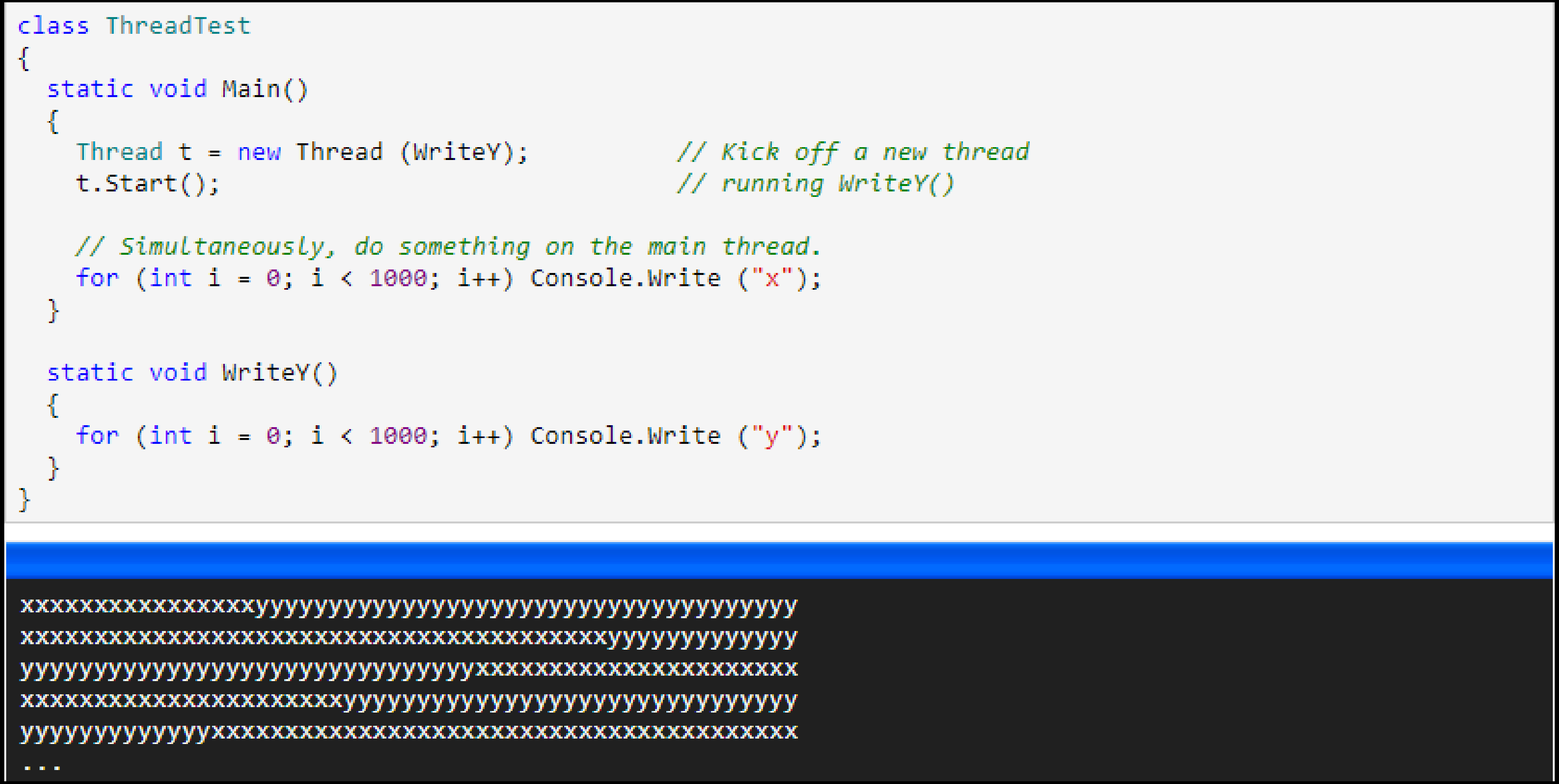


Thread 1	T1	T2	T1	T3	T4	T2	T1	T3	T1
Thread 2	T4	T5	T6	T7	T5	T2			
Thread 3	T5	T8	T6	T5	T6	T7	T5	T7	
Thread 4	T9	T6	T7	T9	T10	T8	T9	T8	T10

Threading Uses

- **Maintaining a responsive user interface**
- **Making efficient use of an otherwise blocked CPU**
- **Parallel programming**
- **Speculative execution**
- **Allowing requests to be processed simultaneously**

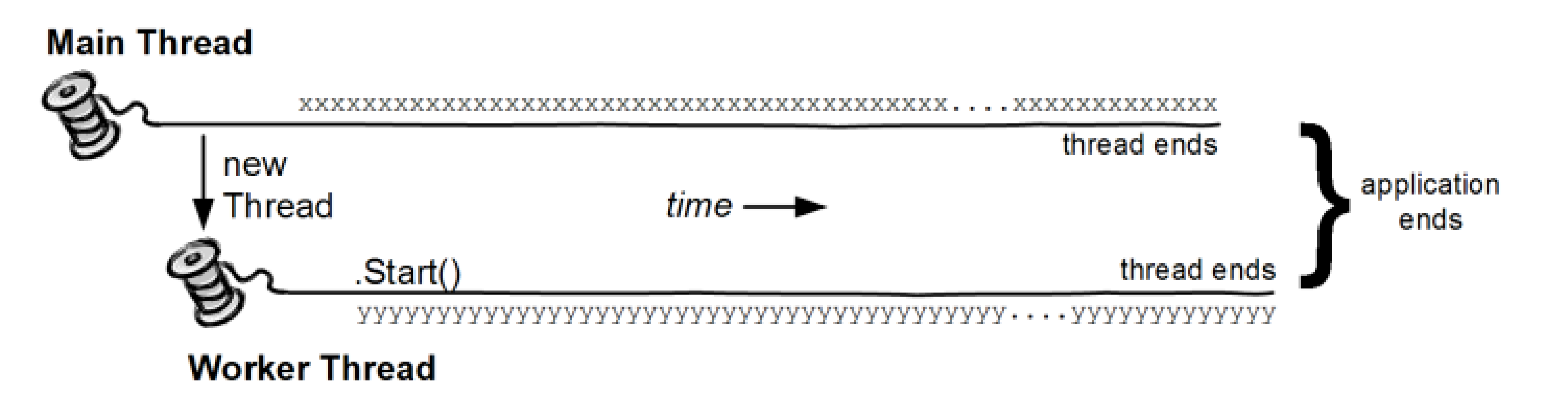
Threads



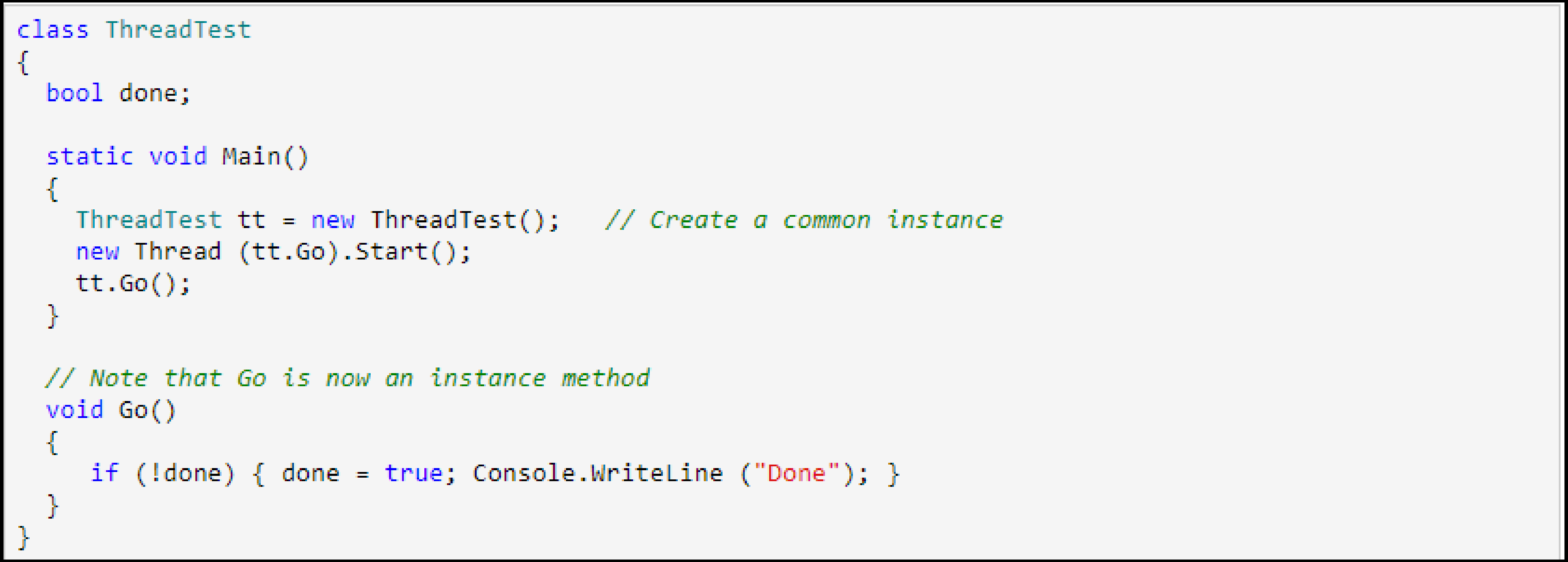
```
class ThreadTest
{
  static void Main()
  {
    Thread t = new Thread (WriteY);          // Kick off a new thread
    t.Start();                               // running WriteY()

    // Simultaneously, do something on the main thread.
    for (int i = 0; i < 1000; i++) Console.Write ("x");
  }

  static void WriteY()
  {
    for (int i = 0; i < 1000; i++) Console.Write ("y");
  }
}
```

```
xxxxxxxxxxxxxxxxyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
...
```

Threads

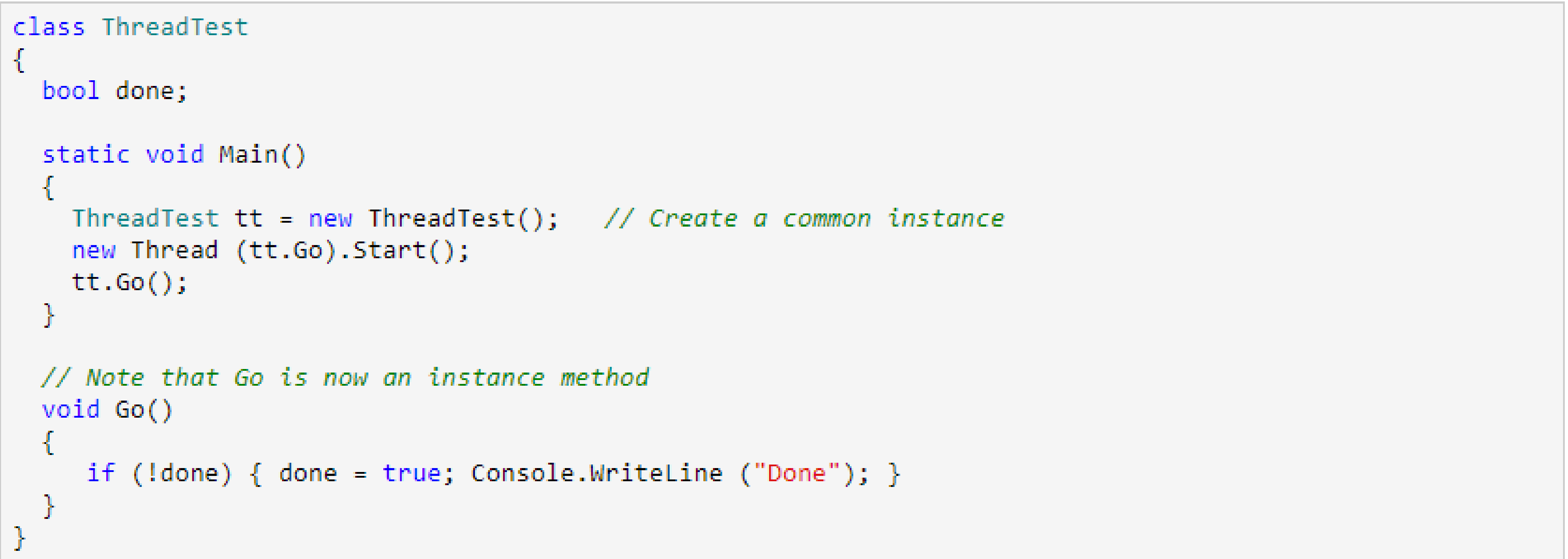

Main Thread
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx....xxxxxxxxxxxxx
thread ends
new
Thread
time
.Start()
thread ends
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy....yyyyyyyyyyyyy
application
ends
Worker Thread

Two Threads

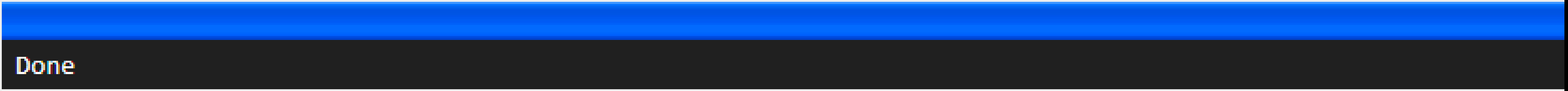


```
class ThreadTest
{
  bool done;

  static void Main()
  {
    ThreadTest tt = new ThreadTest();   // Create a common instance
    new Thread (tt.Go).Start();
    tt.Go();
  }

  // Note that Go is now an instance method
  void Go()
  {
     if (!done) { done = true; Console.WriteLine ("Done"); }
  }
}
```

Two Threads

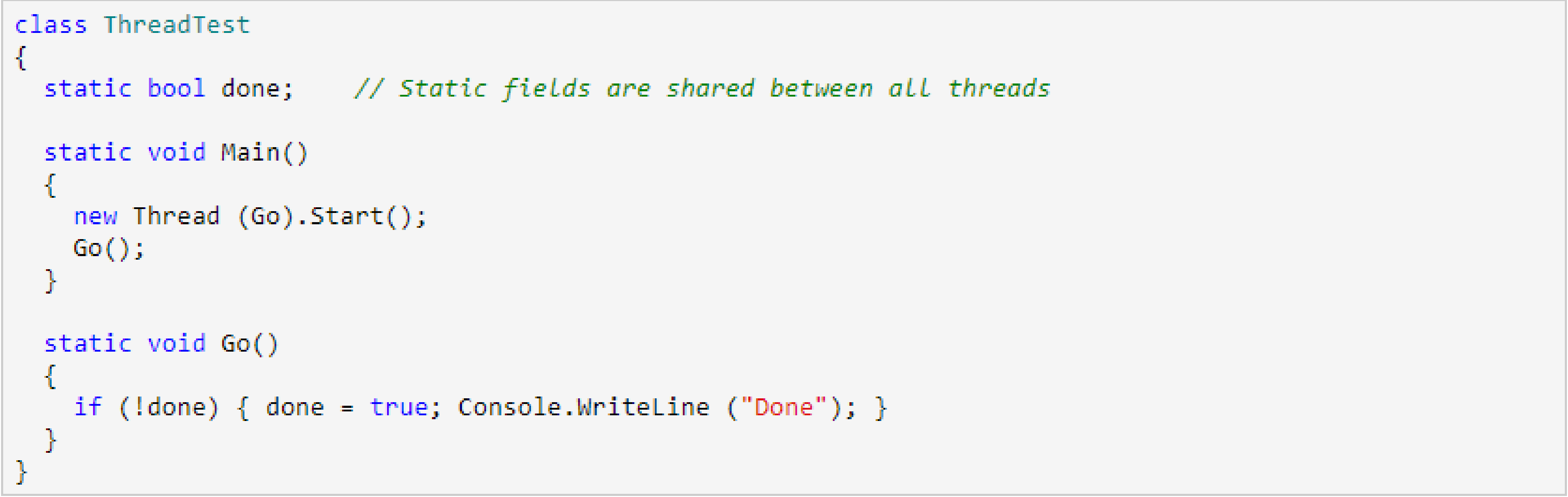


```
class ThreadTest
{
  bool done;

  static void Main()
  {
    ThreadTest tt = new ThreadTest();   // Create a common instance
    new Thread (tt.Go).Start();
    tt.Go();
  }

  // Note that Go is now an instance method
  void Go()
  {
     if (!done) { done = true; Console.WriteLine ("Done"); }
  }
}
```



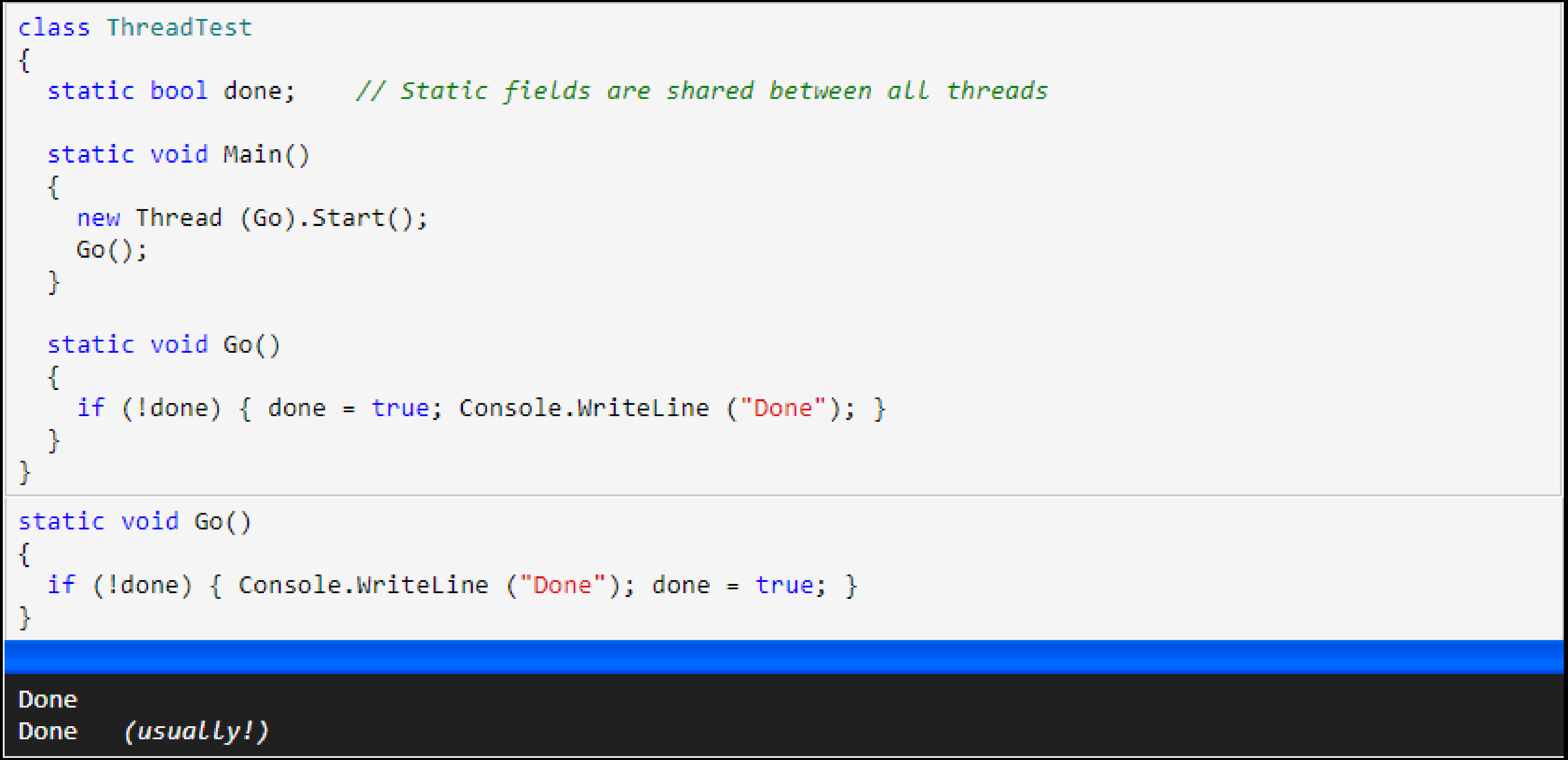
```
Done
```

Two Threads static



```
class ThreadTest
{
  static bool done;    // Static fields are shared between all threads

  static void Main()
  {
    new Thread (Go).Start();
    Go();
  }

  static void Go()
  {
    if (!done) { done = true; Console.WriteLine ("Done"); }
  }
}
```

Two Threads static

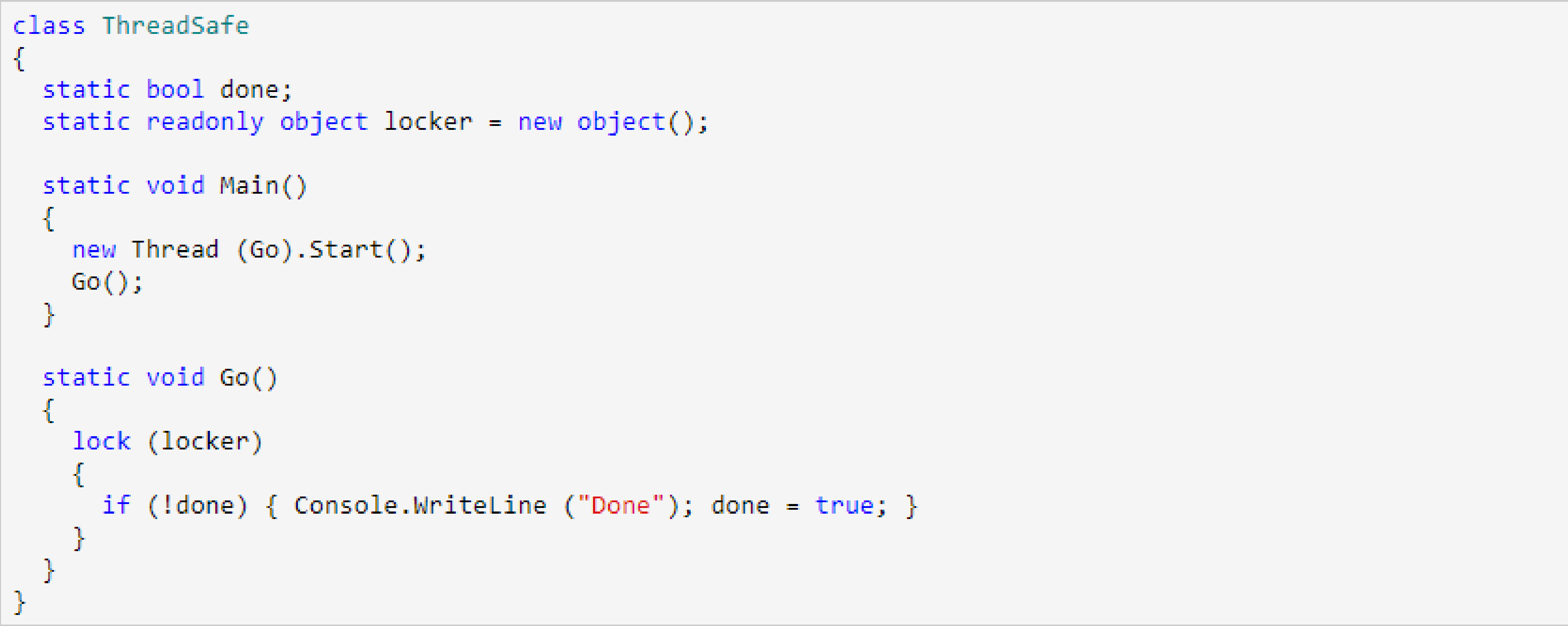


```
class ThreadTest
{
  static bool done;    // Static fields are shared between all threads

  static void Main()
  {
    new Thread (Go).Start();
    Go();
  }

  static void Go()
  {
    if (!done) { done = true; Console.WriteLine ("Done"); }
  }
}
```

```
static void Go()
{
  if (!done) { Console.WriteLine ("Done"); done = true; }
}
```

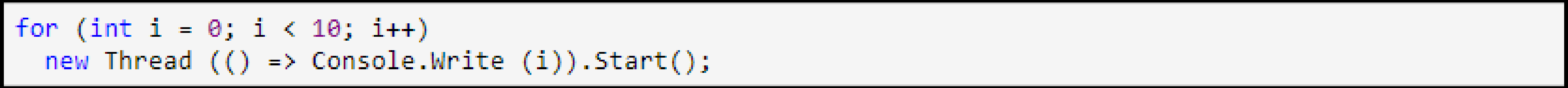
```
Done
Done    (usually!)
```

Two Threads static with Lock



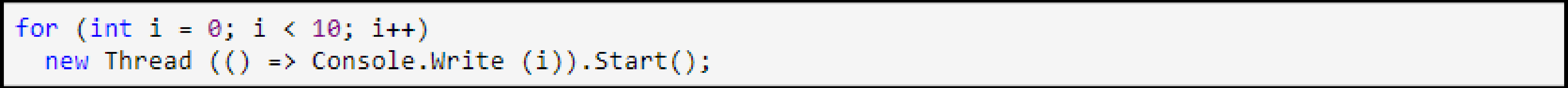
```
class ThreadSafe
{
  static bool done;
  static readonly object locker = new object();

  static void Main()
  {
    new Thread (Go).Start();
    Go();
  }

  static void Go()
  {
    lock (locker)
    {
      if (!done) { Console.WriteLine ("Done"); done = true; }
    }
  }
}
```

Practice

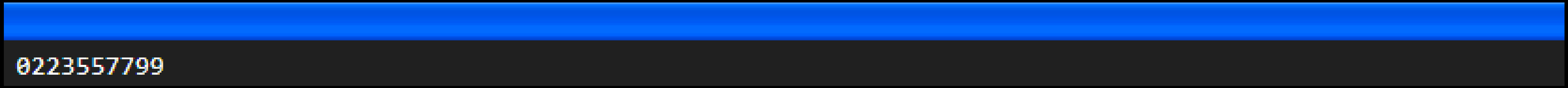


```
for (int i = 0; i < 10; i++)
  new Thread (() => Console.Write (i)).Start();
```

Practice

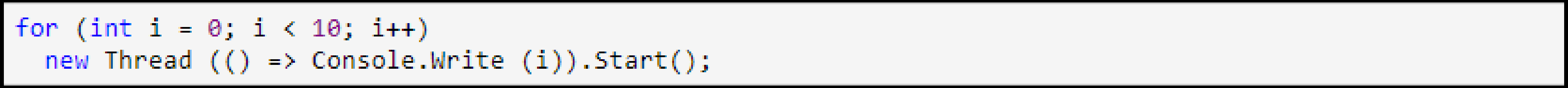


```
for (int i = 0; i < 10; i++)
  new Thread (() => Console.Write (i)).Start();
```

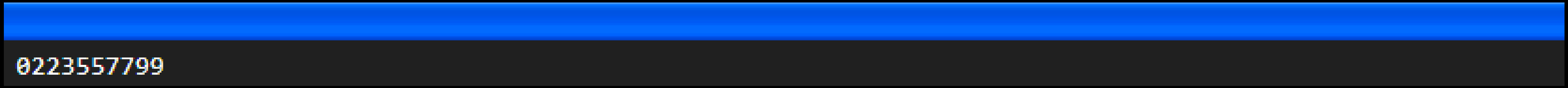


```
0223557799
```

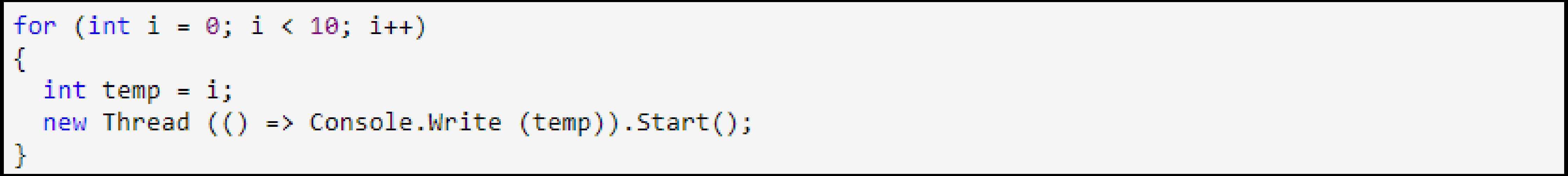
Practice



```
for (int i = 0; i < 10; i++)
  new Thread (() => Console.Write (i)).Start();
```

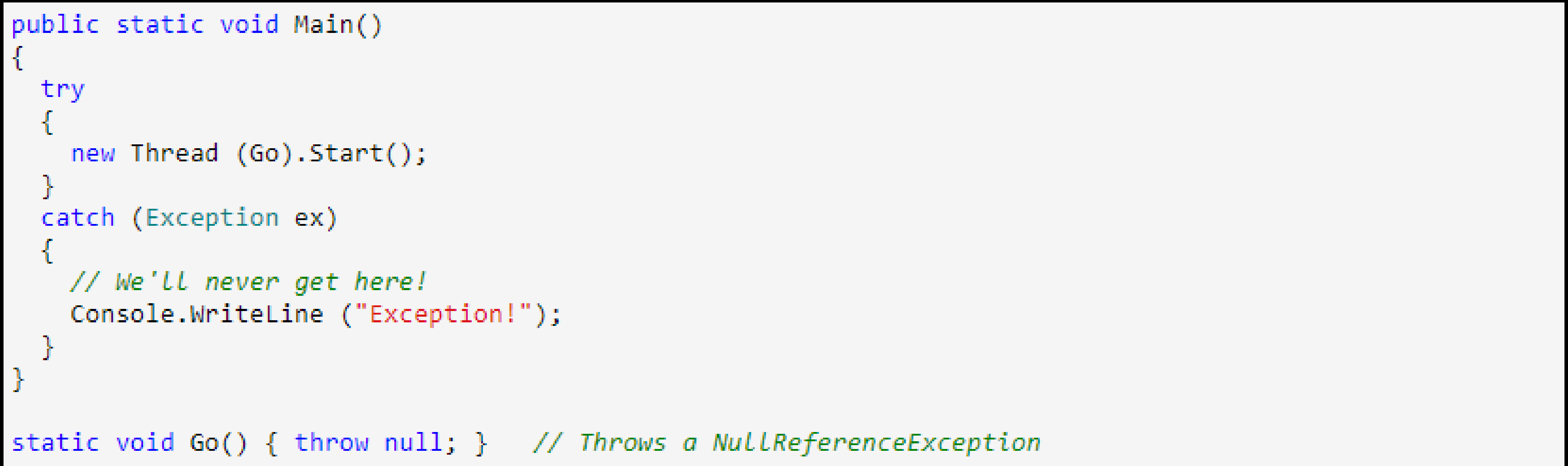


```
0223557799
```



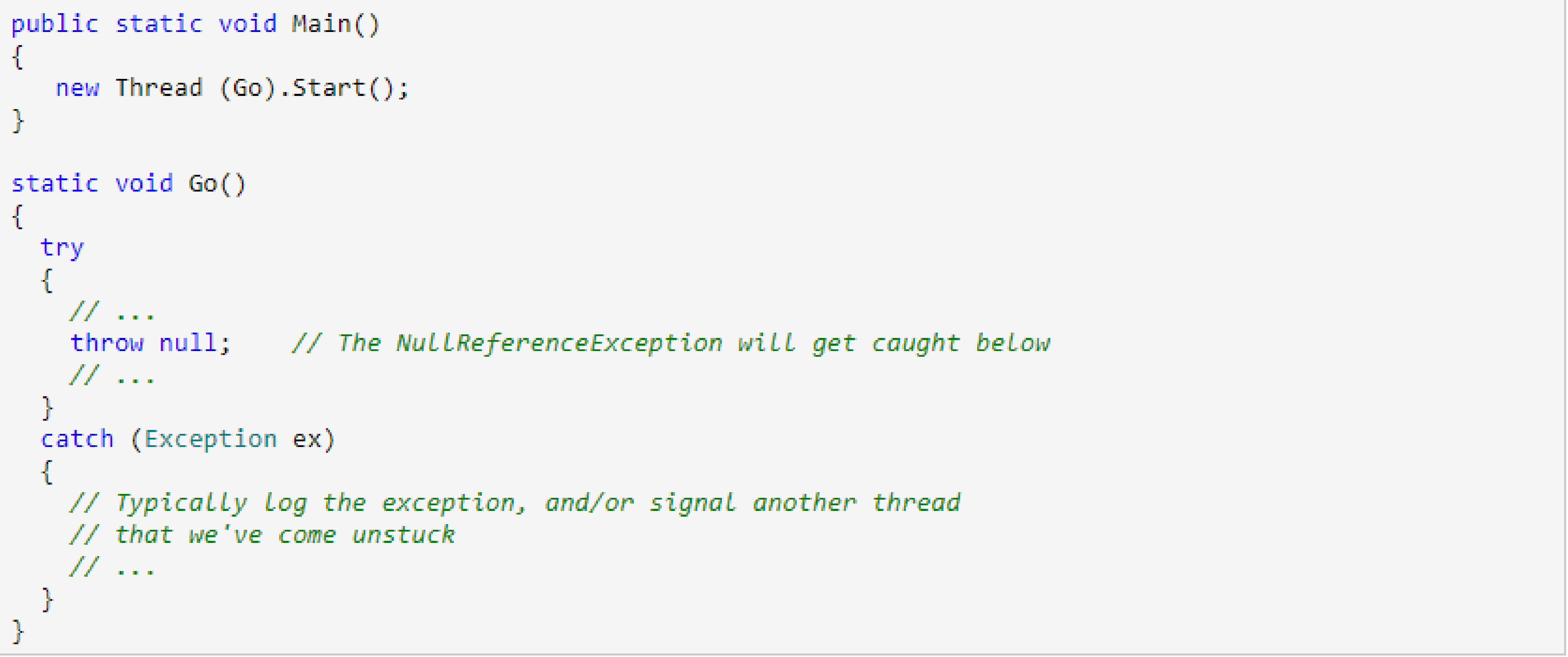
```
for (int i = 0; i < 10; i++)
{
  int temp = i;
  new Thread (() => Console.Write (temp)).Start();
}
```

Exceptions



```
public static void Main()
{
  try
  {
    new Thread (Go).Start();
  }
  catch (Exception ex)
  {
    // We'll never get here!
    Console.WriteLine ("Exception!");
  }
}

static void Go() { throw null; }   // Throws a NullReferenceException
```

Exceptions



```
public static void Main()
{
   new Thread (Go).Start();
}

static void Go()
{
  try
  {
    // ...
    throw null;    // The NullReferenceException will get caught below
    // ...
  }
  catch (Exception ex)
  {
    // Typically log the exception, and/or signal another thread
    // that we've come unstuck
    // ...
  }
}
```

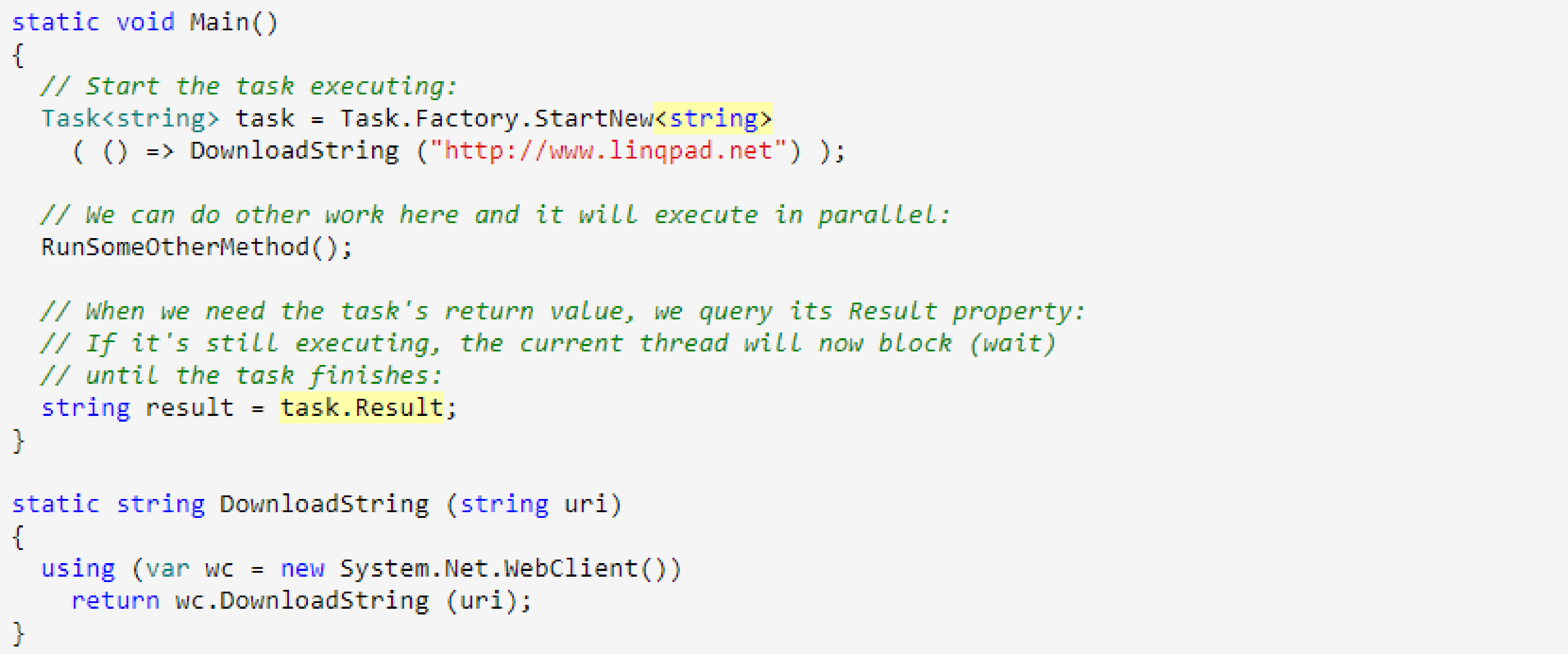
Thread Pooling - Why?

- Whenever you start a thread, a few hundred microseconds are spent
- keeps a lid on the total number of worker threads

Thread Pooling - Ways

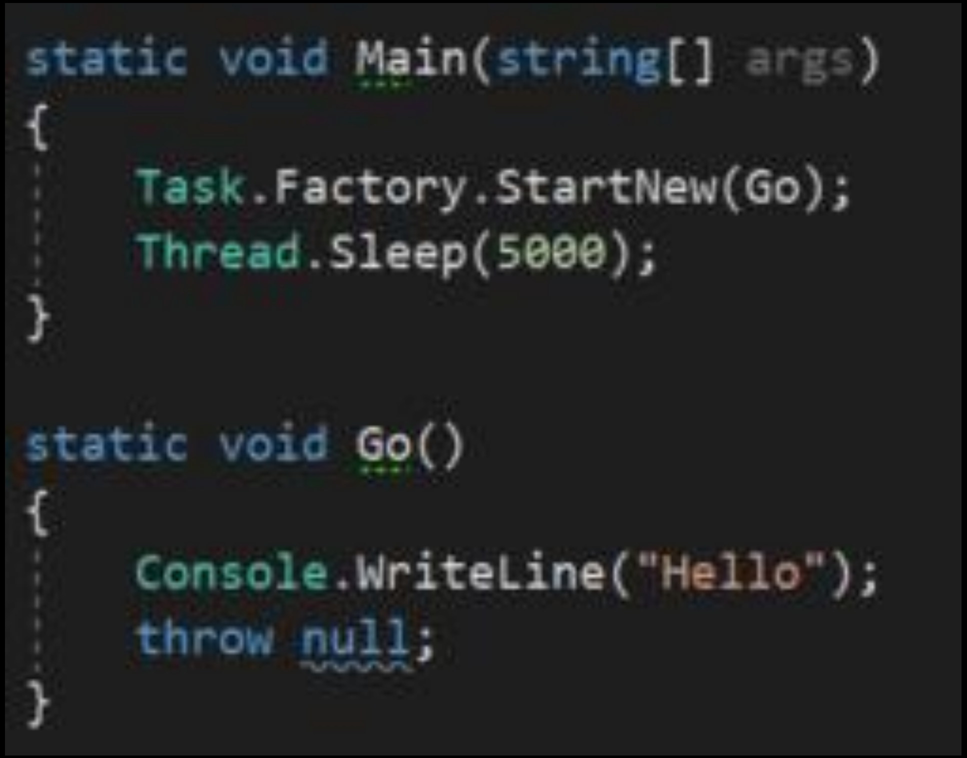
- Via the Task Parallel Library (from Framework 4.0)
- By calling ThreadPool.QueueUserWorkItem
- Via asynchronous delegates

Thread Pooling - TPL

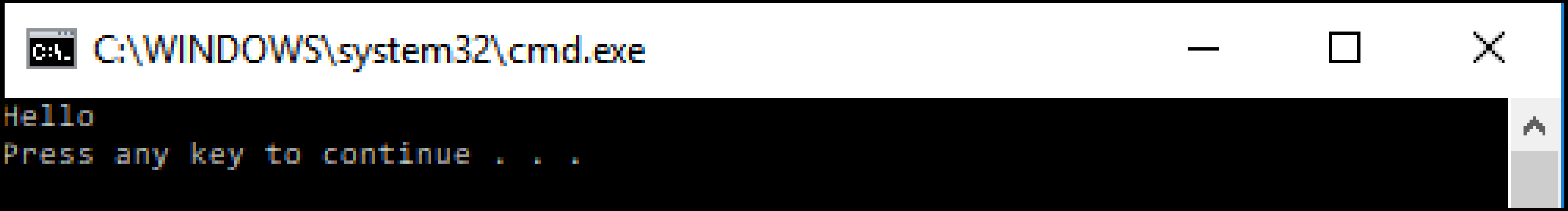


```
static void Main()
{
  // Start the task executing:
  Task<string> task = Task.Factory.StartNew<string>
    ( () => DownloadString ("http://www.linqpad.net") );

  // We can do other work here and it will execute in parallel:
  RunSomeOtherMethod();

  // When we need the task's return value, we query its Result property:
  // If it's still executing, the current thread will now block (wait)
  // until the task finishes:
  string result = task.Result;
}

static string DownloadString (string uri)
{
  using (var wc = new System.Net.WebClient())
    return wc.DownloadString (uri);
}
```

Thread Pooling - Exceptions

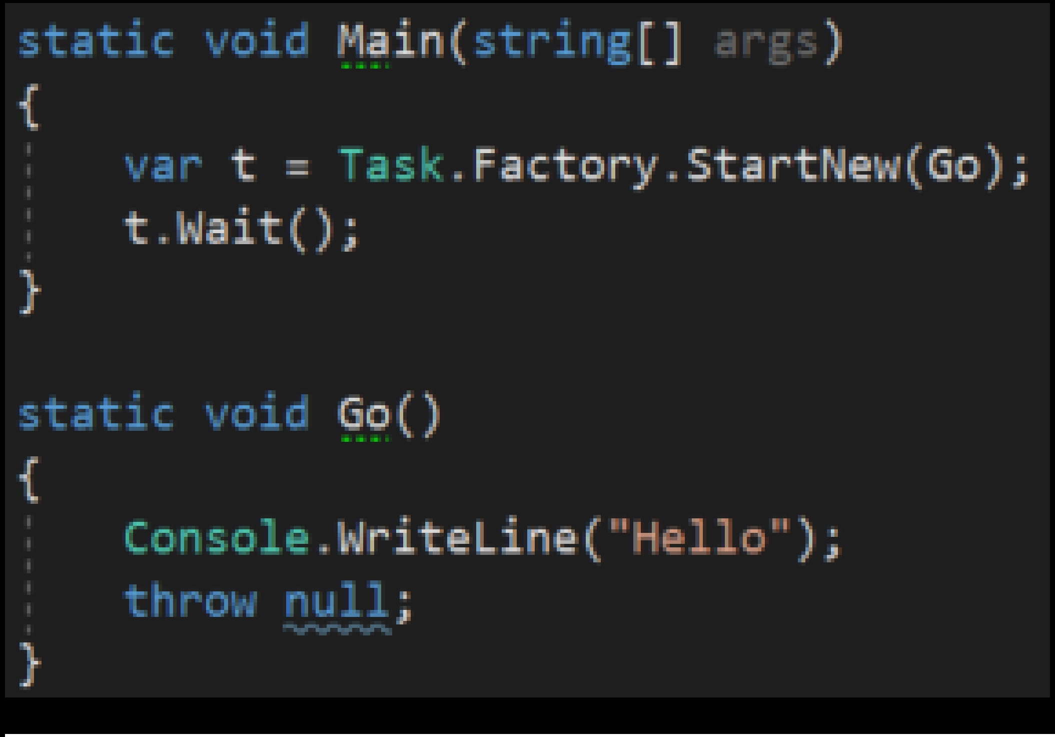


```
static void Main(string[] args)
{
    Task.Factory.StartNew(Go);
    Thread.Sleep(5000);
}

static void Go()
{
    Console.WriteLine("Hello");
    throw null;
}
```

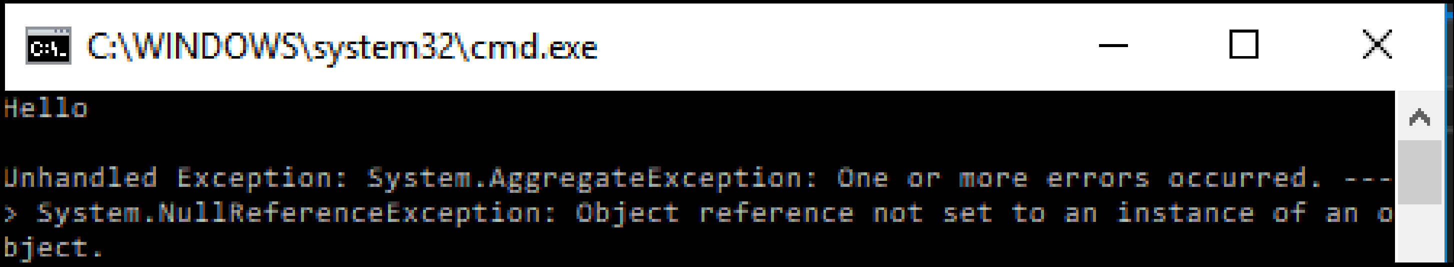


```
Hello
Press any key to continue . . .
```

Thread Pooling - Exceptions

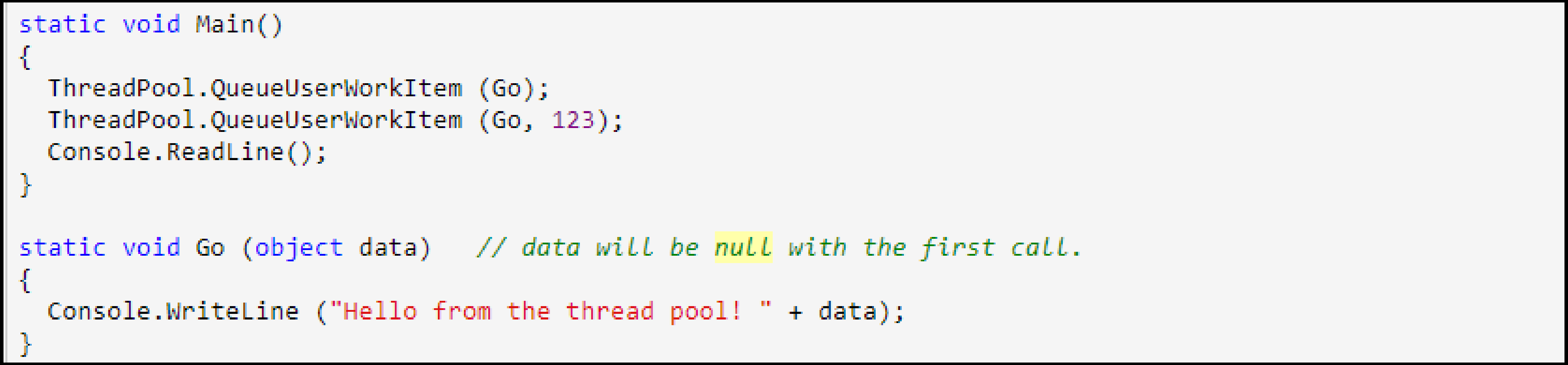


```
static void Main(string[] args)
{
    var t = Task.Factory.StartNew(Go);
    t.Wait();
}

static void Go()
{
    Console.WriteLine("Hello");
    throw null;
}
```



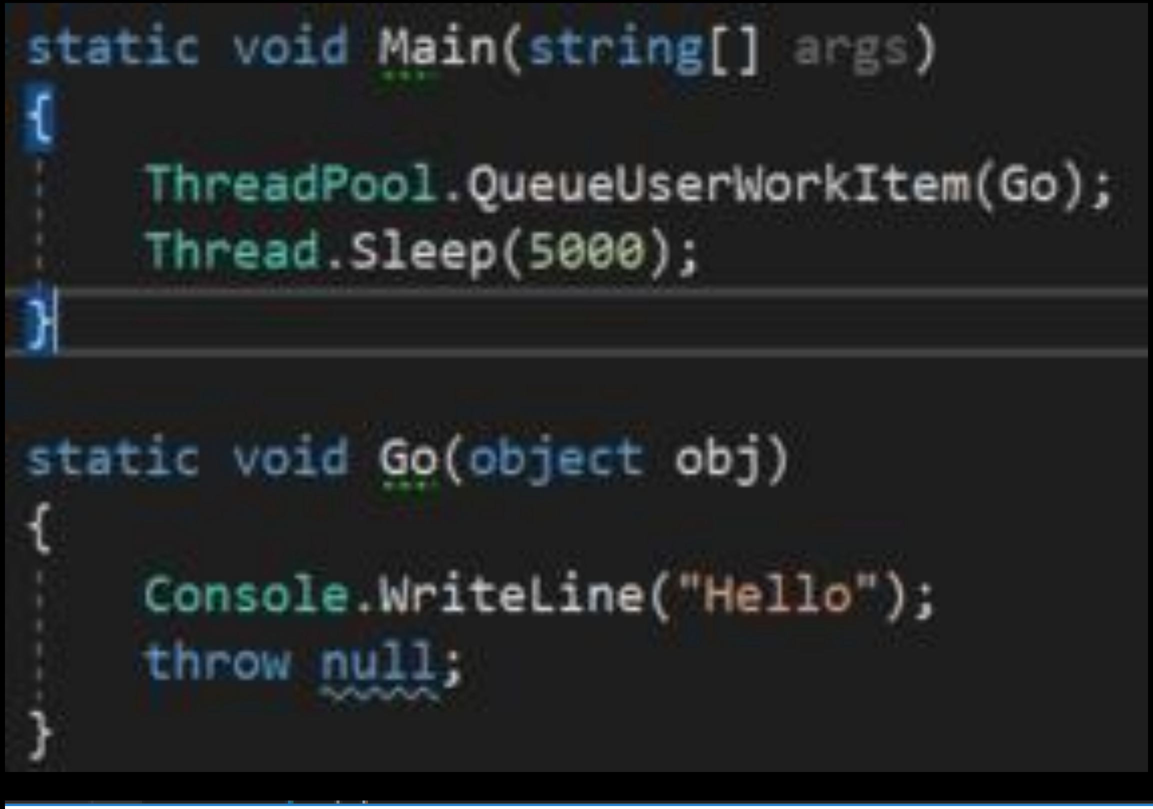
```
Hello

Unhandled Exception: System.AggregateException: One or more errors occurred. ---
> System.NullReferenceException: Object reference not set to an instance of an o
bject.
```

Thread Pooling - QueueUserWorkItem

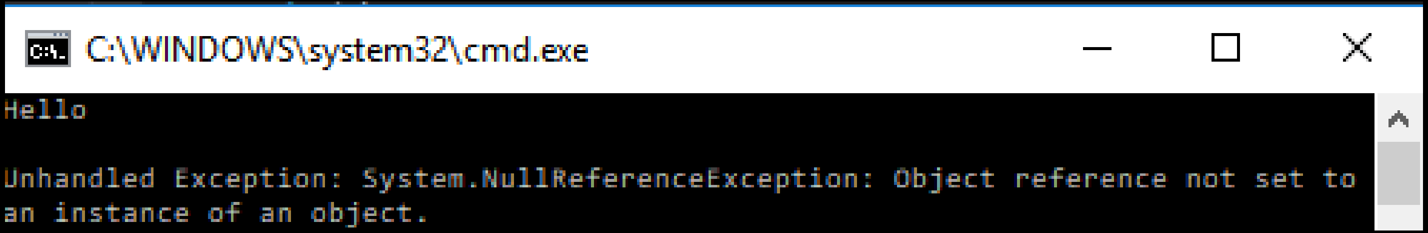


```
static void Main()
{
  ThreadPool.QueueUserWorkItem (Go);
  ThreadPool.QueueUserWorkItem (Go, 123);
  Console.ReadLine();
}

static void Go (object data)   // data will be null with the first call.
{
  Console.WriteLine ("Hello from the thread pool! " + data);
}
```

Thread Pooling - QueueUserWorkItem

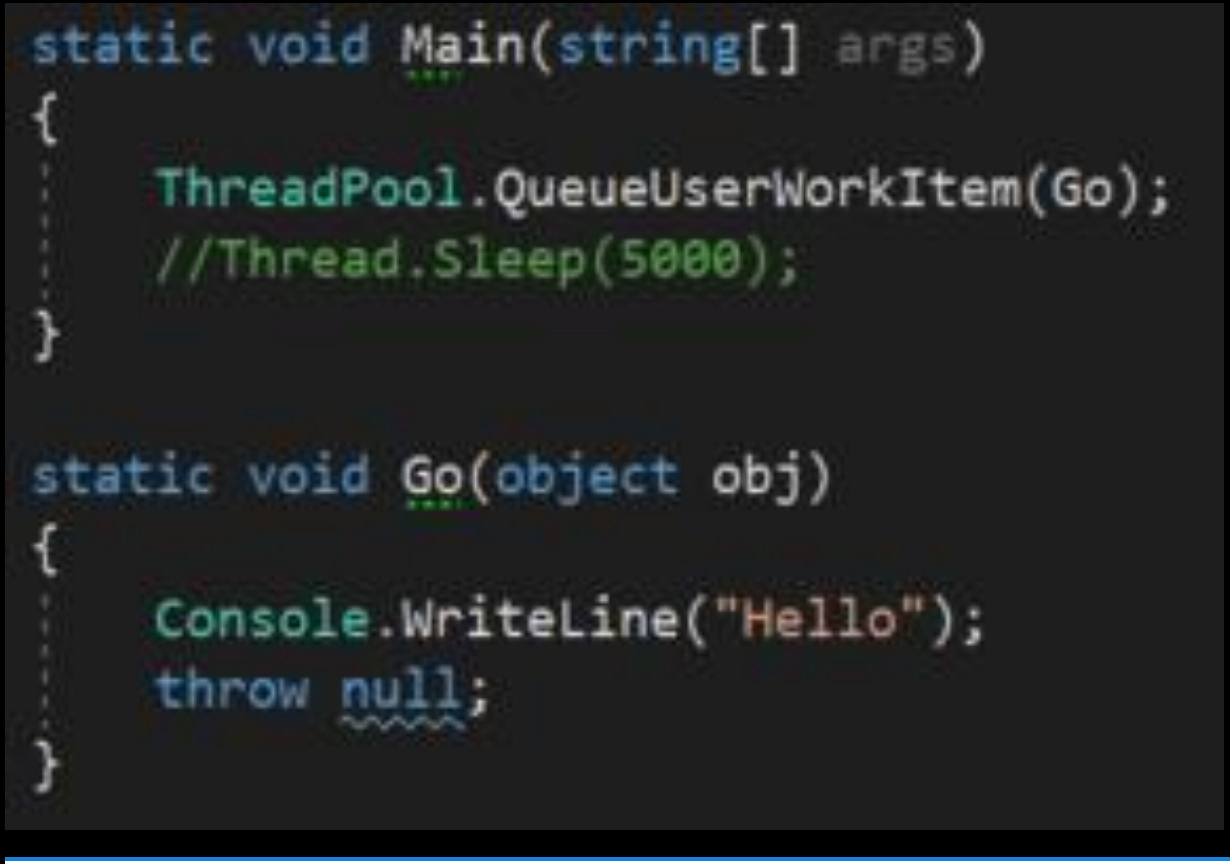


```
static void Main(string[] args)
{
    ThreadPool.QueueUserWorkItem(Go);
    Thread.Sleep(5000);
}

static void Go(object obj)
{
    Console.WriteLine("Hello");
    throw null;
}
```

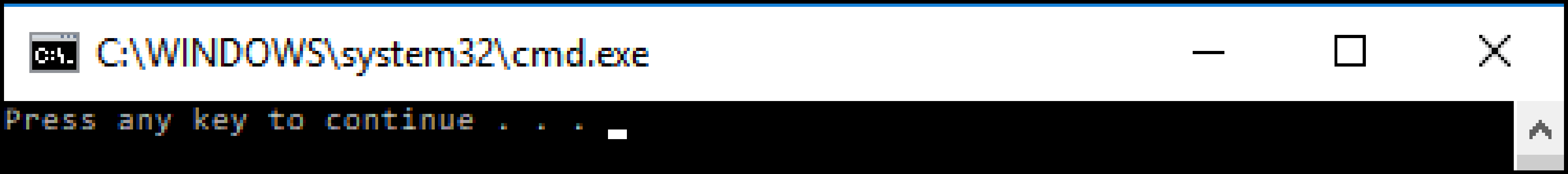


```
Hello

Unhandled Exception: System.NullReferenceException: Object reference not set to
an instance of an object.
```

Thread Pooling - QueueUserWorkItem

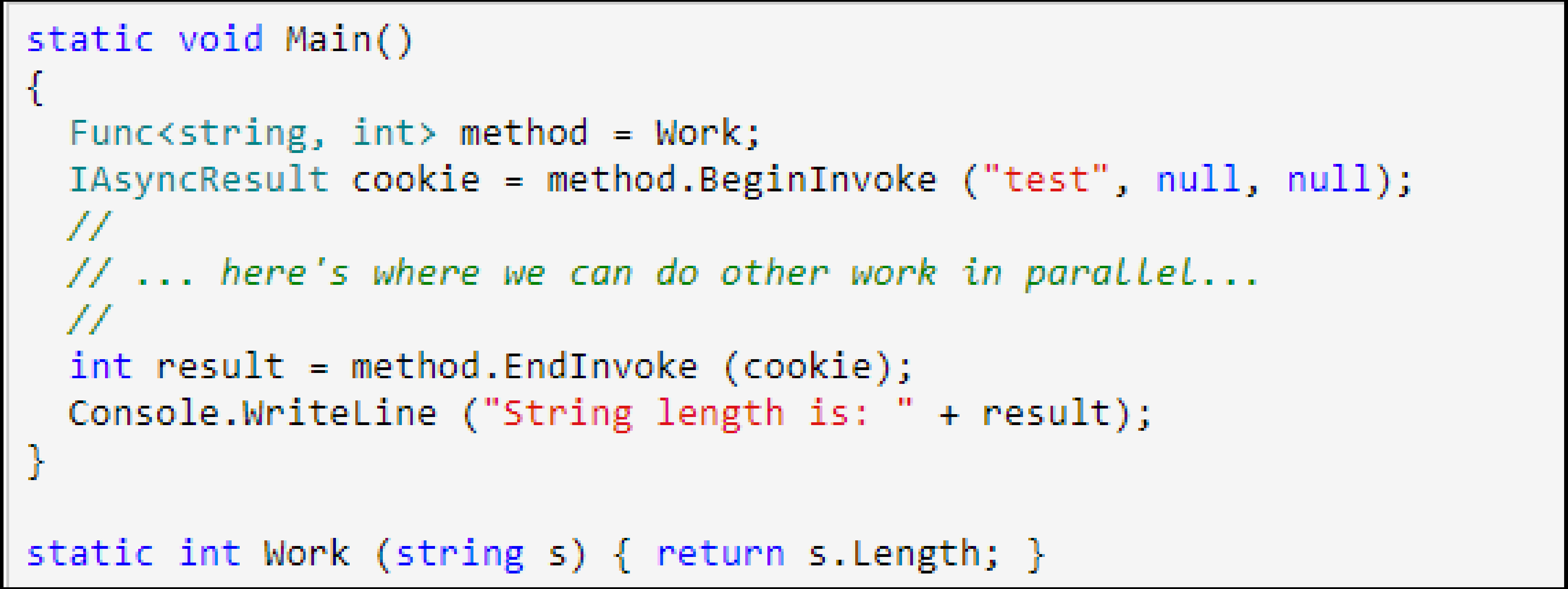


```
static void Main(string[] args)
{
    ThreadPool.QueueUserWorkItem(Go);
    //Thread.Sleep(5000);
}

static void Go(object obj)
{
    Console.WriteLine("Hello");
    throw null;
}
```



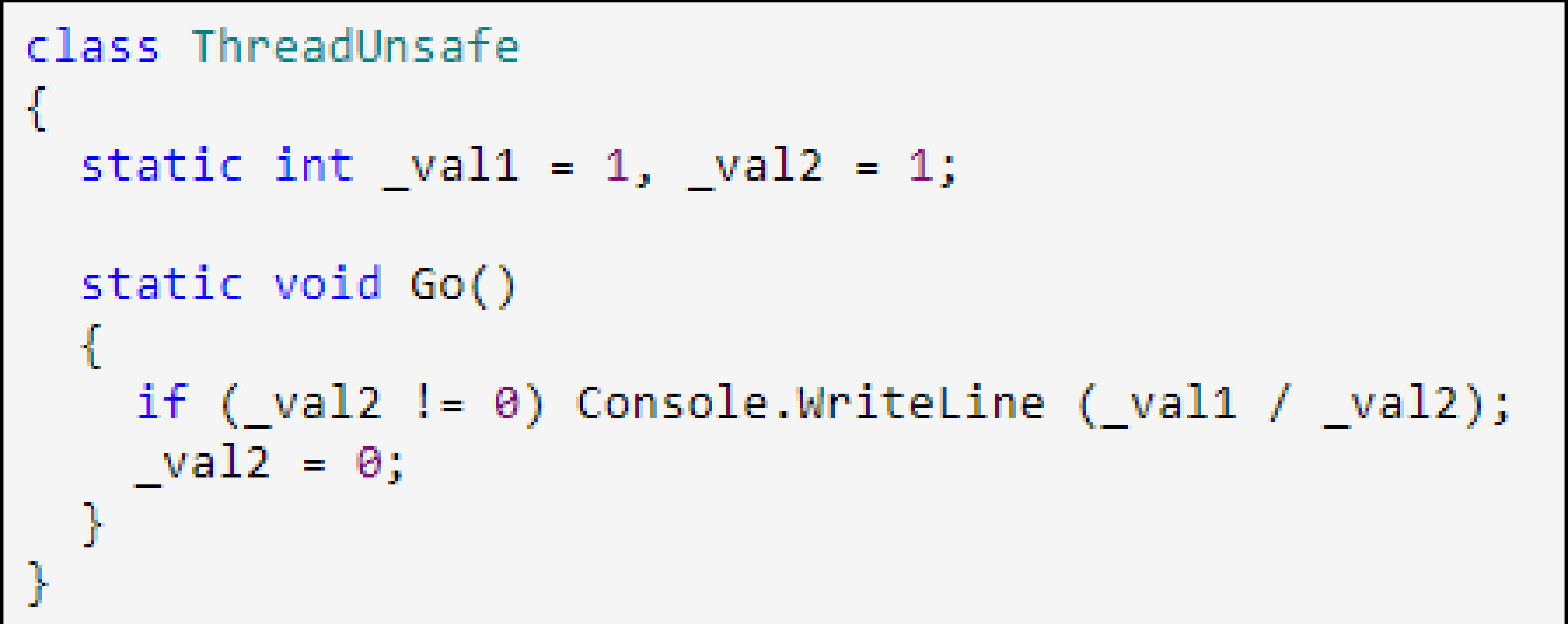
```
Press any key to continue . . .
```

Thread Pooling – Asynchronous delegate



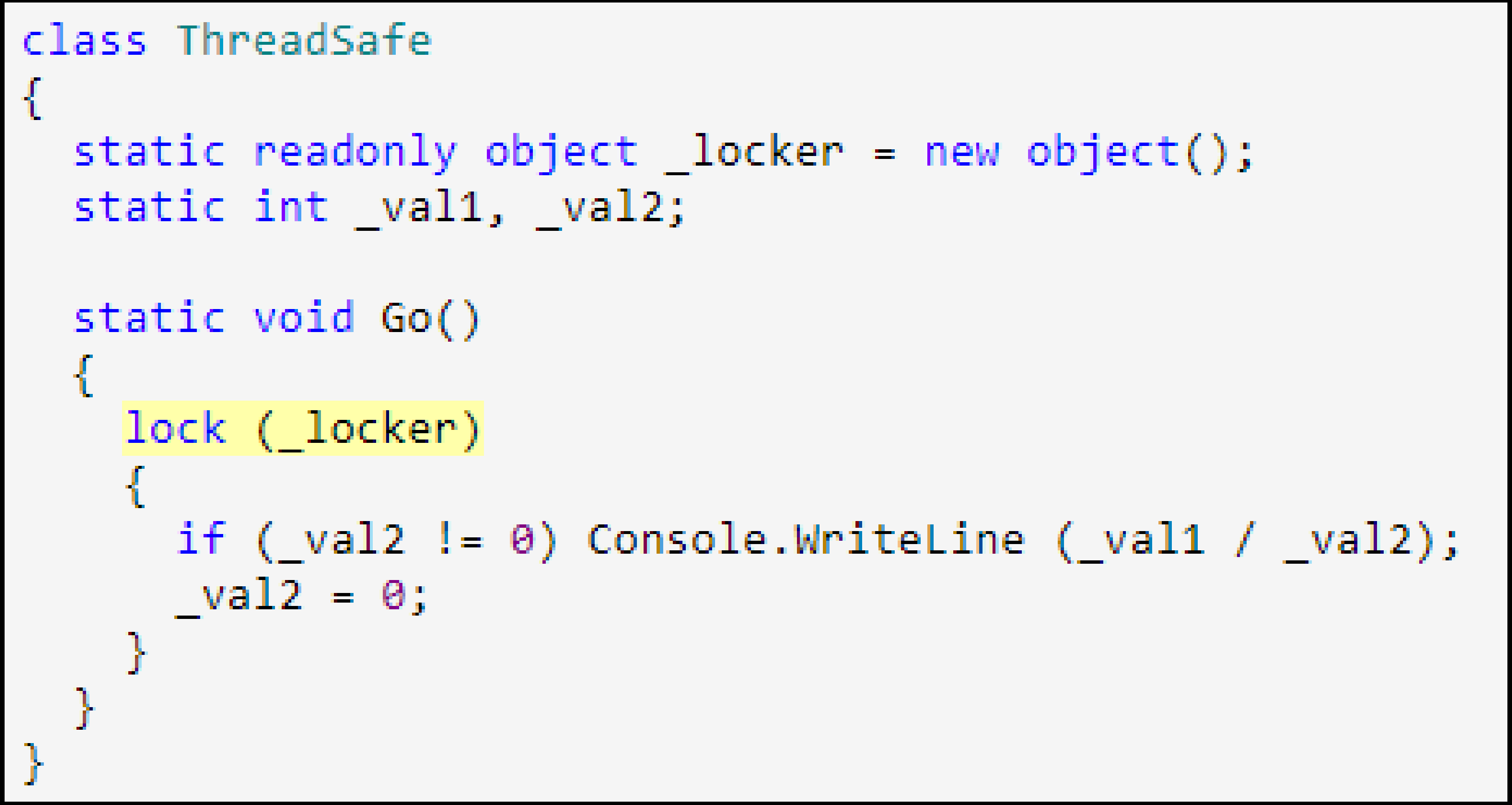
```
static void Main()
{
  Func<string, int> method = Work;
  IAsyncResult cookie = method.BeginInvoke ("test", null, null);
  //
  // ... here's where we can do other work in parallel...
  //
  int result = method.EndInvoke (cookie);
  Console.WriteLine ("String length is: " + result);
}

static int Work (string s) { return s.Length; }
```

Locking



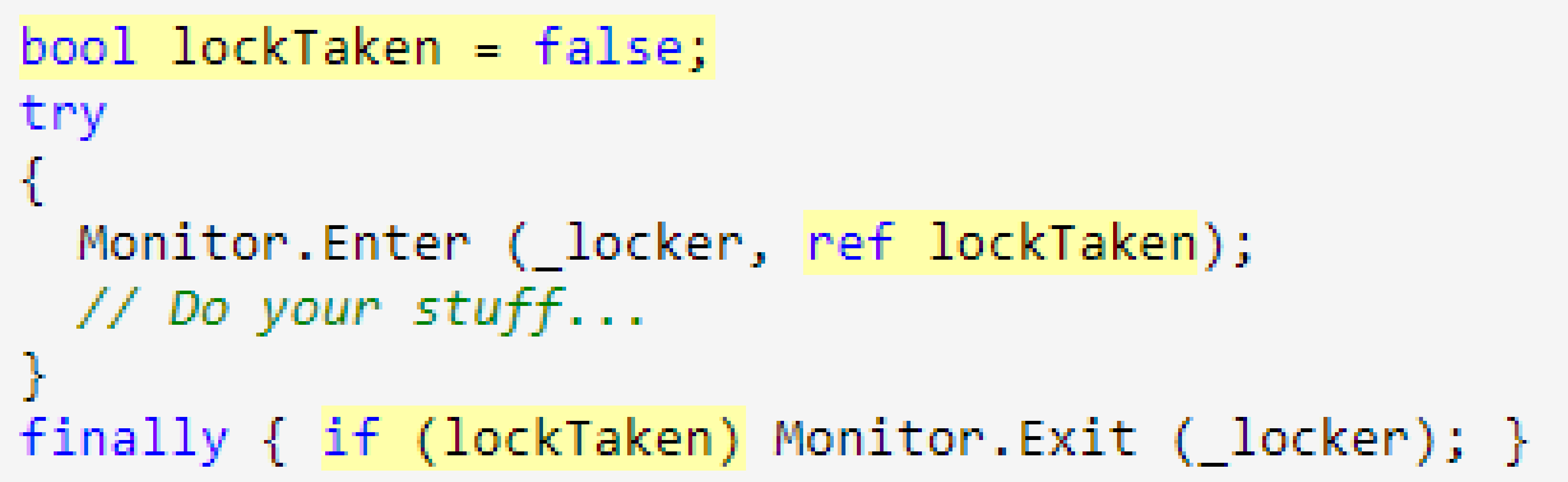
```
class ThreadUnsafe
{
  static int _val1 = 1, _val2 = 1;

  static void Go()
  {
    if (_val2 != 0) Console.WriteLine (_val1 / _val2);
    _val2 = 0;
  }
}
```

Locking



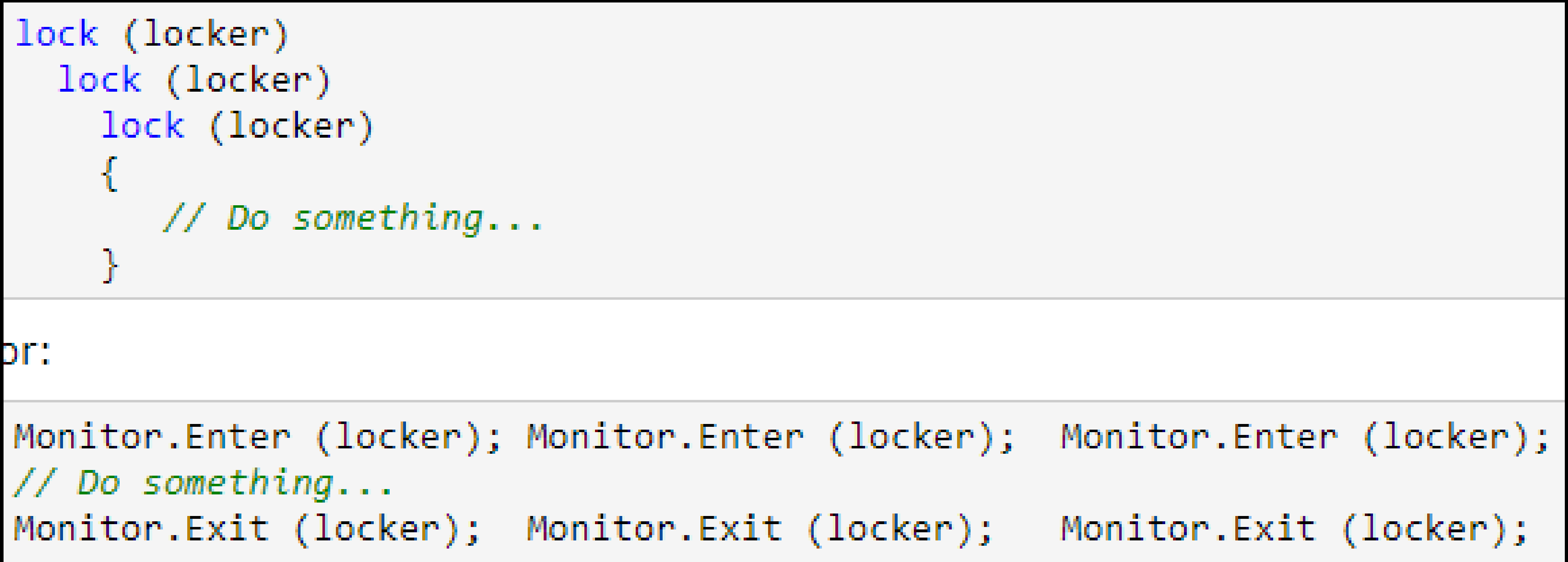
```
class ThreadSafe
{
  static readonly object _locker = new object();
  static int _val1, _val2;

  static void Go()
  {
    lock (_locker)
    {
      if (_val2 != 0) Console.WriteLine (_val1 / _val2);
      _val2 = 0;
    }
  }
}
```

Locking



```
bool lockTaken = false;
try
{
  Monitor.Enter (_locker, ref lockTaken);
  // Do your stuff...
}
finally { if (lockTaken) Monitor.Exit (_locker); }
```

Locking

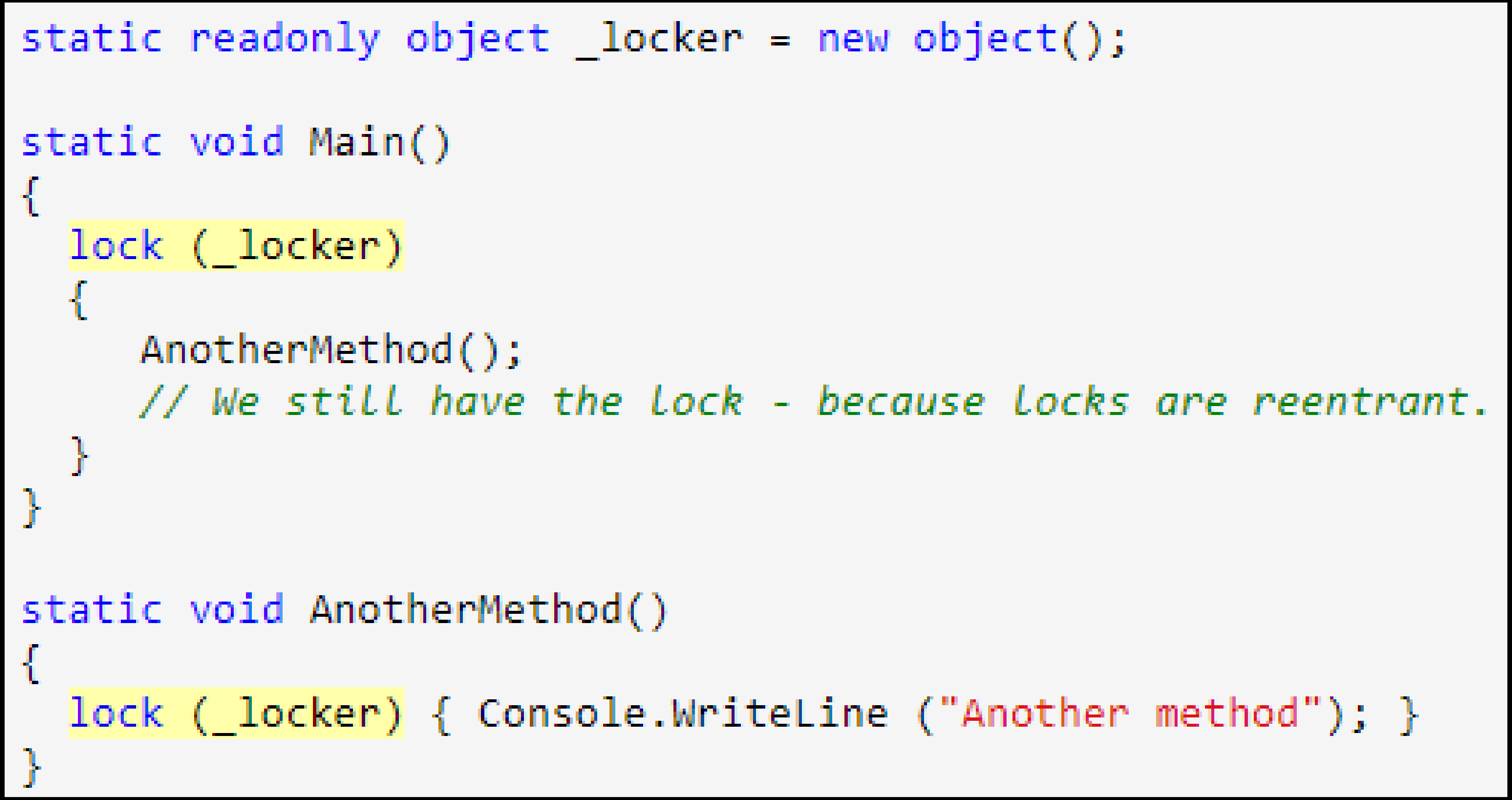


```
lock (locker)
  lock (locker)
    lock (locker)
    {
       // Do something...
    }
```

or:

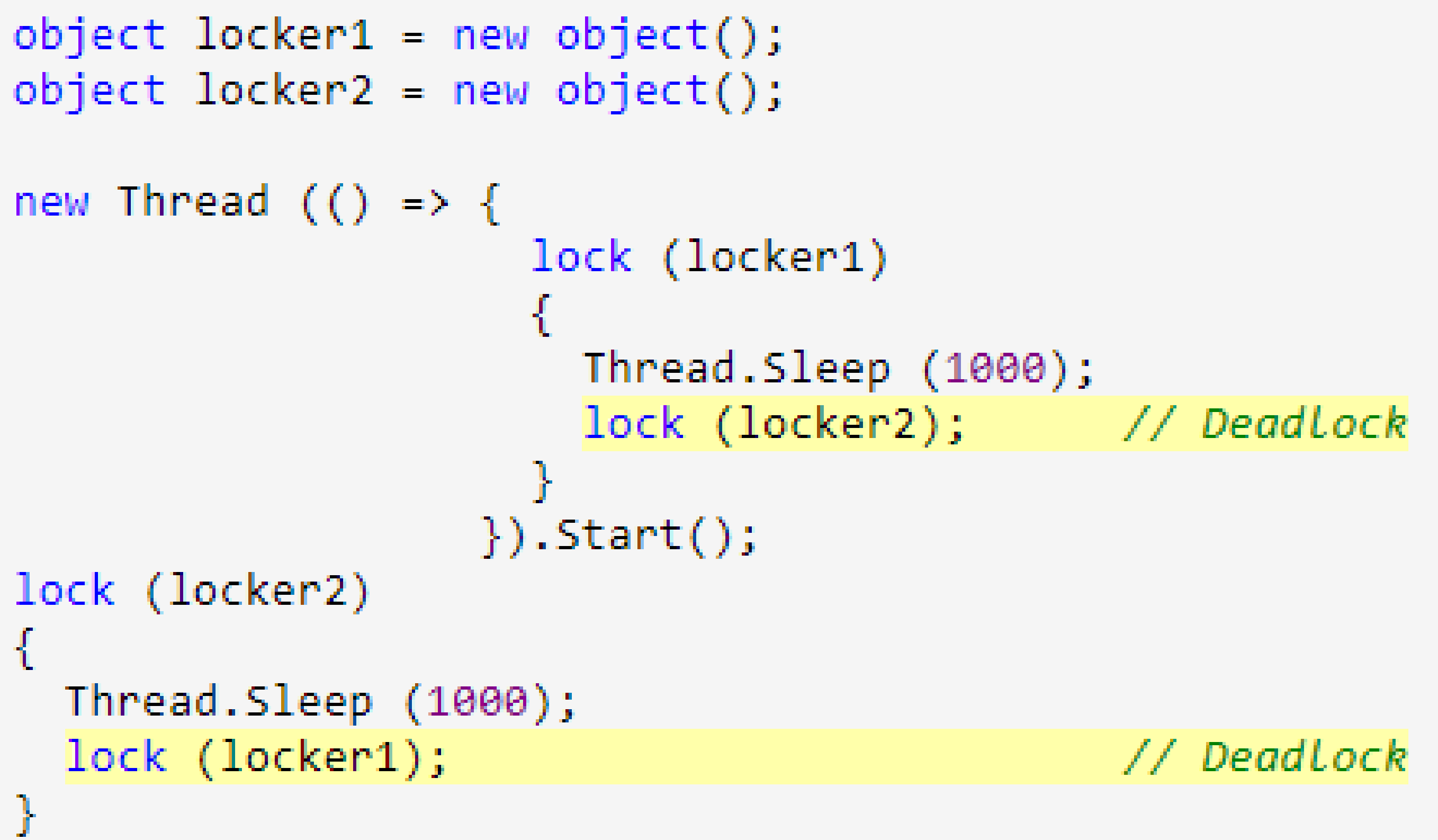
```
Monitor.Enter (locker); Monitor.Enter (locker);  Monitor.Enter (locker);
// Do something...
Monitor.Exit (locker);  Monitor.Exit (locker);   Monitor.Exit (locker);
```

Locking



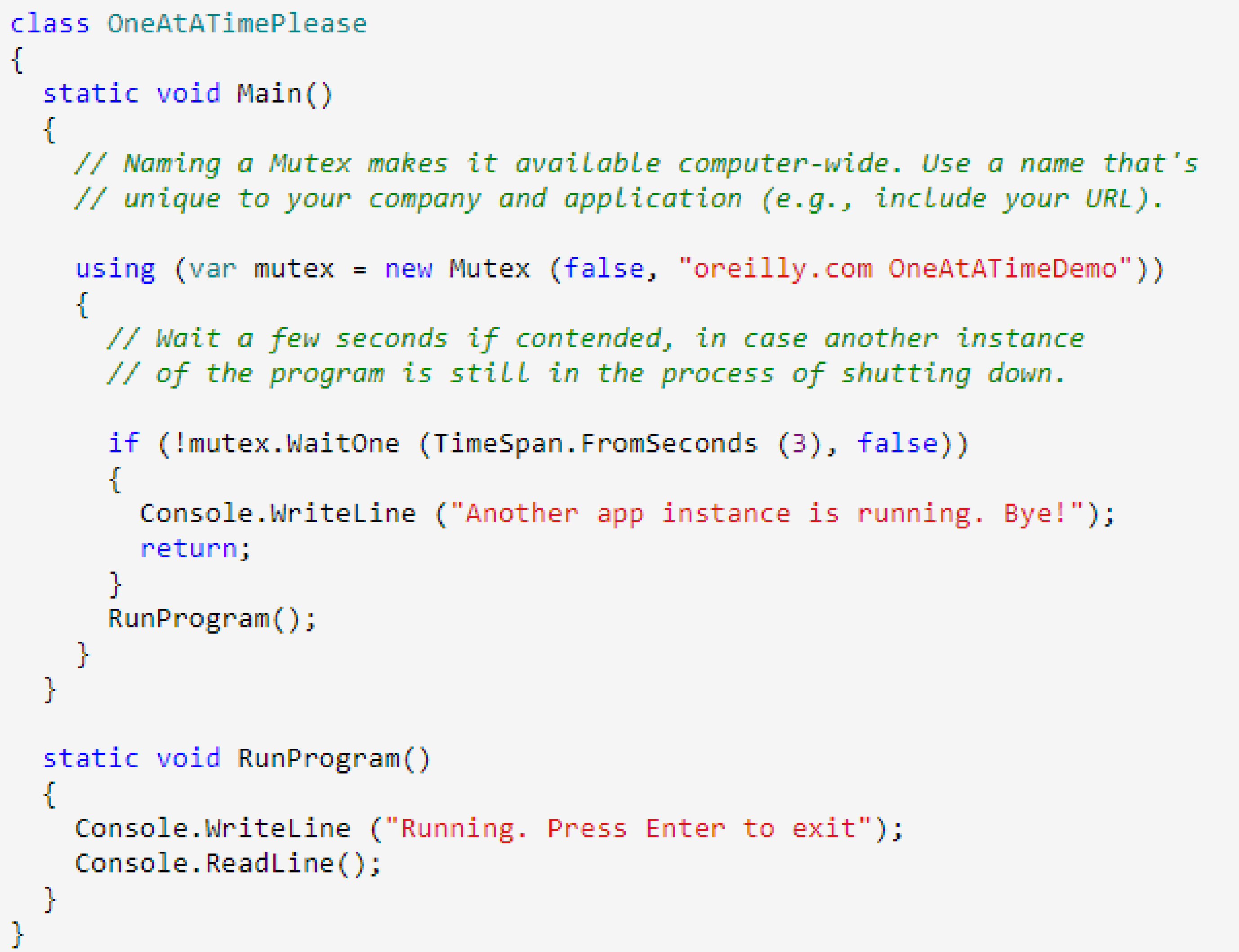
```
static readonly object _locker = new object();

static void Main()
{
  lock (_locker)
  {
     AnotherMethod();
     // We still have the lock - because locks are reentrant.
  }
}

static void AnotherMethod()
{
  lock (_locker) { Console.WriteLine ("Another method"); }
}
```

Deadlock



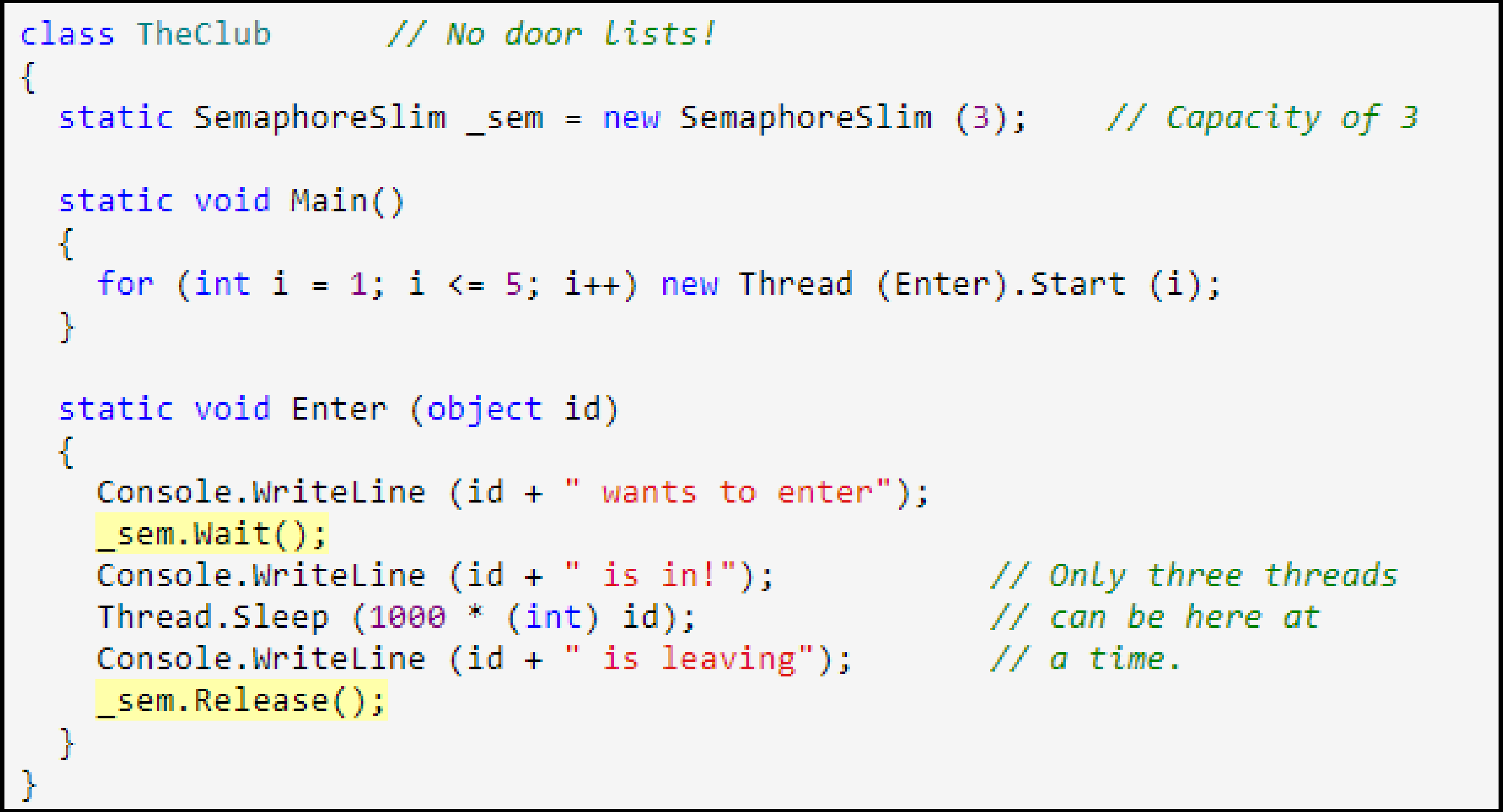
```
object locker1 = new object();
object locker2 = new object();

new Thread (() => {
                    lock (locker1)
                    {
                      Thread.Sleep (1000);
                      lock (locker2);      // Deadlock
                    }
                  }).Start();
lock (locker2)
{
  Thread.Sleep (1000);
  lock (locker1);                          // Deadlock
}
```

Mutex



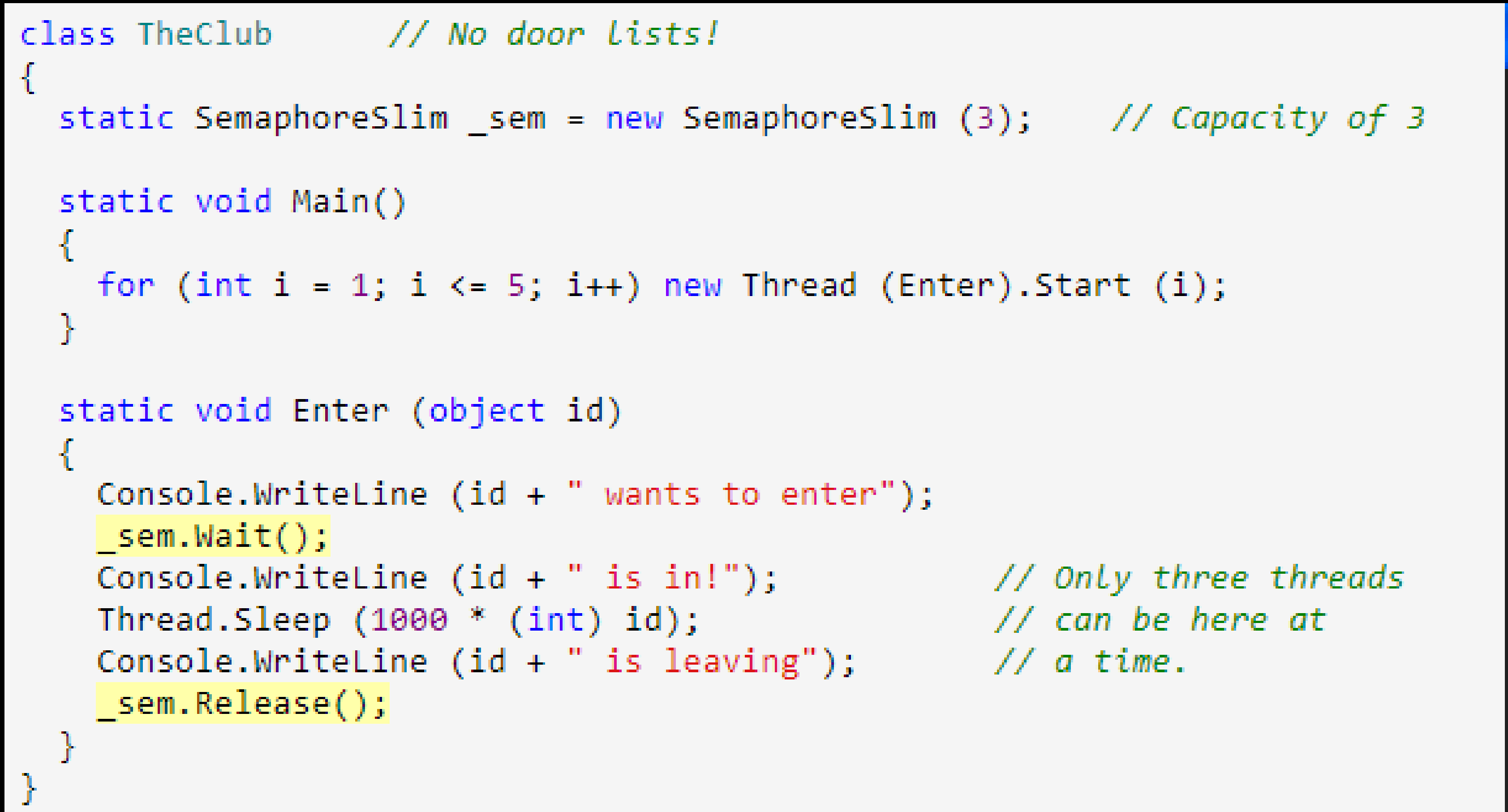
```
class OneAtATimePlease
{
  static void Main()
  {
    // Naming a Mutex makes it available computer-wide. Use a name that's
    // unique to your company and application (e.g., include your URL).

    using (var mutex = new Mutex (false, "oreilly.com OneAtATimeDemo"))
    {
      // Wait a few seconds if contended, in case another instance
      // of the program is still in the process of shutting down.

      if (!mutex.WaitOne (TimeSpan.FromSeconds (3), false))
      {
        Console.WriteLine ("Another app instance is running. Bye!");
        return;
      }
      RunProgram();
    }
  }

  static void RunProgram()
  {
    Console.WriteLine ("Running. Press Enter to exit");
    Console.ReadLine();
  }
}
```

Semaphore

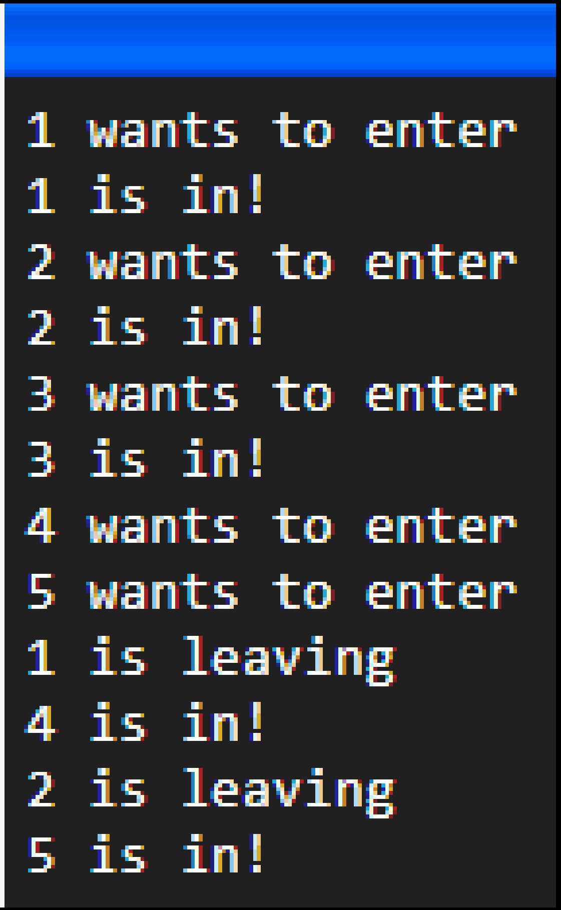


```
class TheClub      // No door lists!
{
  static SemaphoreSlim _sem = new SemaphoreSlim (3);    // Capacity of 3

  static void Main()
  {
    for (int i = 1; i <= 5; i++) new Thread (Enter).Start (i);
  }

  static void Enter (object id)
  {
    Console.WriteLine (id + " wants to enter");
    _sem.Wait();
    Console.WriteLine (id + " is in!");           // Only three threads
    Thread.Sleep (1000 * (int) id);               // can be here at
    Console.WriteLine (id + " is leaving");       // a time.
    _sem.Release();
  }
}
```

Semaphore

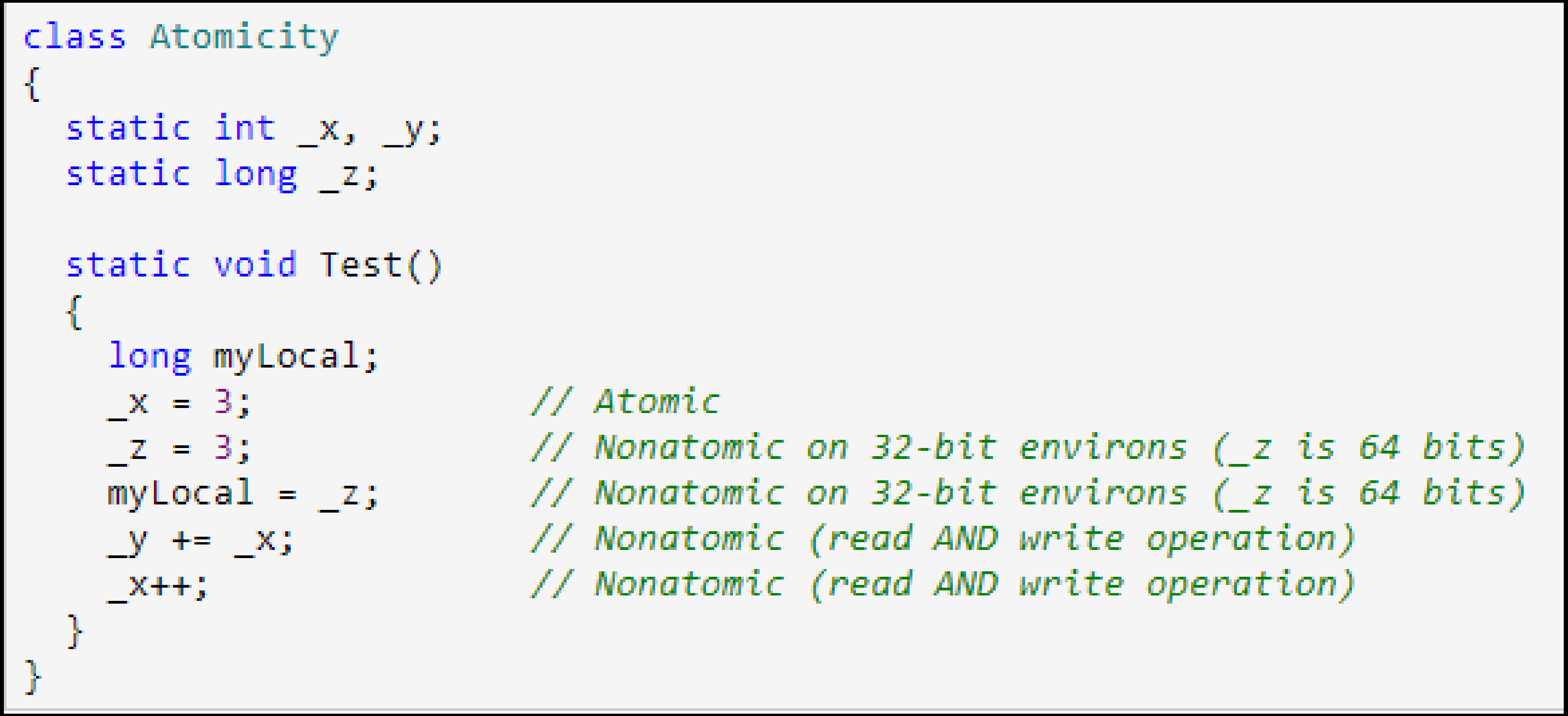


```
class TheClub      // No door lists!
{
  static SemaphoreSlim _sem = new SemaphoreSlim (3);    // Capacity of 3

  static void Main()
  {
    for (int i = 1; i <= 5; i++) new Thread (Enter).Start (i);
  }

  static void Enter (object id)
  {
    Console.WriteLine (id + " wants to enter");
    _sem.Wait();
    Console.WriteLine (id + " is in!");           // Only three threads
    Thread.Sleep (1000 * (int) id);               // can be here at
    Console.WriteLine (id + " is leaving");       // a time.
    _sem.Release();
  }
}
```



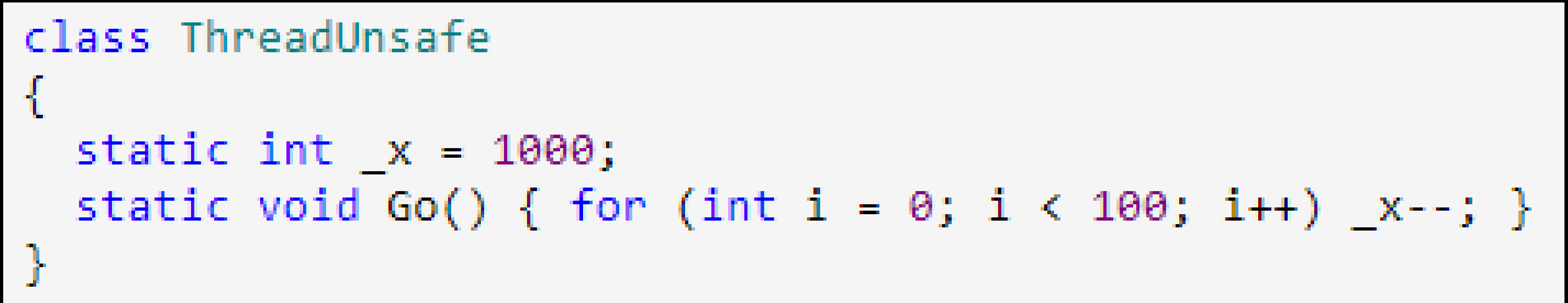
```
1 wants to enter
1 is in!
2 wants to enter
2 is in!
3 wants to enter
3 is in!
4 wants to enter
5 wants to enter
1 is leaving
4 is in!
2 is leaving
5 is in!
```

Interlocked



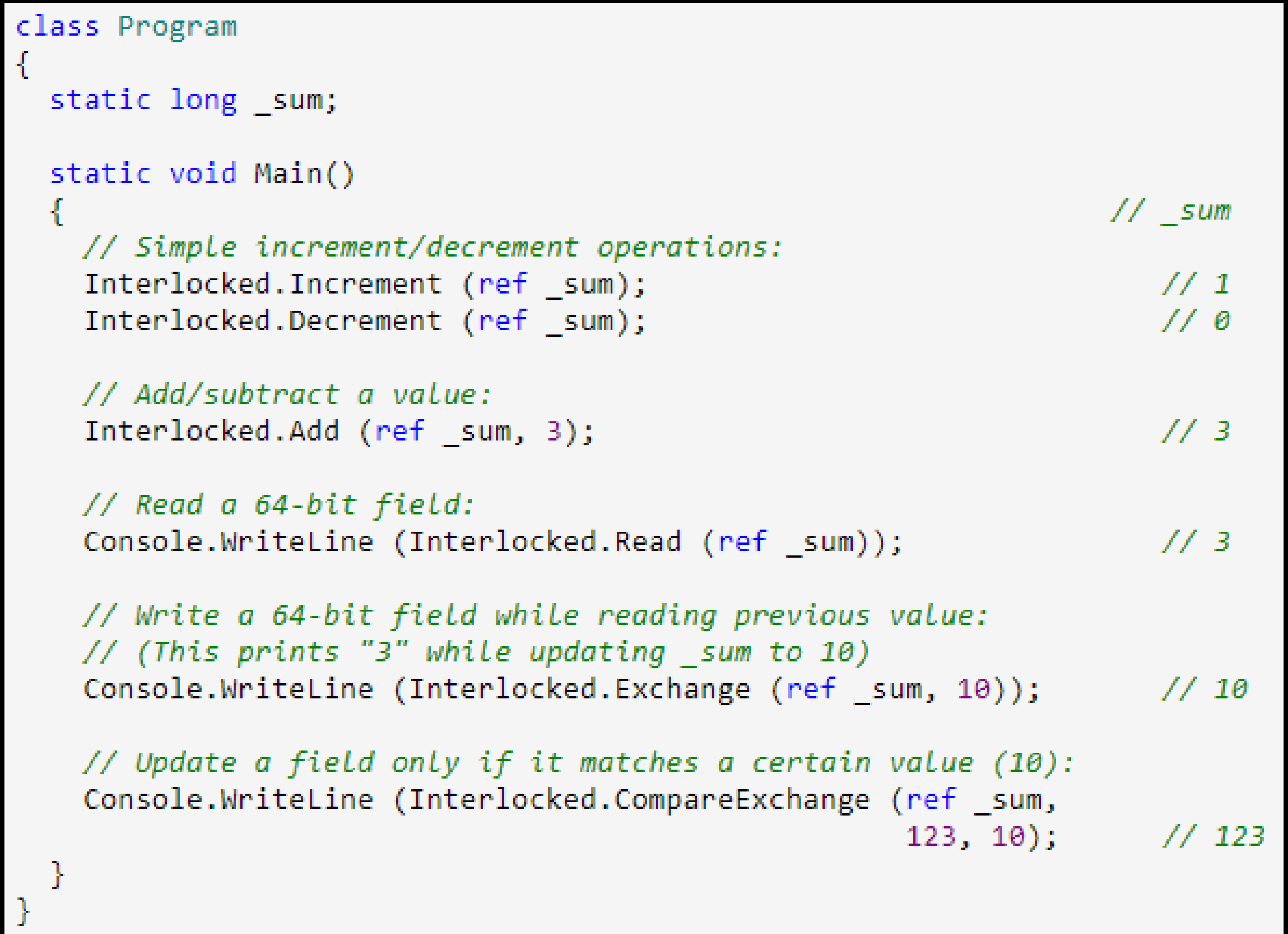
```
class Atomicity
{
  static int _x, _y;
  static long _z;

  static void Test()
  {
    long myLocal;
    _x = 3;             // Atomic
    _z = 3;             // Nonatomic on 32-bit environs (_z is 64 bits)
    myLocal = _z;       // Nonatomic on 32-bit environs (_z is 64 bits)
    _y += _x;           // Nonatomic (read AND write operation)
    _x++;               // Nonatomic (read AND write operation)
  }
}
```

Interlocked



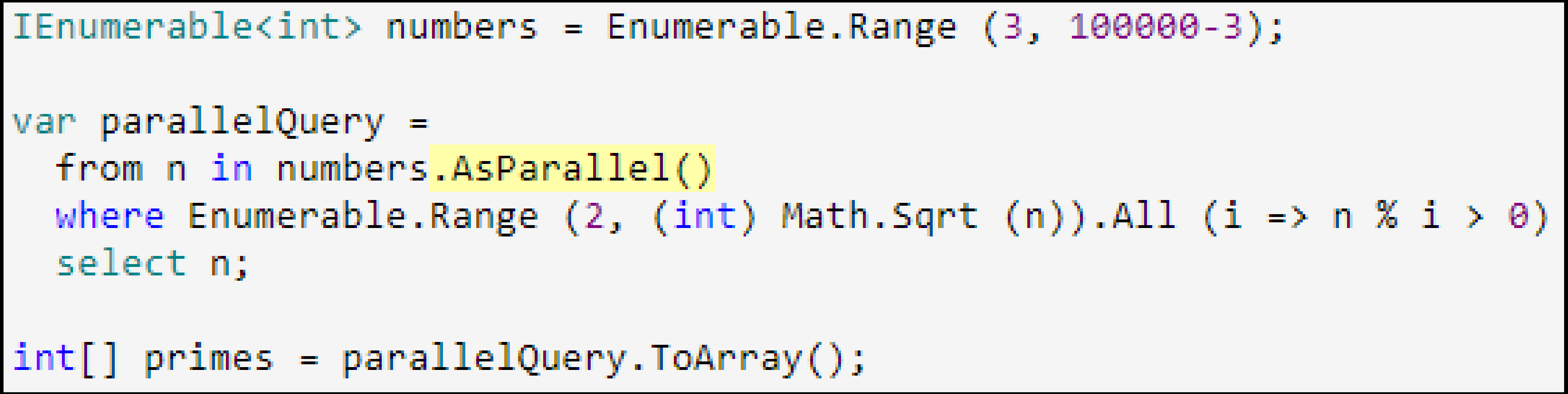
```
class ThreadUnsafe
{
  static int _x = 1000;
  static void Go() { for (int i = 0; i < 100; i++) _x--; }
}
```

Interlocked



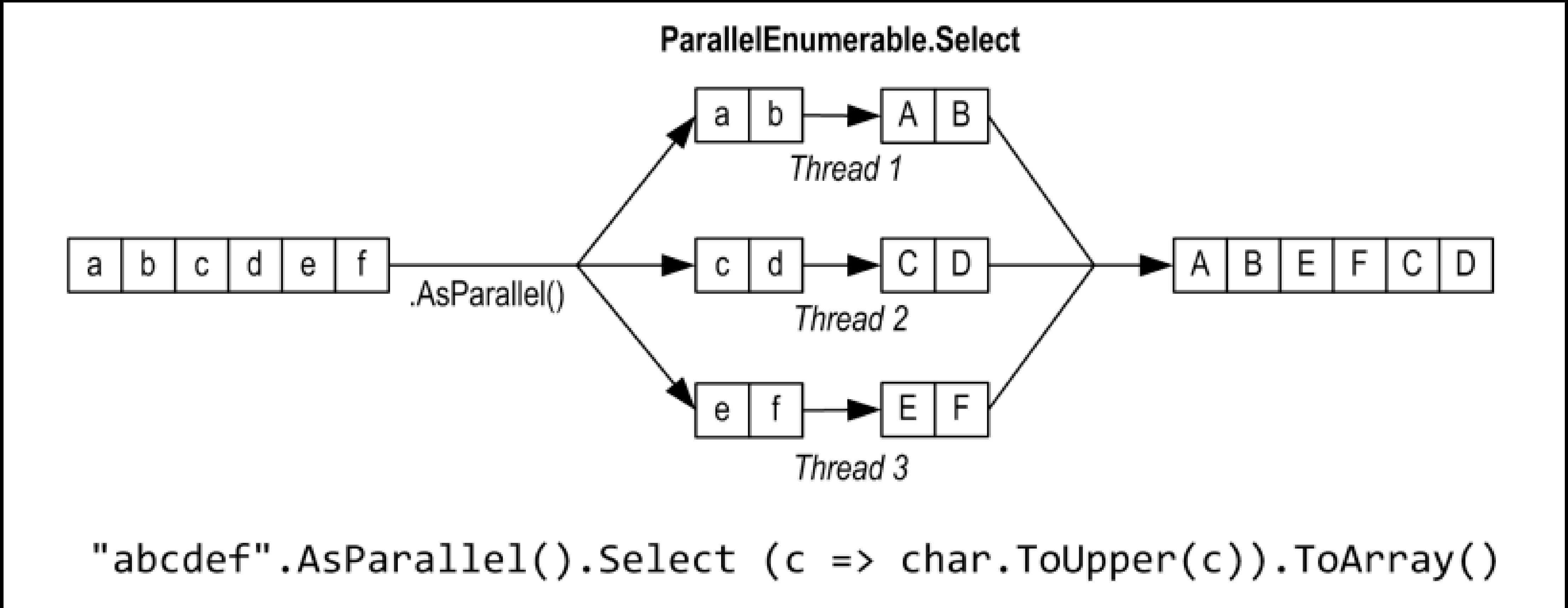
```
class Program
{
  static long _sum;

  static void Main()
  {                                                             // _sum
    // Simple increment/decrement operations:
    Interlocked.Increment (ref _sum);                              // 1
    Interlocked.Decrement (ref _sum);                              // 0

    // Add/subtract a value:
    Interlocked.Add (ref _sum, 3);                                 // 3

    // Read a 64-bit field:
    Console.WriteLine (Interlocked.Read (ref _sum));               // 3

    // Write a 64-bit field while reading previous value:
    // (This prints "3" while updating _sum to 10)
    Console.WriteLine (Interlocked.Exchange (ref _sum, 10));       // 10

    // Update a field only if it matches a certain value (10):
    Console.WriteLine (Interlocked.CompareExchange (ref _sum,
                                                    123, 10);      // 123
  }
}
```

PLINQ



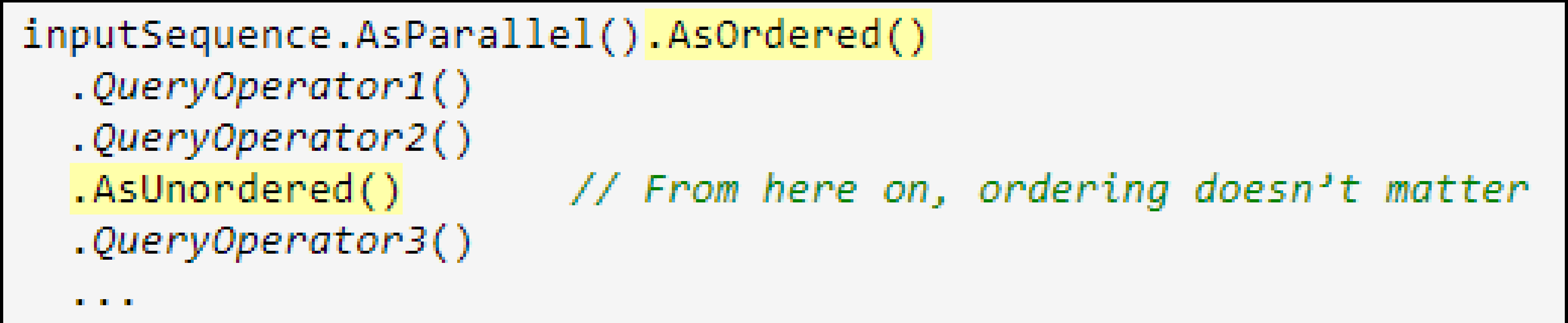
```
IEnumerable<int> numbers = Enumerable.Range (3, 100000-3);

var parallelQuery =
  from n in numbers.AsParallel()
  where Enumerable.Range (2, (int) Math.Sqrt (n)).All (i => n % i > 0)
  select n;

int[] primes = parallelQuery.ToArray();
```

PLINQ



```
"abcdef".AsParallel().Select (c => char.ToUpper(c)).ToArray()
```

PLINQ



```
inputSequence.AsParallel().AsOrdered()
  .QueryOperator1()
  .QueryOperator2()
  .AsUnordered()       // From here on, ordering doesn’t matter
  .QueryOperator3()
  ...
```