

# Grundlagen in C# und .net

- Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
- Hochschule Harz
- FB Automatisierung und Informatik
- mwilhelm@hs-harz.de
- Raum 2.202
- Tel. 03943 / 659 338



## Inhalt

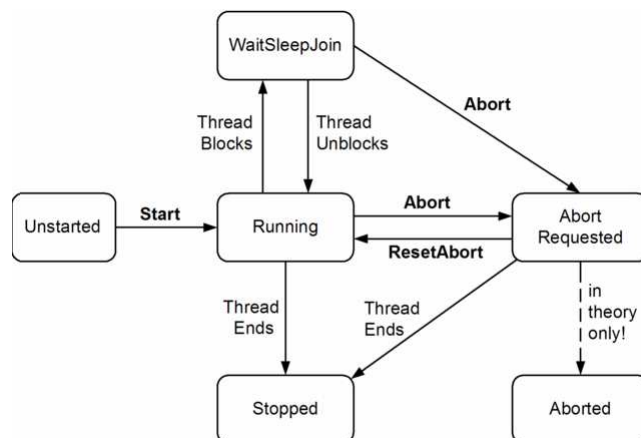
- TabbedPane (Register)
- ListView
- Tree
- Tabelle
- MDI-Programme
- Erweiterte Grafik
- **Threads und Semaphore**
- Datenbanken



## Threads

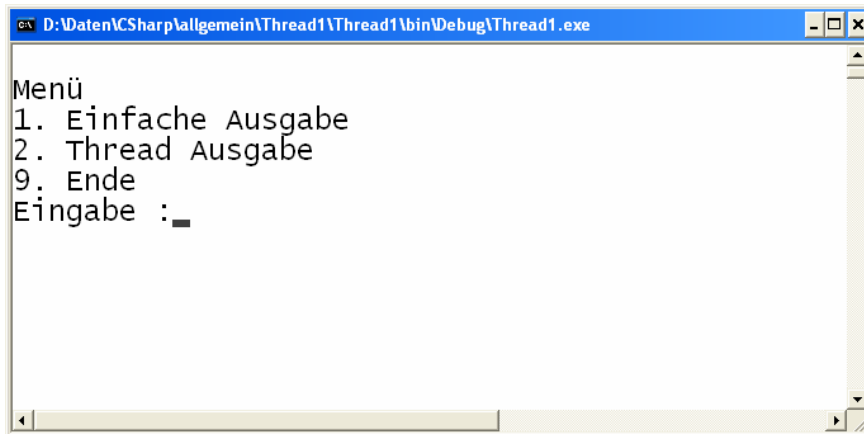
- Threads erlauben eine Parallalität
  - Auslastung von Multi-Core-Rechner
  - laufen im gleichen Adressraum eines Prozesses
- Threads werden in Java durch eigene Klassen unterstützt. Der Aufwand zur Erzeugung ist dadurch minimal.
  - Ableiten der Thread, Methode run
- Threads in .net
  - Verwendet die Delegate-Technik, Methode `public void name();`
  - Erzeugen einer ThreadStart Instanz
  - Erzeugen eines Threads
  - Starten des Threads

## Zustände von Threads

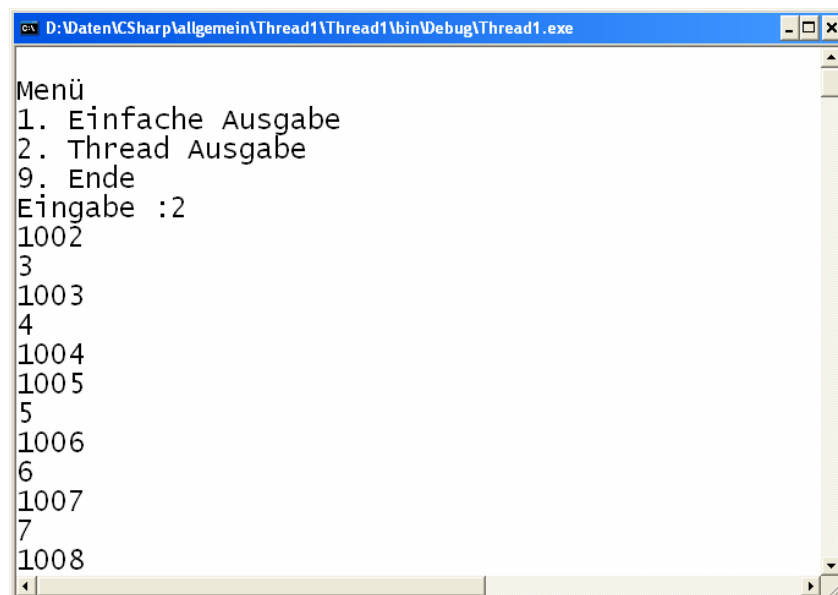


Quelle: <http://www.albahari.com/threading/part2.aspx>

## Threads: 1. Beispiel (Console)



```
CAV D:\Daten\CSharp\allgemein\Thread1\Thread1\bin\Debug\Thread1.exe
Menü
1. Einfache Ausgabe
2. Thread Ausgabe
9. Ende
Eingabe :_
```



```
CAV D:\Daten\CSharp\allgemein\Thread1\Thread1\bin\Debug\Thread1.exe
Menü
1. Einfache Ausgabe
2. Thread Ausgabe
9. Ende
Eingabe :2
1002
3
1003
4
1004
1005
5
1006
6
1007
7
1008
```

## Threads: 1. Beispiel (Console)

```
// siehe Java, Klasse Thread
public void run1() {
    for (int i = 1; i <= 100; i++)
    {
        string sStr = i.ToString();
        Console.WriteLine(sStr);
        Thread.Sleep(10);
    }
}

// siehe Java, Klasse Thread
public void run2() {
    for (int i = 1; i <= 100; i++)
    {
        string sStr = (1000+i).ToString();
        Console.WriteLine(sStr);
        Thread.Sleep(10);
    }
}

private void Ausgabe1()
{
    run1();
    run2();
}
```



## Threads: 1. Beispiel

```
private void Ausgabe2() {
    ThreadStart ts1, ts2;
    ts1 = new ThreadStart(run1); // run1 Delegates
    ts2 = new ThreadStart(run2); // run2 Delegates

    Thread t1, t2;
    t1 = new Thread(ts1);
    t2 = new Thread(ts2);
    t1.Start();
    t2.Start();
    Console.WriteLine("Fertig");
}
```



## Threads: 2. Beispiel, eigene Klasse (Console)

```
static void Main(string[] args) {  
    testThread prog = new testThread(); // Instanz muss vorhanden sein  
  
    ThreadStart ts1, ts2;  
    ts1 = new ThreadStart(prog.run1);  
    ts2 = new ThreadStart(prog.run2);  
  
    Thread t1, t2;  
    t1 = new Thread(ts1);  
    t2 = new Thread(ts2);  
    t1.Start();  
    t2.Start();  
}
```



## Threads: 2. Beispiel, eigene Klasse, gemeinsame Variable

```
class testThread {  
    int number = 0;  
    public void run1()  
    {  
        while (true)  
        {  
            int k;  
            k = number;  
            k++;  
            number = k;  
            if (number > 100) break;  
            Console.WriteLine("A: "+ number);  
        }  
    }  
    public void run2() {}  
}
```

Man erwartet alle Zahlen von 1 bis 100



## Threads: 2. Beispiel, Ergebnis

A: 1	A: 15	B: 28	A: 41
A: 3	A: 16	B: 29	B: 34
A: 4	A: 17	B: 30	B: 43
A: 5	B: 2	B: 31	B: 44
A: 6	B: 19	B: 32	B: 45
A: 7	B: 20	B: 33	B: 46
A: 8	B: 21	A: 18	B: 47
A: 9	B: 22	A: 35	B: 48
A: 10	B: 23	A: 36	B: 49
A: 11	B: 24	A: 37	A: 42
A: 12	B: 25	A: 38	B: 50
A: 13	B: 26	A: 39	
A: 14	B: 27	A: 40	



## Ausgabe mit Thread.Sleep(10);

B: 1	B: 16	B: 31	B: 15	B: 29	B: 27	B: 41	A: 39
B: 3	B: 17	B: 32	B: 16	B: 30	B: 28	B: 42	A: 40
B: 4	B: 18	B: 33	B: 17	B: 31	B: 29	B: 43	A: 41
B: 5	B: 19	B: 34	B: 18	B: 32	B: 30	B: 44	A: 42
B: 6	B: 20	B: 35	B: 19	B: 33	B: 31	B: 45	A: 43
B: 7	B: 21	B: 36	B: 20	B: 34	B: 32	B: 46	A: 44
B: 8	B: 22	B: 37	B: 21	B: 35	A: 24	B: 47	A: 45
B: 9	B: 23	B: 38	B: 22	B: 36	B: 33	B: 48	A: 46
B: 10	B: 24	B: 39	A: 14	B: 37	B: 34	B: 49	A: 47
B: 11	B: 25	B: 40	B: 23	B: 38	B: 35	B: 50	A: 48
B: 12	B: 26	B: 41	B: 24	B: 39	B: 36	A: 34	A: 49
A: 2	B: 27	B: 42	B: 25	B: 40	B: 37	A: 35	A: 50
B: 13	B: 28	B: 43	B: 26	B: 41	B: 38	A: 36	
B: 14	B: 29	B: 44	B: 27	B: 25	B: 39	A: 37	
B: 15	B: 30	B: 45	B: 28	B: 26	B: 40	A: 38	



## Semaphore

Entwickelt 1965 von E. W. Dijkstra.

Der Schwerpunkt liegt hier im Schlafen und Wecken von Prozessen, um so unnötige Prozessorvergeudung zu verhindern.

### Prinzip:

- Einführung einer Integervariablen - Semaphor.
- Operation DOWN
  - Fall **Semaphor > 0**,
    - Semaphor wird um eins erniedrigt
    - Prozess startet
  - Fall **Semaphor <= 0**, Prozess wird schlafen gelegt
- Operation UP (Semaphor wird um eins erhöht),
  - Falls **Semaphor=1**, dann wird ein Prozess aufgeweckt (Sind Prozesse vorhanden?)
  - Falls **Semaphor>1**, dann passiert nichts



## Semaphore

### Prinzipieller Ablauf:

- DOWN(P1)
- kritischer Bereich
- UP(P1)

p = 0	
P1	P2
While (true) do DOWN(p);  criticalSection() UP(p); noncriticalSection(); end	While (true) do DOWN(p);  CriticalSection() UP(p); noncriticalSection(); end



## Semaphore

### Ablauf:

- Das Semaphore  $p$  muss mit 1 initialisiert werden

$p = 1$	
P1	P2
While (true) do DOWN(p);  criticalSection() UP(p); noncriticalSection(); end	While (true) do DOWN(p);  CriticalSection() UP(p); noncriticalSection(); end



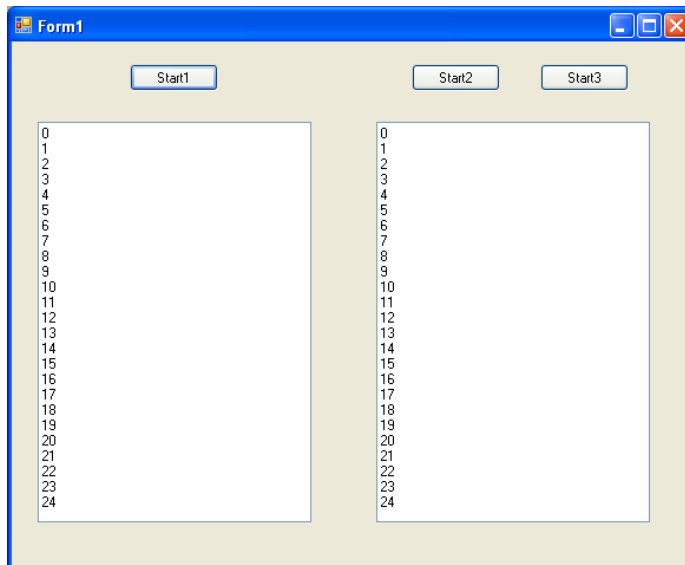
## Thread3: Schutz durch die lock-Anweisung

```
class testThread {
    int number = 0;
    private Object thisLock = new Object();
    public void run() {
        while (true) {
            int k;
            lock (thisLock)
            {
                k = number;
                k++;
                number = k;
            }
            if (number > 50) break;
            Console.WriteLine(number);
        }
    }
}
```





## Threads: 5. Beispiel, Ausgabe in zwei Editoren



### Start1

- Sequentiell

### Start2

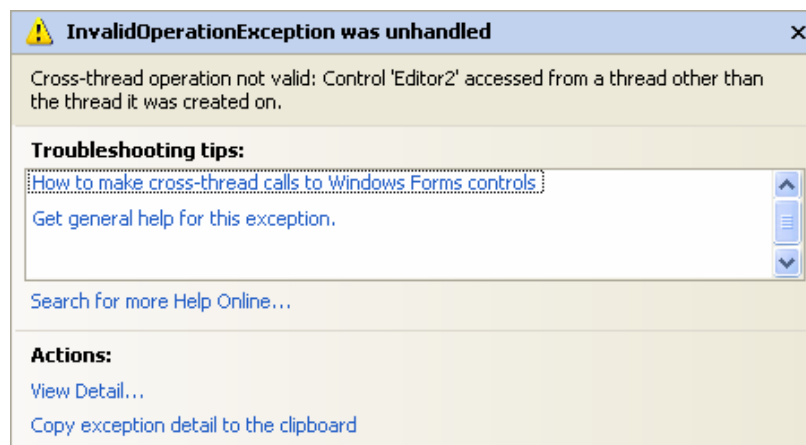
- 2 Threads
- Absturz

### Start3

- 2 Threads
- Kein Absturz
- Schließen geht nicht
- Form\_Closing



## GUI und Threads



```

private void SetText1b(string text)
{
    // Wenn das GUI-Element von einem anderem Thread, auch der
    // Thread der Form belegt ist muss das "Setzen" in eine
    // Warteschlange eingefügt werden
    if (this.Editor1.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(SetText1b);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.Editor1.Text = text;
    }
}

```



## Programm Beenden und Threads

```

private void Form1_FormClosing(object sender, FormClosingEventArgs e) {
    // sind die Threads beendet ?
    if ( (_t1 != null) && (_t1.IsAlive) )
    {
        e.Cancel = true;
        return;
    }
    if ( (_t2 != null) && (_t2.IsAlive) )
    {
        e.Cancel = true;
        return;
    }
    e.Cancel = false;
}

```



## Programm Beenden und Threads

```
private void Form1_FormClosing_Alternative(object sender, FormClosingEventArgs e)
{
    // nun die Threads beenden
    if (_t1.IsAlive)
    {
        _t1.Abort();
    }
    if (_t2.IsAlive)
    {
        _t2.Abort();
    }
    e.Cancel = false;
}
```



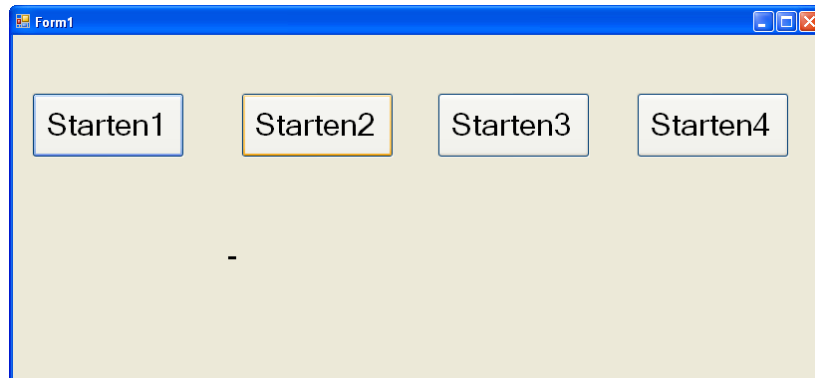
## 6. Thread-Beispiel: Bestimmen der Summe

### Aufgaben:

- Starten von drei Threads
- Jeder berechnet die Summe von 1 bis 100
- Methode getSumme() liefert die Summe
- Im Hauptdialogfenster wird die Summe angezeigt
- Summe von 1 bis 100 = 5050
- $5050 * 3 = 15150$

$$s = \sum_{i=1}^{100} i$$





- Starten1 3 Threads
- Starten2 3 Threads mit j.
- Starten3 3 Threads mit globaler Summe
- Starten4 3 Threads mit S. oder M.

```
private void BnStart1_Click(object sender, EventArgs e) {
    ThreadSumme prog1 = new ThreadSumme(100);
    ThreadSumme prog2 = new ThreadSumme(100);
    ThreadSumme prog3 = new ThreadSumme(100);
    ThreadStart ts1, ts2, ts3;
    ts1 = new ThreadStart(prog1.run);
    ts2 = new ThreadStart(prog2.run);
    ts3 = new ThreadStart(prog3.run);
    Thread t1, t2, t3;
    t1 = new Thread(ts1);
    t2 = new Thread(ts2);
    t3 = new Thread(ts3);
    t1.Start();
    t2.Start();
    t3.Start();
    int summe = prog1.getSumme() + prog2.getSumme() + prog3.getSumme();
    LErgebnis.Text = "Summe: " + summe.ToString();
}
```

```

class ThreadSumme {
    private int _n;
    private int _Summe;
    public ThreadSumme(int n) {
        _n = n;
        _Summe = 0;
    }

    public void run() {
        _Summe = 0;
        for (int i = 1; i <= _n; i++) {
            _Summe += i;
            Thread.Sleep(5);
        }
    }
    public int getSumme() {
        return _Summe;
    }
} // Thread Summe

```



```

private void BnStart2_Click(object sender, EventArgs e) {
    ThreadSumme prog1 = new ThreadSumme(100);
    ThreadSumme prog2 = new ThreadSumme(100);
    ThreadSumme prog3 = new ThreadSumme(100);
    ThreadStart ts1, ts2, ts3;
    ts1 = new ThreadStart(prog1.run); ts2 = new ThreadStart(prog2.run);
    ts3 = new ThreadStart(prog3.run);
    Thread t1, t2, t3; t1 = new Thread(ts1); t2 = new Thread(ts2); t3 = new Thread(ts3);
    t1.Start();
    t2.Start();
    t3.Start();
    t1.Join();
    t2.Join();
    t3.Join();
    int summe = prog1.getSumme() + prog2.getSumme() + prog3.getSumme();
    LErgebnis.Text = "Summe: " + summe.ToString();
}

```



```

private void BnStart3_Click(object sender, EventArgs e) {
    Integer Summe = new Integer();
    Summe.summe = 0;
    ThreadSumme3 prog1 = new ThreadSumme3(100, Summe);
    ThreadSumme3 prog2 = new ThreadSumme3(100, Summe);
    ThreadSumme3 prog3 = new ThreadSumme3(100, Summe);
    ThreadStart ts1, ts2, ts3;
    ts1 = new ThreadStart(prog1.run);          ts2 = new ThreadStart(prog2.run);
    ts3 = new ThreadStart(prog3.run);
    Thread t1, t2, t3;
    t1 = new Thread(ts1);  t2 = new Thread(ts2);  t3 = new Thread(ts3);
    t1.Start();           t2.Start();           t3.Start();

    t1.Join();           t2.Join();           t3.Join();

    LErgebnis.Text = "Summe: " + Summe.summe.ToString();
}

```

**Ergebnis: Manchmal die richtige Summe**



```

class Integer
{
    public int summe;

    public Integer()
    {
        summe=0;
    }
}

class ThreadSumme3 {
    private int _n;
    private Integer _Summe;

    public ThreadSumme3(int n, Integer Summe) {
        _n = n;
        _Summe = Summe;
    }

    public void run() {
        int s;
        for (int i = 1; i <= _n; i++)
        {
            s = _Summe.summe;
            s = s + i;
            _Summe.summe = s;
            Thread.Sleep(5);
        }
    }
} // Thread Summe3

```



```

private void BnStart4_Click(object sender, EventArgs e) {
    Integer Summe = new Integer();
    Summe.summe = 0;
    Mutex mutex = new Mutex(false, "Ein_Beispiel_Mutex");
    ThreadSumme4 prog1 = new ThreadSumme4(100,
    ThreadStart ts1, ts2, ts3;
    ts1 = new ThreadStart(prog1.run);
    Thread t1, t2, t3;
    t1 = new Thread(ts1);
    t1.Start();
    t1.Join();
    LErgebnis.Text = "Summe: " + Summe.summe.ToString();
    mutex.Close();
    mutex = null;
}

```



```

class ThreadSumme4 {
    private int _n;
    private Integer _Summe;
    Mutex _mutex;

    public ThreadSumme4(int n, Integer Summe)
    {
        _n = n;
        _Summe = Summe;
        // Benanntes Mutex
        _mutex = Mutex.OpenExisting("Ein_Beispiel_Mutex");
    }
}

```



```
// Klasse ThreadSumme4

public void run()
{
    int s;
    for (int i = 1; i <= _n; i++)
    {
        _mutex.WaitOne(); // Down
        s = _Summe.summe;
        s = s + i;
        _Summe.summe = s;
        _mutex.ReleaseMutex(); // Up
        Thread.Sleep(5);
    }
}
```



## Mutex: Konstruktoren

- Mutex()
  
- Mutex(String)
  - Benanntes Semaphore





## Mutex: Methoden

- `OpenExisting(String sName)`
  
- Down:
  - `WaitOne()`
  - `WaitOne(int32 count)`
  - `WaitAny / SignalAndWait`                      statische Methode
  
- Up:
  - `Release()`
  
- `Close()`
  
- `Dispose()`



## Semaphore: Konstruktoren

- `Semaphore(Int32 initialCount, int32 MaxCount)`
  
- `Semaphore(Int32 initialCount, int32 MaxCount, String sName)`
  - Benanntes Semaphore
  
- `Semaphore(Int32 initialCount, Int32 MaxCount, String, out Boolean createNew)`
  
- `Semaphore(Int32 initialCount, Int32 MaxCount, String, out Boolean createNew, SemaphoreSecurity)`



## Semaphore: Methoden

- `OpenExisting(String sName)`
  
- **Down:**
  - `WaitOne()`
  - `WaitOne(int32 count)`
  - `WaitAny / SignalAndWait`                      statische Methode
  
- **Up:**
  - `Release()`
  - `Release(int32 count)`
  
- `Close()`
  
- `Dispose()`



### Korrespondierendes Warten zweier Threads:

- `WaitHandle.SignalAndWait (wh1, wh2);`
- `WaitHandle.SignalAndWait (wh2, wh1);`

### `Semaphore.WaitAny( WaitHandle[] waitHandle);`

#### **Beispiel:**

- `Semaphore s1 = new Semaphore(2, 2);`
- `Semaphore s2 = new Semaphore(2, 2);`
- `// WaitHandle w;`
- `// Semaphore.WaitAny( WaitHandle[] waitHandle);`
- `Semaphore.WaitAny( new WaitHandle[] { s1,s2 } );`

