Mohammad Shaker
mohammadshaker.com
@ZGTRShaker
2011, 2012, 2013, 2014

C# ADVANCED

L04-THREADING

Threading

Concept of Threading

- What is a “Thread”?
  - The advantage of threading is the ability to create applications that use more than one thread of execution. For example, a process can have a user interface thread that manages interactions with the user and worker threads that perform other tasks while the user interface thread waits for user input.

- What’s this?

```
for (int i = 1; i < 50; i++)
{
    Console.SetWindowSize(i, i);
    System.Threading.Thread.Sleep(50);
}
```

Let’s `Parallel`

Let's Parallel

```
class Program
{
    private static void Main()
    {
        for (int i = 0; i < 10; i++)
        {
            long total = GetTotal();
            Console.WriteLine("{0} - {1}", i, total);
        }
    }

    private static long GetTotal()
    {
        long total = 0;
        for (int i = 1; i < 100000000; i++)
        {
              total += i;
        }
        return total;
    }
}
```

Let's Parallel

```
class Program
{
    private static void Main()
    {
        for (int i = 0; i < 10; i++)
        {
            long total = GetTotal();
            Console.WriteLine("{0} - {1}", i, total);
        }
    }

    private static long GetTotal()
    {
        long total = 0;
        for (int i = 1; i < 100000000; i++)
        {
                total += i;
        }
        return total;
    }
}
```

```
0 - 4999999950000000
1 - 4999999950000000
2 - 4999999950000000
3 - 4999999950000000
4 - 4999999950000000
5 - 4999999950000000
6 - 4999999950000000
7 - 4999999950000000
8 - 4999999950000000
9 - 4999999950000000
Press any key to continue . . .
```

Let's Parallel

```
class Program
{
    private static void Main()
    {
        for (int i = 0; i < 10; i++)
        {
            long total = GetTotal();
            Console.WriteLine("{0} - {1}", i, total);
        }
    }

    private static long GetTotal()
    {
        long total = 0;
        for (int i = 1; i < 100000000; i++)
        {
              total += i;
        }
        return total;
    }
}
```

Let's Parallel

```
class Program
{
    private static void Main()
    {
          Parallel.For(0, 10, i =>
          {
              long total = GetTotal();
              Console.WriteLine("{0} - {1}", i, total);
          });
    }

    private static long GetTotal()
    {
        long total = 0;
        for (int i = 1; i < 100000000; i++)
        {
               total += i;
        }
        return total;
    }
}
```

Let's Parallel

```
class Program
{
    private static void Main()
    {
          Parallel.For(0, 10, i =>
          {
              long total = GetTotal();
              Console.WriteLine("{0} - {1}", i, total);
          });
    }

    private static long GetTotal()
    {
        long total = 0;
        for (int i = 1; i < 100000000; i++)
        {
              total += i;
        }
        return total;
    }
}
```

```
0 - 4999999950000000
1 - 4999999950000000
2 - 4999999950000000
3 - 4999999950000000
4 - 4999999950000000
5 - 4999999950000000
6 - 4999999950000000
7 - 4999999950000000
8 - 4999999950000000
9 - 4999999950000000
Press any key to continue . . .
```

The Same Result but Much Faster

Parallel Task

Let's Parallel

```
private void Test()
{
          Parallel.For(0, 20, i => Console.WriteLine("{0} on Task {1}", i, Task.CurrentId));
}
```

```
0 on Task 1
2 on Task 1
5 on Task 2
10 on Task 3
1 on Task 5
4 on Task 5
8 on Task 5
15 on Task 4
16 on Task 4
17 on Task 4
18 on Task 4
19 on Task 4
13 on Task 4
14 on Task 4
6 on Task 2
7 on Task 2
9 on Task 5
11 on Task 3
12 on Task 3
3 on Task 1
Press any key to continue . . .
```

5 different tasks working

Let's Parallel

```
private void Test()
{
        Parallel.For(0, 20, i => Console.WriteLine("{0} on Task {1}", i, Task.CurrentId));
}
```

```
0 on Task 1
2 on Task 1
5 on Task 2
10 on Task 3
1 on Task 5
4 on Task 5
8 on Task 5
15 on Task 4
16 on Task 4
17 on Task 4
18 on Task 4
19 on Task 4
13 on Task 4
14 on Task 4
6 on Task 2
7 on Task 2
9 on Task 5
11 on Task 3
12 on Task 3
3 on Task 1
Press any key to continue . . .
```

```
0 on Task 1
2 on Task 1
5 on Task 2
6 on Task 2
10 on Task 3
11 on Task 3
12 on Task 3
13 on Task 3
14 on Task 3
16 on Task 3
3 on Task 1
15 on Task 4
8 on Task 4
9 on Task 4
7 on Task 2
1 on Task 5
17 on Task 3
4 on Task 1
18 on Task 3
19 on Task 3
Press any key to continue . . .
```

```
0 on Task 1
1 on Task 1
5 on Task 2
6 on Task 2
7 on Task 2
8 on Task 2
9 on Task 2
11 on Task 2
12 on Task 2
13 on Task 2
15 on Task 4
16 on Task 4
17 on Task 4
18 on Task 4
19 on Task 4
2 on Task 1
4 on Task 4
14 on Task 2
10 on Task 3
3 on Task 5
Press any key to continue . . .
```

Let's Parallel

- The following:

```
private void Test()
{
        Parallel.For(0, 20, i => Console.WriteLine("{0} on Task {1}", i, Task.CurrentId));
}
```

- Is the same as:

```
private void Test()
       {
           Parallel.For(0, 20, i =>
           {
               Console.WriteLine("{0} on Task {1}", i, Task.CurrentId);
           });
       }
```

Let's Parallel

```
ParallelOptions parallelOptions = new ParallelOptions();
parallelOptions.MaxDegreeOfParallelism = 2;

Parallel.For(0, 20, parallelOptions, i =>
{
    Console.WriteLine("{0} on Task {1}", i, Task.CurrentId);
});
```

```
0 on Task 1
1 on Task 1
2 on Task 1
3 on Task 1
4 on Task 1
5 on Task 1
6 on Task 1
7 on Task 1
8 on Task 1
10 on Task 2
11 on Task 2
12 on Task 2
13 on Task 2
14 on Task 2
15 on Task 2
16 on Task 2
9 on Task 1
17 on Task 2
18 on Task 2
19 on Task 2
Press any key to continue . . .
```

```
0 on Task 1
1 on Task 1
2 on Task 1
3 on Task 1
4 on Task 1
10 on Task 2
11 on Task 2
12 on Task 2
13 on Task 2
14 on Task 2
15 on Task 2
16 on Task 2
17 on Task 2
18 on Task 2
19 on Task 2
7 on Task 2
8 on Task 2
9 on Task 2
5 on Task 1
6 on Task 1
Press any key to continue . . .
```

Limiting the number of tasks to just 2

Parallel.Invoke

Parallel.Invoke

```
static void Main()
{
    Parallel.Invoke(
        () => RunTask(1),
        () => RunTask(2),
        () => RunTask(3),
        () => RunTask(4),
        () => RunTask(5)
        );
}

static void RunTask(int taskNumber)
{
    Console.WriteLine("Task {0} started", taskNumber);
    Console.WriteLine("Task {0} complete", taskNumber);
}
```

Parallel.Invoke

```
static void Main()
{
    Parallel.Invoke(
        () => RunTask(1),
        () => RunTask(2),
        () => RunTask(3),
        () => RunTask(4),
        () => RunTask(5)
        );
}

static void RunTask(int taskNumber)
{
    Console.WriteLine("Task {0} started", taskNumber);
    Console.WriteLine("Task {0} complete", taskNumber);
}
```

```
Task 2 started
Task 2 complete
Task 3 started
Task 4 started
Task 1 started
Task 1 complete
Task 3 complete
Task 5 started
Task 5 complete
Task 4 complete
Press any key to continue . . .
```

Parallel.Invoke

```
static void Main()
{
    Parallel.Invoke(
        () => RunTask(1),
        () => RunTask(2),
        () => RunTask(3),
        () => RunTask(4),
        () => RunTask(5)
        );
}

static void RunTask(int taskNumber)
{
    Console.WriteLine("Task {0} started", taskNumber);
    Console.WriteLine("Task {0} complete", taskNumber);
}
```

Use the following to limit the number of concurrent tasks:

```
ParallelOptions parallelOptions = new ParallelOptions();
parallelOptions.MaxDegreeOfParallelism = 2;
```

```
Task 2 started
Task 2 complete
Task 3 started
Task 4 started
Task 1 started
Task 1 complete
Task 3 complete
Task 5 started
Task 5 complete
Task 4 complete
Press any key to continue . . .
```

Threading Problems, like, **ALOT**

Threading Problems

```
class Program
{
    static int _counter = 0;

    static void Main(string[] args)
    {
        Parallel.Invoke(AddOne,
                        SubtractOne);
        Console.WriteLine("Final counter value is {0}.", _counter);
    }

    static void AddOne()
    {
        int temp = _counter;
        temp++;
        Thread.Sleep(2000);
        Console.WriteLine("Incremented counter to {0}.", temp);
        _counter = temp;
    }

    static void SubtractOne()
    {
        int temp = _counter;
        temp--;
        Thread.Sleep(2000);
        Console.WriteLine("Decremented counter to {0}.", temp);
        _counter = temp;
    }
}
```

Threading Problems

```
class Program
{
    static int _counter = 0;

    static void Main(string[] args)
    {
        Parallel.Invoke(AddOne,
                        SubtractOne);
        Console.WriteLine("Final counter value is {0}.", _counter);
    }

    static void AddOne()
    {
        int temp = _counter;
        temp++;
        Thread.Sleep(2000);
        Console.WriteLine("Incremented counter to {0}.", temp);
        _counter = temp;
    }

    static void SubtractOne()
    {
        int temp = _counter;
        temp--;
        Thread.Sleep(2000);
        Console.WriteLine("Decremented counter to {0}.", temp);
        _counter = temp;
    }
}
```

```
Incremented counter to 1.
Decremented counter to -1.
Final counter value is -1.
```

Threading Problems

```
class Program
{
    static int _counter = 0;

    static void Main(string[] args)
    {
        Parallel.Invoke(AddOne,
                        SubtractOne);
        Console.WriteLine("Final counter value is {0}.", _counter);
    }

    static void AddOne()
    {
        int temp = _counter;
        temp++;
        Thread.Sleep(2000);
        Console.WriteLine("Incremented counter to {0}.", temp);
        _counter = temp;
    }

    static void SubtractOne()
    {
        int temp = _counter;
        temp--;
        Thread.Sleep(2000);
        Console.WriteLine("Decremented counter to {0}.", temp);
        _counter = temp;
    }
}
```

```
Incremented counter to 1.
Decremented counter to -1.
Final counter value is -1.
```

BAM!

Threading Problems

```
class Program
{
    static int _counter = 0;

    static void Main(string[] args)
    {
        Parallel.Invoke(AddOne,
                        SubtractOne);
        Console.WriteLine("Final counter value is {0}.", _counter);
    }

    static void AddOne()
    {
        int temp = _counter;
        temp++;
        Thread.Sleep(2000);
        Console.WriteLine("Incremented counter to {0}.", temp);
        _counter = temp;
    }

    static void SubtractOne()
    {
        int temp = _counter;
        temp--;
        Thread.Sleep(2000);
        Console.WriteLine("Decremented counter to {0}.", temp);
        _counter = temp;
    }
}
```

Race Conditions

```
Incremented counter to 1.
Decremented counter to -1.
Final counter value is -1.
```

Racing Conditions

The Solution

Threading Problems

```
class Program
{
    static int _counter = 0;
    static object _lock = new object();

    static void Main(string[] args)
    {
        Parallel.Invoke(AddOne,
                          SubtractOne);
        Console.WriteLine("Final counter value is {0}.", _counter);
    }

    static void AddOne()
    {
        lock (_lock)
        {
            int temp = _counter;
            temp++;
            Thread.Sleep(2000);
            Console.WriteLine("Incremented counter to {0}.", temp);
            _counter = temp;
        }
    }

    static void SubtractOne()
    {
        lock (_lock)
        {
            int temp = _counter;
            temp--;
            Thread.Sleep(2000);
            Console.WriteLine("Decremented counter to {0}.", temp);
            _counter = temp;
        }
    }
}
```

Race Conditions

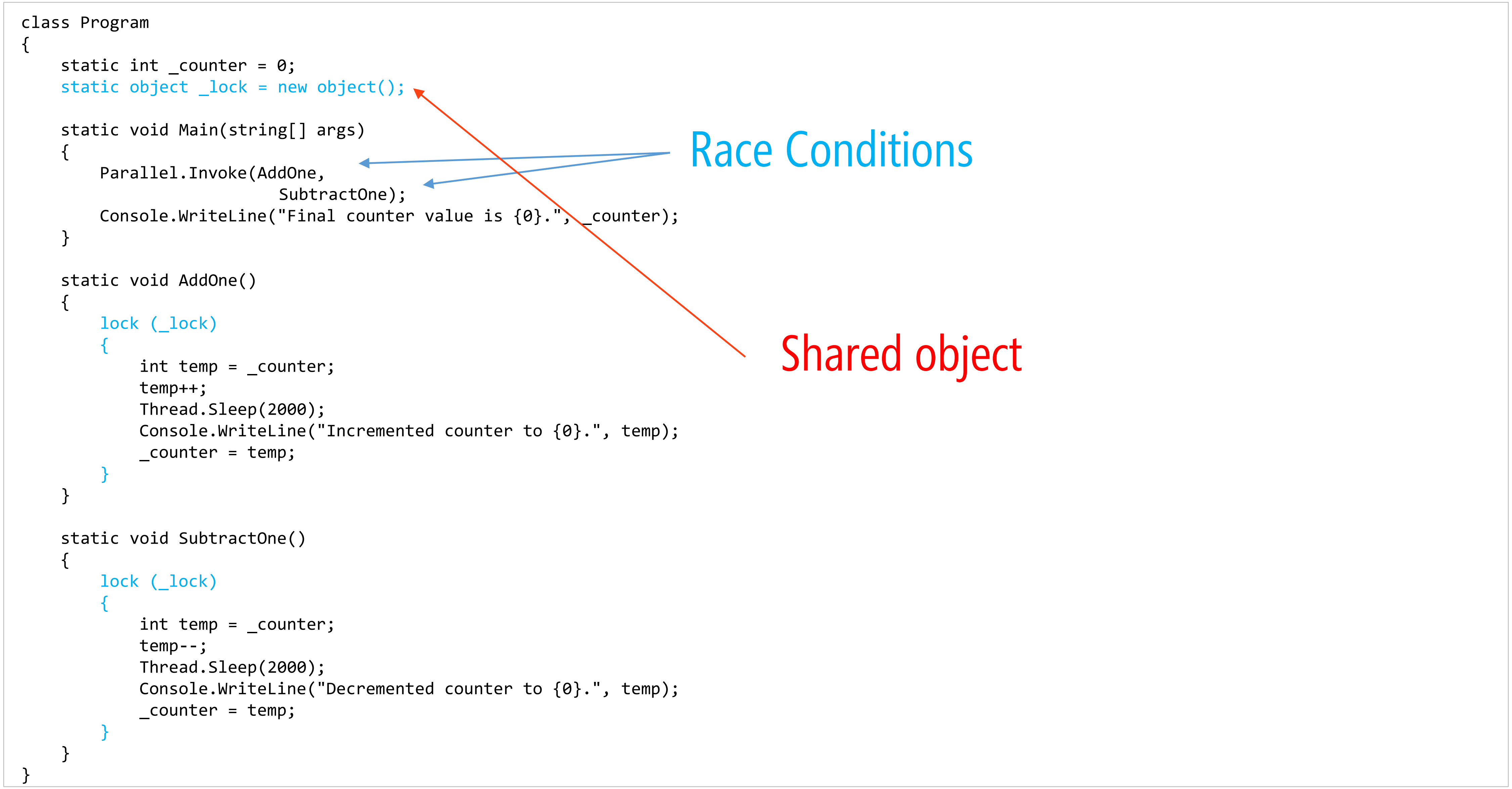
Threading Problems

```
class Program
{
    static int _counter = 0;
    static object _lock = new object();

    static void Main(string[] args)
    {
        Parallel.Invoke(AddOne,
                             SubtractOne);
        Console.WriteLine("Final counter value is {0}.", _counter);
    }

    static void AddOne()
    {
        lock (_lock)
        {
            int temp = _counter;
            temp++;
            Thread.Sleep(2000);
            Console.WriteLine("Incremented counter to {0}.", temp);
            _counter = temp;
        }
    }

    static void SubtractOne()
    {
        lock (_lock)
        {
            int temp = _counter;
            temp--;
            Thread.Sleep(2000);
            Console.WriteLine("Decremented counter to {0}.", temp);
            _counter = temp;
        }
    }
}
```

Race Conditions

New

Threading Problems

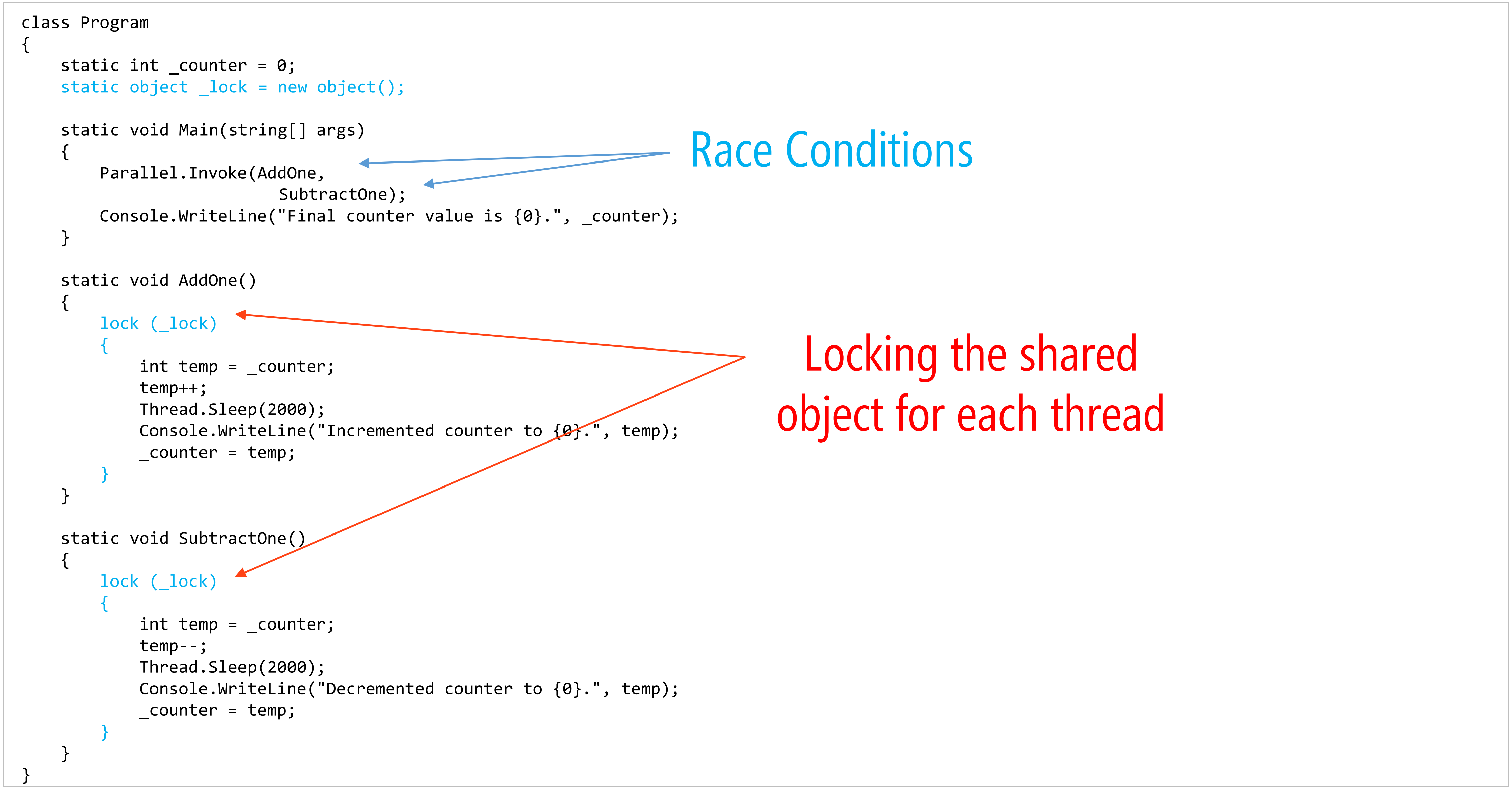


```
class Program
{
    static int _counter = 0;
    static object _lock = new object();

    static void Main(string[] args)
    {
        Parallel.Invoke(AddOne,
                        SubtractOne);
        Console.WriteLine("Final counter value is {0}.", _counter);
    }

    static void AddOne()
    {
        lock (_lock)
        {
            int temp = _counter;
            temp++;
            Thread.Sleep(2000);
            Console.WriteLine("Incremented counter to {0}.", temp);
            _counter = temp;
        }
    }

    static void SubtractOne()
    {
        lock (_lock)
        {
            int temp = _counter;
            temp--;
            Thread.Sleep(2000);
            Console.WriteLine("Decremented counter to {0}.", temp);
            _counter = temp;
        }
    }
}
```

Race Conditions

Shared object

Threading Problems



```
class Program
{
    static int _counter = 0;
    static object _lock = new object();

    static void Main(string[] args)
    {
        Parallel.Invoke(AddOne,
                        SubtractOne);
        Console.WriteLine("Final counter value is {0}.", _counter);
    }

    static void AddOne()
    {
        lock (_lock)
        {
            int temp = _counter;
            temp++;
            Thread.Sleep(2000);
            Console.WriteLine("Incremented counter to {0}.", temp);
            _counter = temp;
        }
    }

    static void SubtractOne()
    {
        lock (_lock)
        {
            int temp = _counter;
            temp--;
            Thread.Sleep(2000);
            Console.WriteLine("Decremented counter to {0}.", temp);
            _counter = temp;
        }
    }
}
```

Race Conditions

Locking the shared object for each thread

Threading Problems

```
class Program
{
    static int _counter = 0;
    static object _lock = new object();

    static void Main(string[] args)
    {
        Parallel.Invoke(AddOne,
                        SubtractOne);
        Console.WriteLine("Final counter value is {0}.", _counter);
    }

    static void AddOne()
    {
        lock (_lock)
        {
            int temp = _counter;
            temp++;
            Thread.Sleep(2000);
            Console.WriteLine("Incremented counter to {0}.", temp);
            _counter = temp;
        }
    }

    static void SubtractOne()
    {
        lock (_lock)
        {
            int temp = _counter;
            temp--;
            Thread.Sleep(2000);
            Console.WriteLine("Decremented counter to {0}.", temp);
            _counter = temp;
        }
    }
}
```

```
Incremented counter to 1.
Decremented counter to 0.
Final counter value is 0.
```

The right output

Lock (*Object*)

Threading Problems

```
class Program
{
    static int _counter = 0;
    static object _lock = new object();

    static void Main(string[] args)
    {
        Parallel.Invoke(AddOne,
                          SubtractOne);
        Console.WriteLine("Final counter value is {0}.", _counter);
    }

    static void AddOne()
    {
        lock (_lock)
        {
            int temp = _counter;
            temp++;
            Thread.Sleep(2000);
            Console.WriteLine("Incremented counter to {0}.", temp);
            _counter = temp;
        }
    }

    static void SubtractOne()
    {
        lock (_lock)
        {
            int temp = _counter;
            temp--;
            Thread.Sleep(2000);
            Console.WriteLine("Decremented counter to {0}.", temp);
            _counter = temp;
        }
    }
}
```

Not good, don’t do this,
or lock a string or etc.

Threading Problems

```
class Program
{
    static int _counter = 0;
    static object _lock = new object();

    static void Main(string[] args)
    {
        Parallel.Invoke(AddOne,
                          SubtractOne);
        Console.WriteLine("Final counter value is {0}.", _counter);
    }

    static void AddOne()
    {
        lock (_lock)
        {
            int temp = _counter;
            temp++;
            Thread.Sleep(2000);
            Console.WriteLine("Incremented counter to {0}.", temp);
            _counter = temp;
        }
    }

    static void SubtractOne()
    {
        lock (_lock)
        {
            int temp = _counter;
            temp--;
            Thread.Sleep(2000);
            Console.WriteLine("Decremented counter to {0}.", temp);
            _counter = temp;
        }
    }
}
```

The best choice for a locking object is a private or protected object defined within the class that controls the shared state.

`Thread` Class

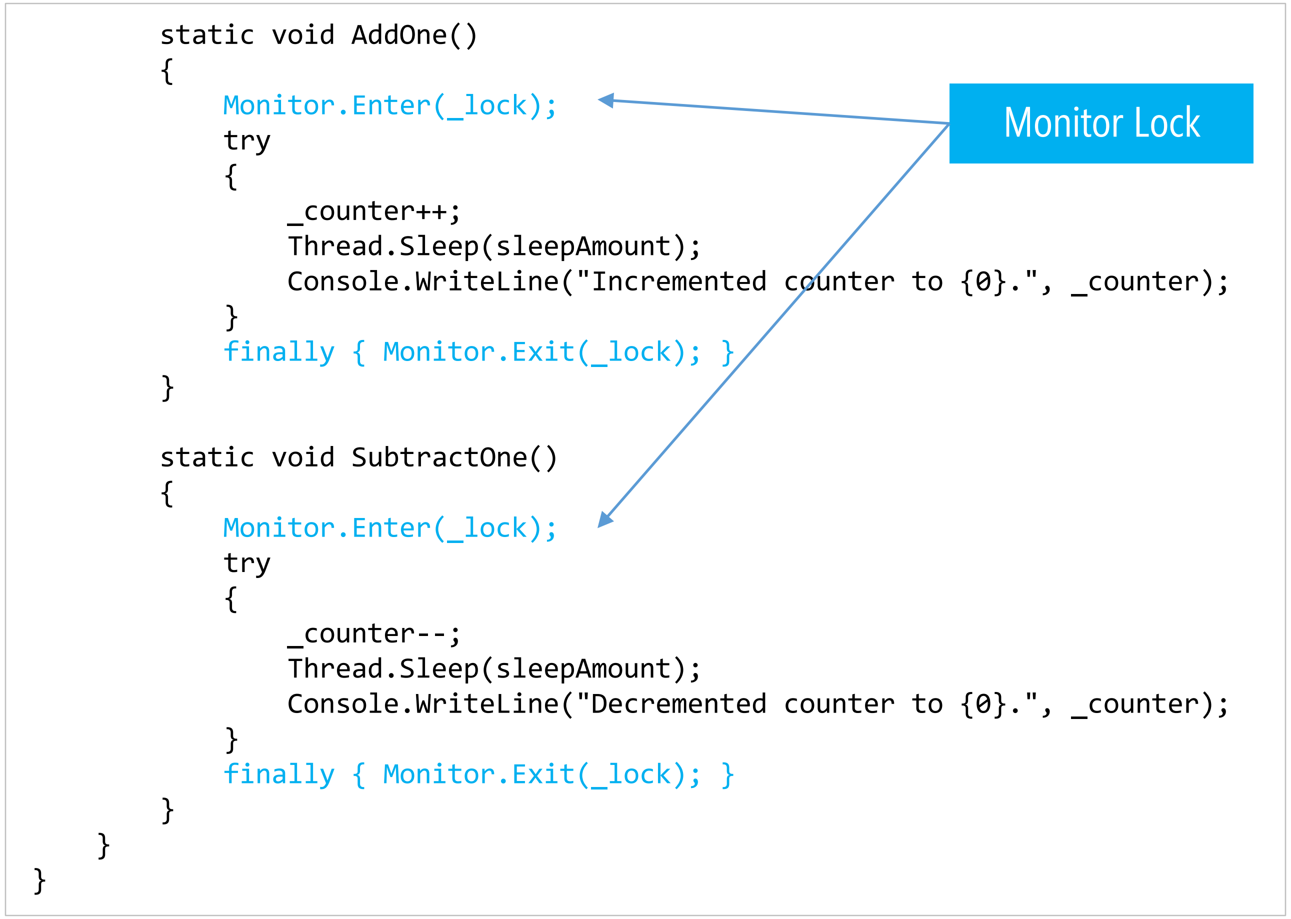
Threading Problems

```
class Program
{
    static int _counter = 0;
    static object _lock = new object();
    const int sleepAmount = 500;

    static void Main(string[] args)
    {
        Thread tAdd = new Thread(AddOne);
        Thread tSub = new Thread(SubtractOne);

        tAdd.Start();
        tSub.Start();

        Console.WriteLine("Final counter value is {0}.", _counter);
    }
```

```
        static void AddOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter++;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Incremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }

        static void SubtractOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter--;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Decremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }
    }
}
```

Threading Problems

```
    class Program
    {
        static int _counter = 0;
        static object _lock = new object();
        const int sleepAmount = 500;

        static void Main(string[] args)
        {
            Thread tAdd = new Thread(AddOne);
            Thread tSub = new Thread(SubtractOne);

            tAdd.Start();
            tSub.Start();

            Console.WriteLine("Final counter value is {0}.", _counter);
        }
```



```
        static void AddOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter++;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Incremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }

        static void SubtractOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter--;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Decremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }
    }
}
```

Threading Problems

```
    class Program
    {
        static int _counter = 0;
        static object _lock = new object();
        const int sleepAmount = 500;

        static void Main(string[] args)
        {
            Thread tAdd = new Thread(AddOne);
            Thread tSub = new Thread(SubtractOne);

            tAdd.Start();
            tSub.Start();

            Console.WriteLine("Final counter value is {0}.", _counter);
        }
```

```
        static void AddOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter++;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Incremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }

        static void SubtractOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter--;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Decremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }
    }
}
```

Acquire Lock

Threading Problems

```
    class Program
    {
        static int _counter = 0;
        static object _lock = new object();
        const int sleepAmount = 500;

        static void Main(string[] args)
        {
            Thread tAdd = new Thread(AddOne);
            Thread tSub = new Thread(SubtractOne);

            tAdd.Start();
            tSub.Start();

            Console.WriteLine("Final counter value is {0}.", _counter);
        }
```

```
        static void AddOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter++;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Incremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }

        static void SubtractOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter--;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Decremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }
    }
}
```

Release Lock

Threading Problems

```
    class Program
    {
        static int _counter = 0;
        static object _lock = new object();
        const int sleepAmount = 500;

        static void Main(string[] args)
        {
            Thread tAdd = new Thread(AddOne);
            Thread tSub = new Thread(SubtractOne);

            tAdd.Start();
            tSub.Start();

            Console.WriteLine("Final counter value is {0}.", _counter);
        }
```

```
        static void AddOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter++;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Incremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }

        static void SubtractOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter--;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Decremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }
    }
}
```

```
Final counter value is 0.
Incremented counter to 1.
Decremented counter to 0.
Press any key to continue . . .
```

Threading Problems

```
    class Program
    {
        static int _counter = 0;
        static object _lock = new object();
        const int sleepAmount = 500;

        static void Main(string[] args)
        {
            Thread tAdd = new Thread(AddOne);
            Thread tSub = new Thread(SubtractOne);

            tAdd.Start();
            tSub.Start();

            Console.WriteLine("Final counter value is {0}.", _counter);
        }
```

```
        static void AddOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter++;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Incremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }

        static void SubtractOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter--;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Decremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }
    }
}
```

```
Final counter value is 0.
Incremented counter to 1.
Decremented counter to 0.
Press any key to continue . . .
```

Why?

Threading Problems

```
class Program
{
    static int _counter = 0;
    static object _lock = new object();
    const int sleepAmount = 500;

    static void Main(string[] args)
    {
        Thread tAdd = new Thread(AddOne);
        Thread tSub = new Thread(SubtractOne);

        tAdd.Start();
        tSub.Start();

        tAdd.Join();
        tSub.Join();

        Console.WriteLine("Final counter value is {0}.", _counter);
    }
```

Blocks the calling thread until the thread terminates

```
        static void AddOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter++;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Incremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }

        static void SubtractOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter--;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Decremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }
    }
}
```

Threading Problems

```
    class Program
    {
        static int _counter = 0;
        static object _lock = new object();
        const int sleepAmount = 500;

        static void Main(string[] args)
        {
            Thread tAdd = new Thread(AddOne);
            Thread tSub = new Thread(SubtractOne);

            tAdd.Start();
            tSub.Start();

            tAdd.Join();
            tSub.Join();

            Console.WriteLine("Final counter value is {0}.", _counter);
        }
```

Blocks the calling thread until the thread terminates

```
        static void AddOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter++;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Incremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }

        static void SubtractOne()
        {
            Monitor.Enter(_lock);
            try
            {
                _counter--;
                Thread.Sleep(sleepAmount);
                Console.WriteLine("Decremented counter to {0}.", _counter);
            }
            finally { Monitor.Exit(_lock); }
        }
    }
}
```

```
Incremented counter to 1.
Decremented counter to 0.
Final counter value is 0.
Press any key to continue . . .
```

Task.Factory

```
var myTask = Task.Factory.StartNew(() => { return "Hello, world!"; });
Console.WriteLine(myTask.Result);
```

Tast Wait

Wait

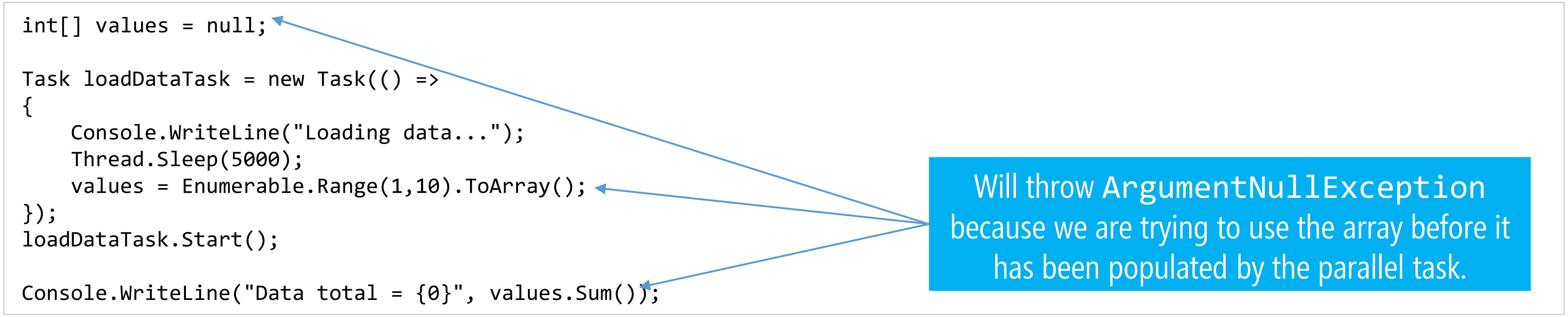
```
int[] values = null;

Task loadDataTask = new Task(() =>
{
    Console.WriteLine("Loading data...");
    Thread.Sleep(5000);
    values = Enumerable.Range(1,10).ToArray();
});
loadDataTask.Start();

Console.WriteLine("Data total = {0}", values.Sum());
```

Wait

```
int[] values = null;

Task loadDataTask = new Task(() =>
{
    Console.WriteLine("Loading data...");
    Thread.Sleep(5000);
    values = Enumerable.Range(1,10).ToArray();
});
loadDataTask.Start();

Console.WriteLine("Data total = {0}", values.Sum());
```

Will throw `ArgumentNullException` because we are trying to use the array before it has been populated by the parallel task.

Wait



```
int[] values = null;

Task loadDataTask = new Task(() =>
{
    Console.WriteLine("Loading data...");
    Thread.Sleep(5000);
    values = Enumerable.Range(1,10).ToArray();
});
loadDataTask.Start();

Console.WriteLine("Data total = {0}", values.Sum());
```

Will throw `ArgumentNullException` because we are trying to use the array before it has been populated by the parallel task.

```
int[] values = null;

Task loadDataTask = new Task(() =>
{
    Console.WriteLine("Loading data...");
    Thread.Sleep(5000);
    values = Enumerable.Range(1,10).ToArray();
});
loadDataTask.Start();
loadDataTask.Wait();
loadDataTask.Dispose();

Console.WriteLine("Data total = {0}", values.Sum());    // Data total = 55
```

Wait

```
int[] values = null;

Task loadDataTask = new Task(() =>
{
    Console.WriteLine("Loading data...");
    Thread.Sleep(5000);
    values = Enumerable.Range(1,10).ToArray();
});
loadDataTask.Start();

Console.WriteLine("Data total = {0}", values.Sum());
```

```
int[] values = null;

Task loadDataTask = new Task(() =>
{
    Console.WriteLine("Loading data...");
    Thread.Sleep(5000);
    values = Enumerable.Range(1,10).ToArray();
});
loadDataTask.Start();
loadDataTask.Wait();
loadDataTask.Dispose();

Console.WriteLine("Data total = {0}", values.Sum());    // Data total = 55
```

To fix it, we will now `wait` for the task to be done

Wait

```
int[] values = null;

Task loadDataTask = new Task(() =>
{
    Console.WriteLine("Loading data...");
    Thread.Sleep(5000);
    values = Enumerable.Range(1,10).ToArray();
});
loadDataTask.Start();

Console.WriteLine("Data total = {0}", values.Sum());
```

```
int[] values = null;

Task loadDataTask = new Task(() =>
{
    Console.WriteLine("Loading data...");
    Thread.Sleep(5000);
    values = Enumerable.Range(1,10).ToArray();
});
loadDataTask.Start();
loadDataTask.Wait();
loadDataTask.Dispose();

Console.WriteLine("Data total = {0}", values.Sum());
```

The output will be Data total = 55

Tasks Continuation

```
Task continuation = firstTask.ContinueWith(antecedent => { /* functionality */ });
```