

Threads

Teil 2

Threads erzeugen

- eine neue Instanz der Klasse Thread anlegen
- Der Konstruktor von Thread hat ein einziges Argument, eine delegate-Instanz. Die CLR bietet dafür die Delegate-Klasse ThreadStart an, die auf eine benannte Methode verweist.
- Die Deklaration des ThreadStart-Delegate ist:
public delegate void ThreadStart();
- die Methode darf keine Parameter haben und muss void liefern
Thread myThread = new Thread(new ThreadStart(myMethode));
- ab der Version 2.0 des .NET Framework, ist es nicht mehr erforderlich explizit einen Delegaten zu erzeugen
Thread myThread = new Thread(myMethode);

```
using System.Threading;
namespace Multithreading.Test{
    class StandardThread {
        public static void Main(string[] args){
            Console.WriteLine("1");
            Console.WriteLine("2");
            Console.WriteLine("3");
            Console.WriteLine("A");
            Console.WriteLine("B");
            Console.WriteLine("C");
        }
    }
}
```

```
using System.Threading;
namespace Multithreading.Test{
    class StandardThread {
        public static void Main(string[] args){
            Thread t1 = new Thread(new ThreadStart(ZahlenAnzeigen));
            Thread t2 = new Thread(new ThreadStart(BuchstabeAnzeigen));
            // Threads starten
            t1.Start();
            t2.Start();
        }
        public static void ZahlenAnzeigen(){
            Console.WriteLine("1");
            Console.WriteLine("2");
            Console.WriteLine("3");
        }
        public static BuchstabeAnzeigen () {
            Console.WriteLine("A");
            Console.WriteLine("B");
            Console.WriteLine("C");
        }
    }
}
```

Schreiben Sie folgende Anwendung:

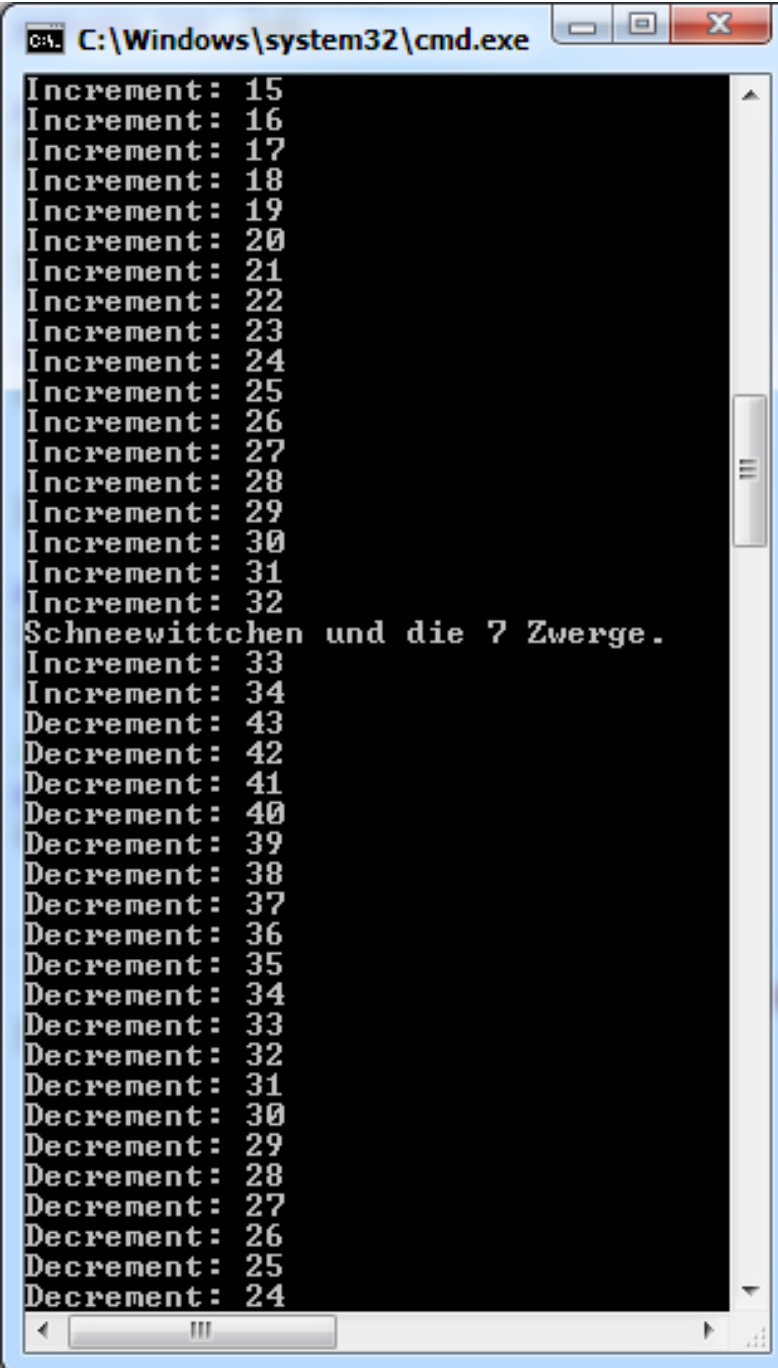
Im Mainthread werden zwei Threads gestartet. Der erste Thread inkrementiert eine Variable von 0 bis 100. Der zweite Thread dekrementiert eine Variable von 100 bis 0. Geben Sie die Variablenwerte aus. Zeigen Sie an, welcher Thread die Ausgabe durchführt.

```
using System.Threading;
namespace Multithreading.Test{
    class StandardThread {
        public static void Main(string[] args){
            Thread t1 = new Thread(new ThreadStart(Increment));
            Thread t2 = new Thread(new ThreadStart(Decrement));
            // Threads starten
            t1.Start();
            t2.Start();
        }
        public static void Increment(){
            for (int i = 0; i < 100; i++) {
                Console.WriteLine("Increment: {0}", i);
            }
        }
        public static void Decrement() {
            for (int i = 100; i >= 0; i--) {
                Console.WriteLine("Decrement: {0}", i);
            }
        }
    }
}
```

```
C:\Windows\system32\cmd.exe

Increment: 0
Increment: 1
Increment: 2
Increment: 3
Increment: 4
Increment: 5
Increment: 6
Decrement: 100
Decrement: 99
Decrement: 98
Decrement: 97
Decrement: 96
Decrement: 95
Decrement: 94
Decrement: 93
Decrement: 92
Decrement: 91
Decrement: 90
Decrement: 89
Decrement: 88
Decrement: 87
Decrement: 86
Decrement: 85
Decrement: 84
Decrement: 83
Decrement: 82
Decrement: 81
Decrement: 80
Decrement: 79
Decrement: 78
Increment: 7
Increment: 8
Increment: 9
Increment: 10
Increment: 11
Increment: 12
Increment: 13
Decrement: 77
Decrement: 76
Decrement: 75
Decrement: 74
Decrement: 73
Decrement: 72
Decrement: 71
Decrement: 70
```

```
using System.Threading;
namespace Multithreading.Test {
    class ParameterizedThread {
        public static void Main(string[] args) {
            Thread t1 = new Thread(Increment);
            Thread t2 = new Thread(Decrement);
            // Threads starten
            t1.Start(0);
            t2.Start(100);
        }
        public static void Increment(object o) {
            for (int i = Convert.ToInt32(o); i < 100; i++) {
                Console.WriteLine("Increment: {0}", i);
            }
        }
        public static void Decrement(object o) {
            for (int i = Convert.ToInt32(o); i >= 0; i--) {
                Console.WriteLine("Decrement: {0}", i);
            }
        }
    }
}
```

```
C:\Windows\system32\cmd.exe
Increment: 15
Increment: 16
Increment: 17
Increment: 18
Increment: 19
Increment: 20
Increment: 21
Increment: 22
Increment: 23
Increment: 24
Increment: 25
Increment: 26
Increment: 27
Increment: 28
Increment: 29
Increment: 30
Increment: 31
Increment: 32
Schneewittchen und die 7 Zwerge.
Increment: 33
Increment: 34
Decrement: 43
Decrement: 42
Decrement: 41
Decrement: 40
Decrement: 39
Decrement: 38
Decrement: 37
Decrement: 36
Decrement: 35
Decrement: 34
Decrement: 33
Decrement: 32
Decrement: 31
Decrement: 30
Decrement: 29
Decrement: 28
Decrement: 27
Decrement: 26
Decrement: 25
Decrement: 24
```

```
public class MyThreadClass {  
    // Automatische Eigenschaften in C# 3.0  
    public string Name { get; set; }  
    public int Number { get; set; }  
  
    public MyThreadClass()  
    {  
        // Thread Funktion  
        public void ShowSentence()  
        {  
            // Empfange die Argumente  
            string localName = Name;  
            int localNumber = Number;  
            Console.WriteLine("{0} und die {1} Zwerge.", localName, localNumber);  
        }  
    }  
}
```

Main:

```
...  
MyThreadClass m = new MyThreadClass();  
m.Name = "Schneewittchen";  
m.Number = 7;  
  
Thread t3 = new Thread(m.ShowSentence);  
t3.Start();
```

Threads blockieren

Die Klasse Thread bietet diverse Methoden an, um einen laufenden Thread zu blockieren.

- Suspend:** die Unterbrechung eines Thread
- Sleep:** das Schlafen legen
- Join:** das Warten auf die Beendigung eines anderen Threads

Sleep

```
using System.Threading;
Namespace Multithreading.Test{
    class SleepingThread {
        public static void Main(string[] args) {
            Thread t1 = new Thread(new ThreadStart(LazyIncrement));
            t1.Start();
        }
        public static void LazyIncrement() {
            for (int i = 0; i < 100; i++) {
                Console.WriteLine("Lazy Increment: {0}", i);
                if (i % 10 == 0) {
                    Console.WriteLine("Sleep...");
                    Thread.Sleep(1000);
                }
            }
        }
    }
}
```

C:\Windows\system32\cmd.exe

```
Lazy Increment: 20  
Sleep...  
Lazy Increment: 21  
Lazy Increment: 22  
Lazy Increment: 23  
Lazy Increment: 24  
Lazy Increment: 25  
Lazy Increment: 26  
Lazy Increment: 27  
Lazy Increment: 28  
Lazy Increment: 29  
Lazy Increment: 30  
Sleep...  
Lazy Increment: 31  
Lazy Increment: 32  
Lazy Increment: 33  
Lazy Increment: 34  
Lazy Increment: 35  
Lazy Increment: 36  
Lazy Increment: 37  
Lazy Increment: 38  
Lazy Increment: 39  
Lazy Increment: 40  
Sleep...
```

SpinWait

Der aufgerufene Thread muss für eine bestimmte Anzahl an Iterationen warten und wird in dieser Zeit nicht in die Schlange der auf die Ausführung wartenden Threads eingereiht. Der Thread tritt in den Zustand "WaitSleepJoin" ein ohne dabei sein noch bestehendes Quantum zu verlieren.

```
const int VALUE = 2000;  
Thread.SpinWait(VALUE);
```

Die SpinWait-Methode ist beispielsweise nützlich zum Implementieren von Sperren. Klassen, wie Monitor und ReaderWriterLock, verwenden diese Methode intern. Bestimmte Operationen können damit effizienter gestaltet werden, da kein Kontextwechsel zwangsläufig stattfindet, ein Vorgang im OS, der sehr kostenintensiv ist.

Suspend/Resume

Die Methoden `Thread.Suspend` und `Thread.Resume` dienen dazu, Threads zu unterbrechen und wieder fortzusetzen. Jeder Thread kann die Methode `Suspend` an einem Thread-Objekt aufrufen, inklusive dem Thread, der unterbrochen werden soll. Die Fortsetzung eines Thread mit `Resume` muss natürlich von einem laufenden Thread heraus erfolgen und kann nicht von dem unterbrochenen Thread erzwungen werden.

Die Unterbrechung eines Thread ist keine unmittelbare Operation. `Suspend` und `Resume` werden oftmals benutzt um die Ausführung von Threads zu synchronisieren.

Der direkte Gebrauch von `Suspend` und `Resume` ist nicht empfehlenswert, weil der Programmierer nicht weiß, wann die Operationen ausgeführt werden.

Join

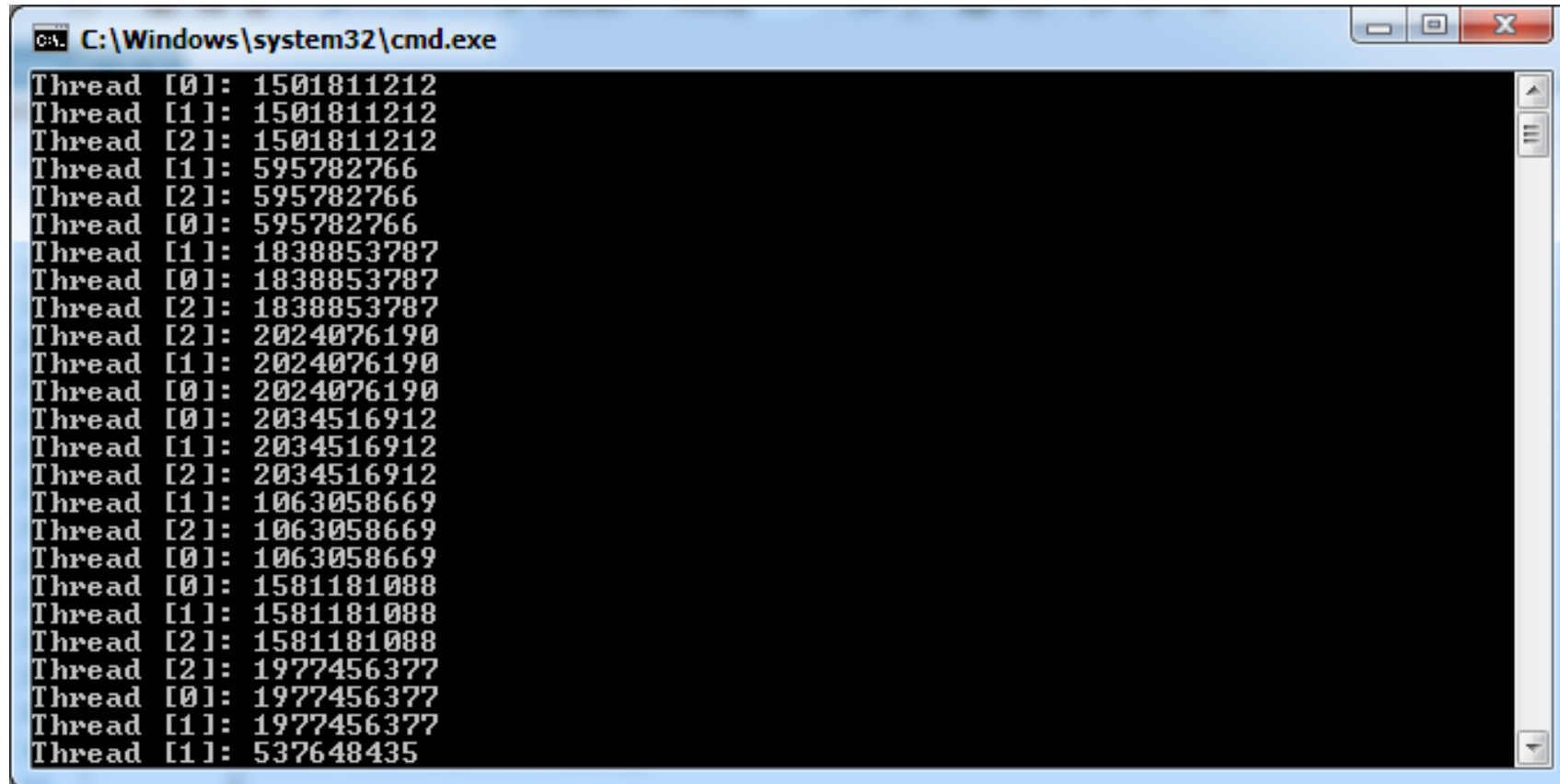
Veranlasst einen Thread dazu, die Verarbeitung zu unterbrechen und darauf zu warten, dass ein anderer Thread seine Arbeit abgeschlossen hat. Als würde man die Spitze des ersten Threads mit dem Ende des zweiten Threads verknüpfen, also »vereinigen«.

Um einen Thread t1 mit einem zweiten Thread t2 zu vereinigen, schreibt man innerhalb der Methode von Thread t1:

```
t2.Join();
```

```
foreach(Thread myThread in myThreads) //ThreadCollection
{
    myThread.Join();
}
```


Join



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a blue title bar and standard Windows window controls (minimize, maximize, close) in the top right corner. The command prompt area is black with white text. It displays a list of 20 thread identifiers, each on a new line. The format for each line is "Thread [index]: [hexadecimal ID]". The indices range from [0] to [1] for each of the 10 thread groups. The hexadecimal IDs are: 1501811212, 595782766, 1838853787, 2024076190, 2034516912, 1063058669, 1581181088, 1977456377, and 537648435.

```
C:\Windows\system32\cmd.exe
Thread [0]: 1501811212
Thread [1]: 1501811212
Thread [2]: 1501811212
Thread [1]: 595782766
Thread [2]: 595782766
Thread [0]: 595782766
Thread [1]: 1838853787
Thread [0]: 1838853787
Thread [2]: 1838853787
Thread [2]: 2024076190
Thread [1]: 2024076190
Thread [0]: 2024076190
Thread [0]: 2034516912
Thread [1]: 2034516912
Thread [2]: 2034516912
Thread [1]: 1063058669
Thread [2]: 1063058669
Thread [0]: 1063058669
Thread [0]: 1581181088
Thread [1]: 1581181088
Thread [2]: 1581181088
Thread [2]: 1977456377
Thread [0]: 1977456377
Thread [1]: 1977456377
Thread [1]: 537648435
```

Threads abbrechen

- Thread beendet sich, wenn er seine Aufgabe erledigt hat
- Thread kann auch vorzeitig abgebrochen werden
- saubere Methode: Flag setzen über boolsche Variable

```
if (KeepAlive == false) {
    return;
}
```

//Projektmappe:Threadzustände Projekt: EndThread

- Thread.Interrupt() fordert den Thread auf, sich selbst zu beenden
- Thread.Abort() löst die ThreadAbortException aus, die der abgebrochene Thread abfangen kann. Signal für Thread, sich sofort zu beenden.

//Projektmappe:Threadzustände Projekt:ThreadAbort