Ten years ago, I built a complicated, multithreaded C# application for a trade show, and I could never get it completely stable, the sales team that demoed the app got used to restart the program a couple of times to get it to work. What was the problem?

I wrote multithreaded C# code, but this was years before the task parallel library and parallel LINQ were released, so I had to do everything manually, locking shared fields, synchronizing threads and I have overlooked several sections of code that were not thread safe.

In my tests, everything seems fine, but in production, the program behaved erratically. I did eventually fix the problem but I created this course to make sure this kind of thing will never to you. Therefore, I will teach you how to write robust multithreaded code using tasks and parallel LINQ that works perfectly on your first try.

We will work towards a simple goal, have one thread reliably return data to another thread. But even though this sounds very simple, you will learn that it is actually very hard to write code that does this reliably. We will end the section with a working code example that uses locking and two-way synchronization to get the job done.

In the next section of the course, I will show you two asynchronous programming libraries in the .Net Framework that have a robust locking and synchronization built-in. By using these libraries, you avoid many common pitfalls of multi-threaded programming because all of the complex code has already been written for you.

In the first section, we will take a closer look at the task parallel library. This library introduces the Task class, the workhorse of asynchronous programming in C#. Tasks make it very easy to start an asynchronous operation on another thread, wait for completion and then return a result to the calling thread.

In the final section, I will demonstrate the Parallel LINQ Library or PLINQ. This library is a parallel version of the regular LINQ library and it lets you executes LINQ queries in parallel. This library makes it very easy to implement Map Reduce operations in parallel. I will describe how to make your LINQ queries run in parallel.

What the consequences are for the ordering of the results and what the limitations of the Parallel LINQ library are.