Window Presentation Foundation

- 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

- Einführung WPF
- **Layouts**
- C# Sprache
- Dialog-Elemente, Menüs
- 2D- / 3D-Grafik, Audio, Video, Animation
- Eigene Komponenten
- **Threads**
- Datenbanken
- Routet Events, Dependency Properties, Command
- Textdarstellung (Flow-FixedDocuments)

Threads

Threads erlauben eine Parallalität

- Auslastung von Multi-Core-Rechner
- laufen im gleichen Adressraum eines Prozesses

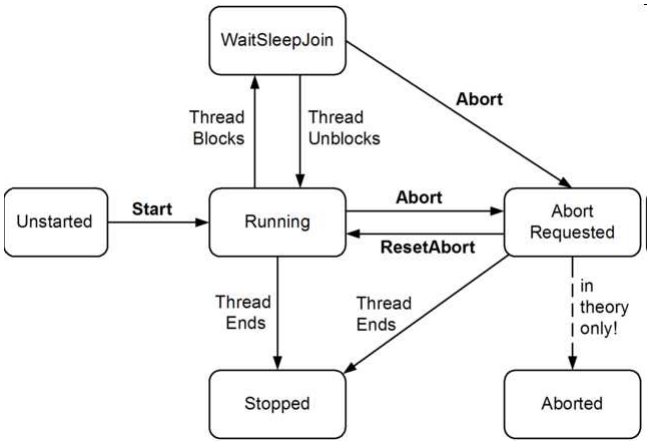
Threads werden in Java durch eigene Klassen unterstützt. Der Aufwand zur Erzeugung ist dadurch minimal.

- Ableiten der Thread, Methode run

Threads in .net

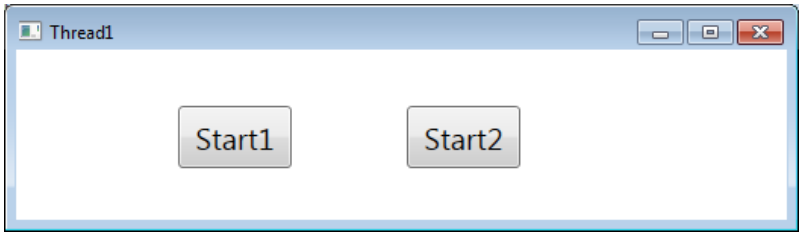
- Verwendet die Delegate-Technik, Methode public void beliebigername();
- Erzeugen einer ThreadStart Instanz
- Erzeugen eines Threads
- Starten des Threads

Zustände von Threads



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

Threads: 1. Beispiel (Console)



Button „Start1"

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

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

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

```
void bnStart1_Click(object sender, RoutedEventArgs e)
{
        Ausgabe1();
}
```

Ansicht: Ausgabe

1
2
...
97
98
99
100
1001
1002
...
1096
1097
1098
1099
1100

Ergebnis:

- Die Ausgaben der beiden Threads werden hintereinander ausgegeben

Threads-Beispiel

```
public void run1()   {
}

public void run2()   {
}

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();
        System.Diagnostics.Debug.WriteLine("Fertig");
}
```

run1,2 siehe Folie vorher

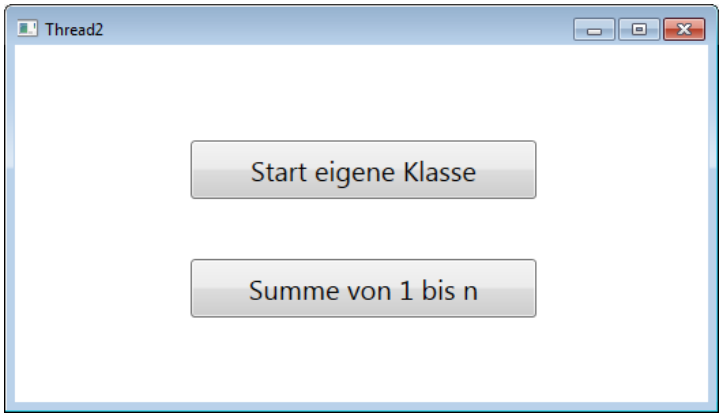
Ansicht: Ausgabe

1001
1
Fertig
2
3
1002
4
5
1003
6
7
8
1004
9
10
11
1005
12

Ergebnis:

- Die Ausgaben der beiden Threads werden ineinander ausgegeben

Threads: 2. Beispiel, eigene Klasse



Threads: 2. Beispiel, eigene Klasse, gemeinsame Variable

```
using System.Threading;

class testThread   {
      int number = 0;
      public void run1()
      {
         while (true)
         {
            int k;
            k = number;
            k++;
            number = k;
            if (number > 50) break;
            System.Diagnostics.Debug.Console.WriteLine("A: "+ number);
         }
      }
      public void run2()  {}
 }
```


Threads: 2. WPF-Beispiel, eigene Klasse, 2x run

```
using System.Threading;

private void bnStart1_Click(object sender, RoutedEventArgs e) {
         TestThread prog = new TestThread();

         ThreadStart ts1, ts2;
         ts1 = new ThreadStart(prog.run1);
         // ts2 = new ThreadStart(prog.run2);

         Thread t1, t2;
         t1 = new Thread( ts1 );
         t2 = new Thread( prog.run2 );  // Vereinfachte Übergabe
         t1.Start();
         t2.Start();

            }
```

Ansicht: Ausgabe

A: 1
B: 2
A: 2
A: 3
B: 3
A: 4
B: 4
A: 5
B: 5
A: 6
B: 6
A: 7
B: 7
A: 8
B: 8
A: 9
B: 9

Ergebnis:

- Die Ausgaben der beiden Threads werden ineinander ausgegeben
- Die Nummern sind unabhängig
- static

2. Schalter: Berechne Summe: Klasse ThreadSumme

```
class ThreadSumme {
    public static int summe = 0;

    private int start;
    private int ende;

    public ThreadSumme(int a, int b) {
        start = a;
        ende = b;
    }

    public void run() {
        for (int i=start; i<=ende; i++) {
            int s=summe;
            Thread.Sleep(10);
            summe = s + i;
        }
    }
}
```

Aufruf der Klasse ThreadSumme

```
private void bnSumme_Click(object sender, RoutedEventArgs e) {
        int summe = 0;
        for (int i = 1; i <= 1000; i++) summe += i;
        MessageBox.Show("Ergebnis: " + summe);

        ThreadSumme prog1 = new ThreadSumme(1,500);
        ThreadSumme prog2 = new ThreadSumme(501,1000);

        Thread t1, t2;
        t1 = new Thread(prog1.run);
        t2 = new Thread( prog2.run );  // Vereinfachte Übergabe
        t1.Start();
        t2.Start();
        MessageBox.Show("Ergebnis: " + ThreadSumme.summe);
}
```


Aufruf der Klasse ThreadSumme

```
private void bnSumme_Click(object sender, RoutedEventArgs e) {
        int summe = 0;
        for (int i = 1; i <= 1000; i++) summe += i;
        MessageBox.Show("Ergebnis: " + summe);

        ThreadSumme prog1 = new ThreadSumme(1,500);
        ThreadSumme prog2 = new ThreadSumme(501,1000);

        Thread t1, t2;
        t1 = new Thread(prog1.run);
        t2 = new Thread( prog2.run );  // Vereinfachte Übergabe
        t1.Start();
        t2.Start();
        t1.Join();
        t2.Join();
        MessageBox.Show("Ergebnis: " + ThreadSumme.summe);
}
```

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 Semaphor 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

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;
         }
      }
   }
```

3. Schalter: Berechne Summe: Klasse ThreadSumme2

```
class ThreadSumme2 {
    public static int summe = 0;
    private static Object thisLock = new Object();
    private int start, ende;

    public ThreadSumme2(int a, int b) {
      start = a;
      ende = b;
    }
    public void run() {
      for (int i=start; i<=ende; i++) {
        lock (thisLock) {
          int s = summe;
          Thread.Sleep(10);
          summe = s + i;
        }
      }
    }
  }
```

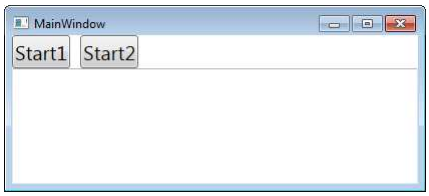

3. Schalter: Berechne Summe: Aufruf

```
private void bnSumme2_Click(object sender, RoutedEventArgs e) {
    int summe = 0;
    for (int i = 1; i <= 1000; i++) summe += i;

    ThreadSumme2 prog1 = new ThreadSumme2(1, 500);
    ThreadSumme2 prog2 = new ThreadSumme2(501, 1000);

    Thread t1, t2;
    t1 = new Thread(prog1.run);
    t2 = new Thread(prog2.run);  // Vereinfachte Übergabe
    ThreadSumme2.summe = 0;
    t1.Start();
    t2.Start();
    t1.Join();
    t2.Join();
    MessageBox.Show("Ergebnis summe: " + summe + "\r\n"
        + "Ergebnis Thread: " + ThreadSumme2.summe);
  }
```

Threads: mit grafischen Elementen: Thread3




```
…
Title="MainWindow" Height="350" Width="525" FontSize="20">
  <DockPanel LastChildFill="True">
    <WrapPanel DockPanel.Dock="Top">
      <Button Name="bnStart1" Content="Start1" Click="bnStart1_Click" />
      <Button Name="bnStart2" Content="Start2" Margin="10 0 0 0"
                                                Click="bnStart2_Click" />
    </WrapPanel>
    <TextBox DockPanel.Dock="Top" Name="editor" AcceptsReturn="True" />
  </DockPanel>
</Window>
```



```
class TestThread1 {
    private int start;
    private int ende;
    private TextBox editor;
    private String bez;

    public TestThread1(String bez, int a, int b, TextBox editor) {
      start = a;
      ende = b;
      this.bez = bez;
      this.editor = editor;
    }

    public void run() {
      for (int i=start; i<=ende; i++) {
        editor.AppendText(bez + i + "\r\n");
        Thread.Sleep(10);
      }
    }
  }
```

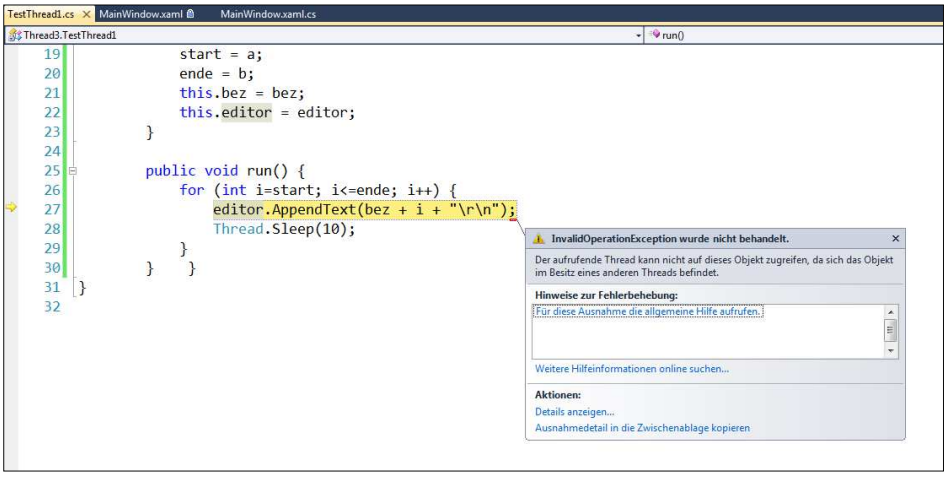
Aufruf der Threads mit Editor

private void bnStart1_Click(object sender, RoutedEventArgs e) {

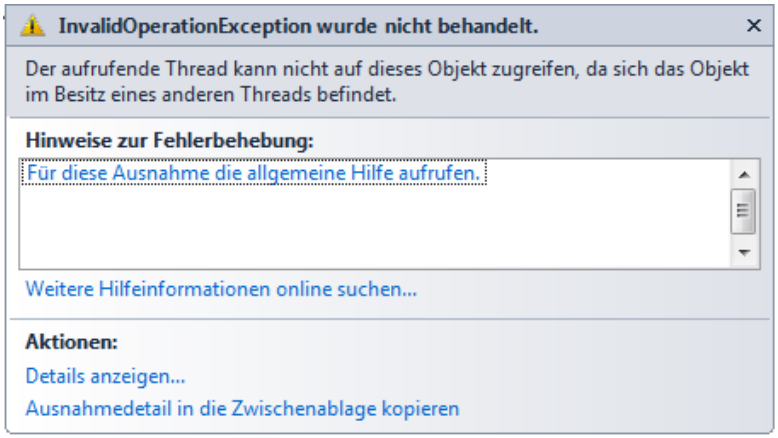
```
 TestThread1 prog1 = new TestThread1("1. Thread", 1, 500, editor);
 TestThread1 prog2 = new TestThread1("2. Thread", 1, 500, editor);

  Thread t1, t2;
   t1 = new Thread(prog1.run);
   t2 = new Thread(prog2.run);  // Vereinfachte Übergabe
   t1.Start();
   t2.Start();
}
```


Aufruf der Threads mit Editor



GUI und Threads



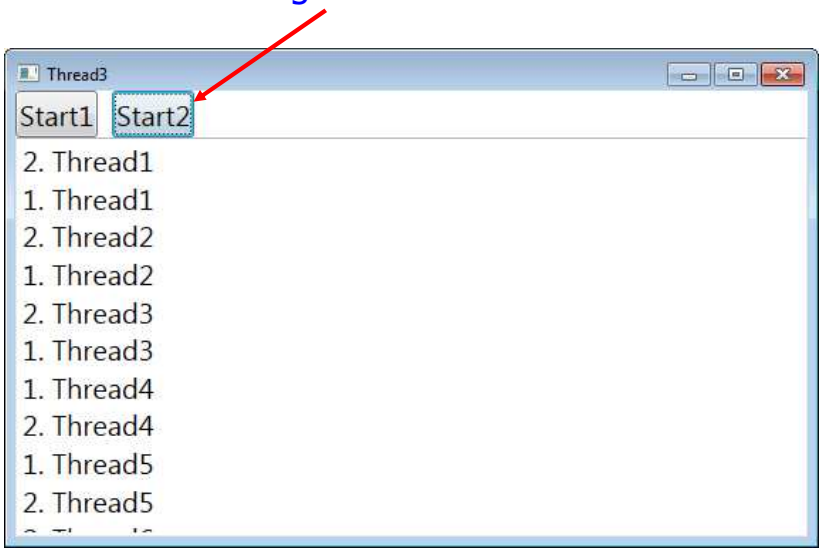

```
class TestThread2 {
    …
    delegate void appendEditorCallback(String item);

    private void add(String item) {
        Application.Current.Dispatcher.Invoke(
                new Action( () => { appendEditor(item); } )
        );
    }

    public void appendEditor(String item) {
        editor.AppendText(item);
    }

    public void run() {
        for (int i=start; i<=ende; i++) {
            add(bez + i + "\r\n");
            Thread.Sleep(10);
        }
    }
}
```

GUI und Threads: gelöst



Beispiel-Code:

```
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, Summe);
        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;
}
```

Benutzung des Mutex (1):

```
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");
    }
```


Benutzung des Mutex (2):

```
// 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 });

Thread2: Thread Summe Mutex

```
class ThreadSummeMutex {
    public static int summe = 0;
    private int start, ende;
    private Mutex mutex;

    public ThreadSummeMutex(int a, int b) {
      start = a;
      ende = b;
      mutex = Mutex.OpenExisting("Thread_Summe_Mutex");
    }

    public void run() {
      for (int i=start; i<=ende; i++) {
        mutex.WaitOne();   // Down
        int s = summe;
        Thread.Sleep(10);
        summe = s + i;
        mutex.ReleaseMutex();  // Up
      }
    }
  }
```

Thread2: Thread Summe Mutex: Aufruf

```
private void bnMutex_Click(object sender, RoutedEventArgs e) {
        Mutex mutex = new Mutex(false, "Thread_Summe_Mutex");

        ThreadSummeMutex prog1 = new ThreadSummeMutex(1, 500);
        ThreadSummeMutex prog2 = new ThreadSummeMutex(501, 1000);

        Thread t1, t2;
        t1 = new Thread(prog1.run);
        t2 = new Thread(prog2.run);  // Vereinfachte Übergabe
        ThreadSummeMutex.summe = 0;
        t1.Start();
        t2.Start();
        t1.Join();
        t2.Join();
        MessageBox.Show("Ergebnis Thread: " + ThreadSummeMutex.summe);
        mutex.Close();
        mutex = null;
    }
```


Thread2: Thread Summe Semaphore

```
class ThreadSummeSemaphore {
    public static int summe = 0;
    private int start, ende;
    private Semaphore crit;

    public ThreadSummeSemaphore(int a, int b) {
        start = a;
        ende = b;
        crit = Semaphore.OpenExisting("Thread_Summe_Semaphore");
    }

    public void run() {
        for (int i = start; i <= ende; i++) {
            crit.WaitOne();   // Down
            int s = summe;
            Thread.Sleep(10);
            summe = s + i;
            crit.Release();  // Up
        }
    }
}
```

Thread2: Thread Summe Semaphore: Aufruf

```
private void bnSemaphore_Click(object sender, RoutedEventArgs e) {
        Semaphore crit = new Semaphore(1, 1, "Thread_Summe_Semaphore");
        ThreadSummeSemaphore prog1 = new ThreadSummeSemaphore(1, 500);
        ThreadSummeSemaphore prog2 = new ThreadSummeSemaphore(501, 1000);

        Thread t1, t2;
        t1 = new Thread(prog1.run);
        t2 = new Thread(prog2.run);  // Vereinfachte Übergabe
        ThreadSummeSemaphore.summe = 0;
        t1.Start();
        t2.Start();
        t1.Join();
        t2.Join();
        MessageBox.Show("Ergebnis Thread: " + ThreadSummeSemaphore.summe);
        crit.Close();
        crit = null;
    }
```