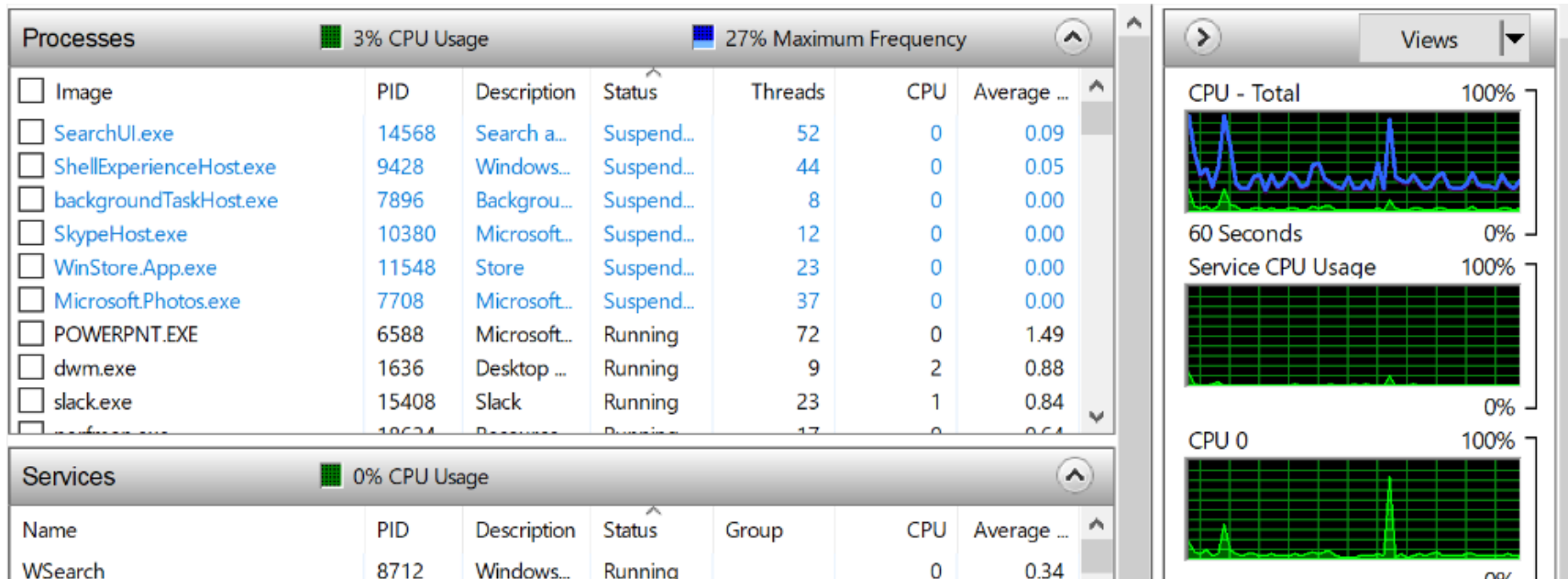

05 – Multithreading

Agenda

- Einführung
- Grundlagen
- Threads in .NET
- Klasse Backgroundworker
- (Task Parallel Library TPL)
- (async await)

Warum Applikationen mit parallelen Prozessen?

- «Responsive UI» bauen
- Langsame I/O-Operationen auslagern
- Mehrkern-Prozessoren nutzen



Herausforderungen

- Applikationen mit parallelen Prozessen schreiben ist schwierig
- Lesbarkeit kann schlechter werden
- Wartbarkeit kann schlechter werden
- Fehleranalyse ist schwieriger
- Testbarkeit kann schwierig bis unmöglich werden
- Deadlocks können auftreten

Multitasking

- Fähigkeit eines Betriebssystems mehrere Anwendungen quasi gleichzeitig auszuführen. Das Betriebssystem stellt den Anwendungen die notwendigen Ressourcen zur Verfügung.
- Cooperative Multitasking
 - Das laufende Programm darf den Prozessor so lange belegen, bis es diesen freiwillig abgibt. Kooperiert das laufende Programm nicht, so werden die anderen Programme nicht ausgeführt.
Es ist in modernen PC-Betriebssystemen nicht mehr üblich.
- Preemptive Multitasking
 - Das laufende Programm kann vom Betriebssystem gezwungen werden den Prozessor abzugeben und ein anderes Programm erhält das Recht den Prozessor zu belegen.
 - Die Verwaltung gemeinsamer Ressourcen führt zum Problem. Die Synchronisation von Ressourcen ist notwendig.

Prozess

- Ein Prozess besteht aus dem Speicherbereich und den notwendigen Ressourcen. Der Speicherbereich ist isoliert von anderen Prozessen.

Thread

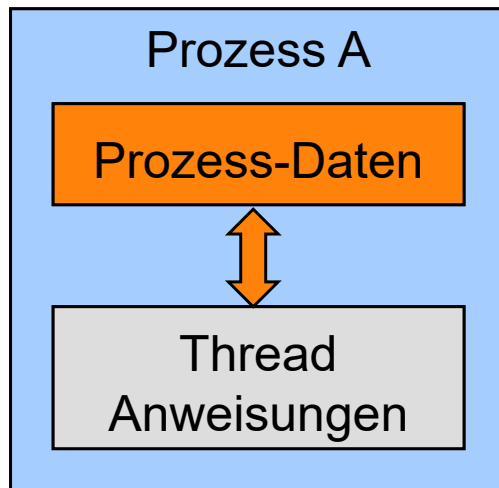
- Ein Prozess kann aus einem oder mehreren Threads bestehen

Ein Thread besteht aus:

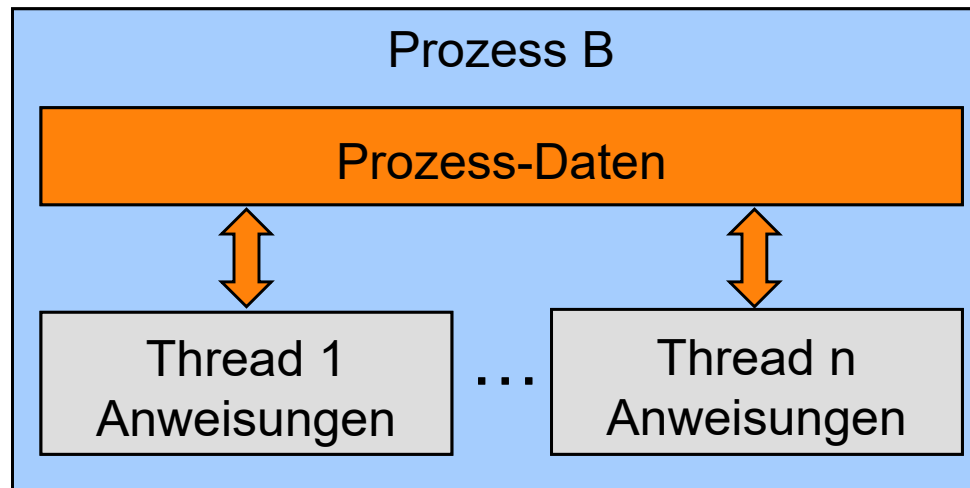
- Anweisungen
- Stack und Speicherbereich, um Thread-Zustand zu verwalten
- Hat Zugriff auf die Daten des Prozesses

Prozesse und Threads

Prozess mit einem Thread



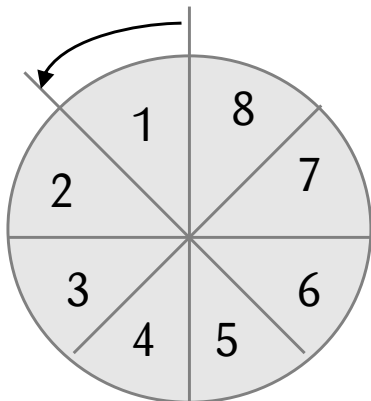
Prozess mit mehreren Threads



Zeitscheiben (Time Slices)

- Das Betriebssystem besitzt einen Scheduler. Dieser gibt einem Prozess eine bestimmte Menge an Zeit, ehe ein anderer Prozess an die Reihe kommt.
- Diese Zeit kann vom Programmierer nicht beeinflusst werden.
- Die Dauer kann von Betriebssystem zu Betriebssystem und von Prozessor zu Prozessor unterschiedlich sein.

Zeitscheibe: 20ms



Bsp. zur Ausführung der Threads

- 1: Prozess A, Thread 1
- 2: Prozess B, Thread 1
- 3: Prozess B, Thread 2
- 4: Prozess C, Thread 1
- 5: Prozess A, Thread 2
- 6: Prozess C, Thread 2
- 7: Prozess D, Thread 1
- 8: Prozess B, Thread 3

Prioritäten

- Prozesse / Threads mit höherer Priorität haben Vorrang vor Prozessen mit niedrigerer Priorität
- Bei gleicher Priorität kommen die Prozesse in einem Round-Robin-Verfahren, d.h. abwechselnd, an die Reihe
- Threads erben die Priorität von zugehörigen Prozess. Jeder Thread kann jedoch seine Priorität selbst setzen.

Threads in .NET

Namespaces

```
using System.Threading;
```

Die Klasse Thread

■ Konstruktor:

```
public Thread(ThreadStart start);
```

■ Delegate ThreadStart:

```
public delegate void ThreadStart();
```

Thread-Design

- Die Thread-Methode enthält typischerweise eine Schleife, welche die auszuführende Thread-Arbeit enthält

```
public void MyThreadMethod()  
{  
    while( <Bedingung> )  
    {  
        <Die Arbeit für den Thread>  
    }  
}
```

- Methode kann static sein
- Wird oft als Worker-Methode bezeichnet

Threads in .NET

Codebeispiel - Thread instanziiieren und starten:

```
using System;
using System.Threading;

class Test
{
    static void Main()
    {
        // Thread-Instanz erzeugen und starten
        Thread newThread =
            new Thread(new ThreadStart(Work.DoWork));
        newThread.Start();
    }
}

class Work
{
    Work() {}

    public static void DoWork() { ... } // Thread Methode
}
```

Threads in .NET

Codebeispiel

```
namespace ThreadsBasic
{
    class Program
    {
        static void Main(string[] args)
        {
            // Thread erzeugen und starten
            Thread t = new Thread(WriteSlash);
            t.Start();

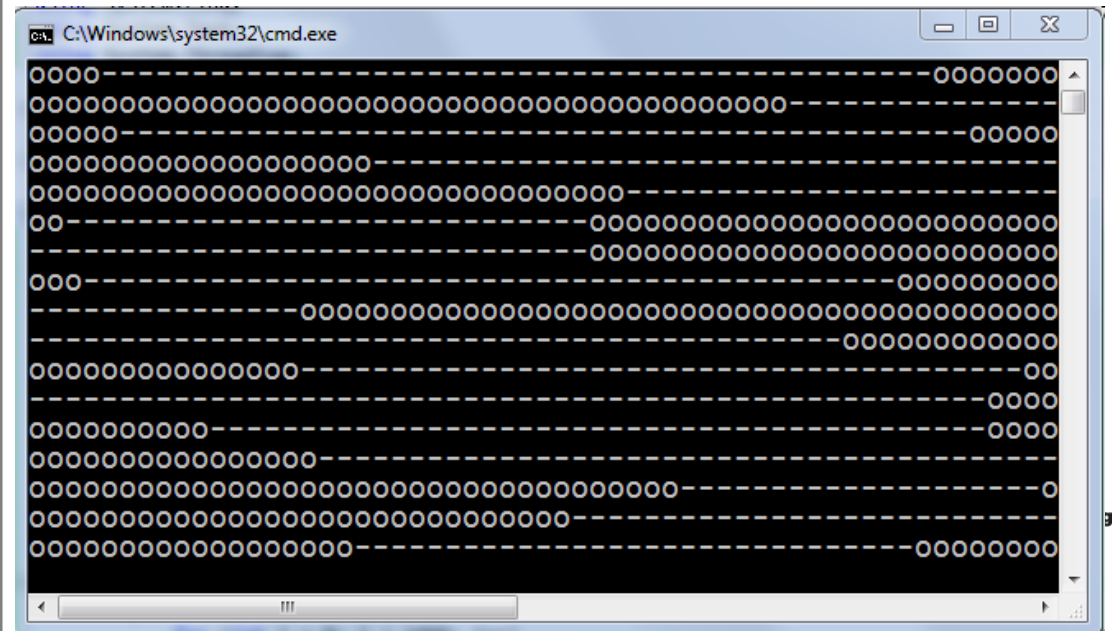
            // Simultan im main thread "o" ausgeben.
            for (int i = 0; i < 1000; i++)
            {
                Console.Write("o");
            }
        }

        // thread worker methode
        static void WriteSlash()
        {
            for (int i = 0; i < 1000; i++)
            {
                Console.Write("-");
            }
        }
    }
}
```

Main-Thread

Statische
Worker-Methode

Ausführung in
zweitem Thread



Folie 13

Übung 5.1



Zwei Threads Incrementer / Decrementer

A) Implementiere eine Klasse `Test` mit zwei Worker-Methoden :

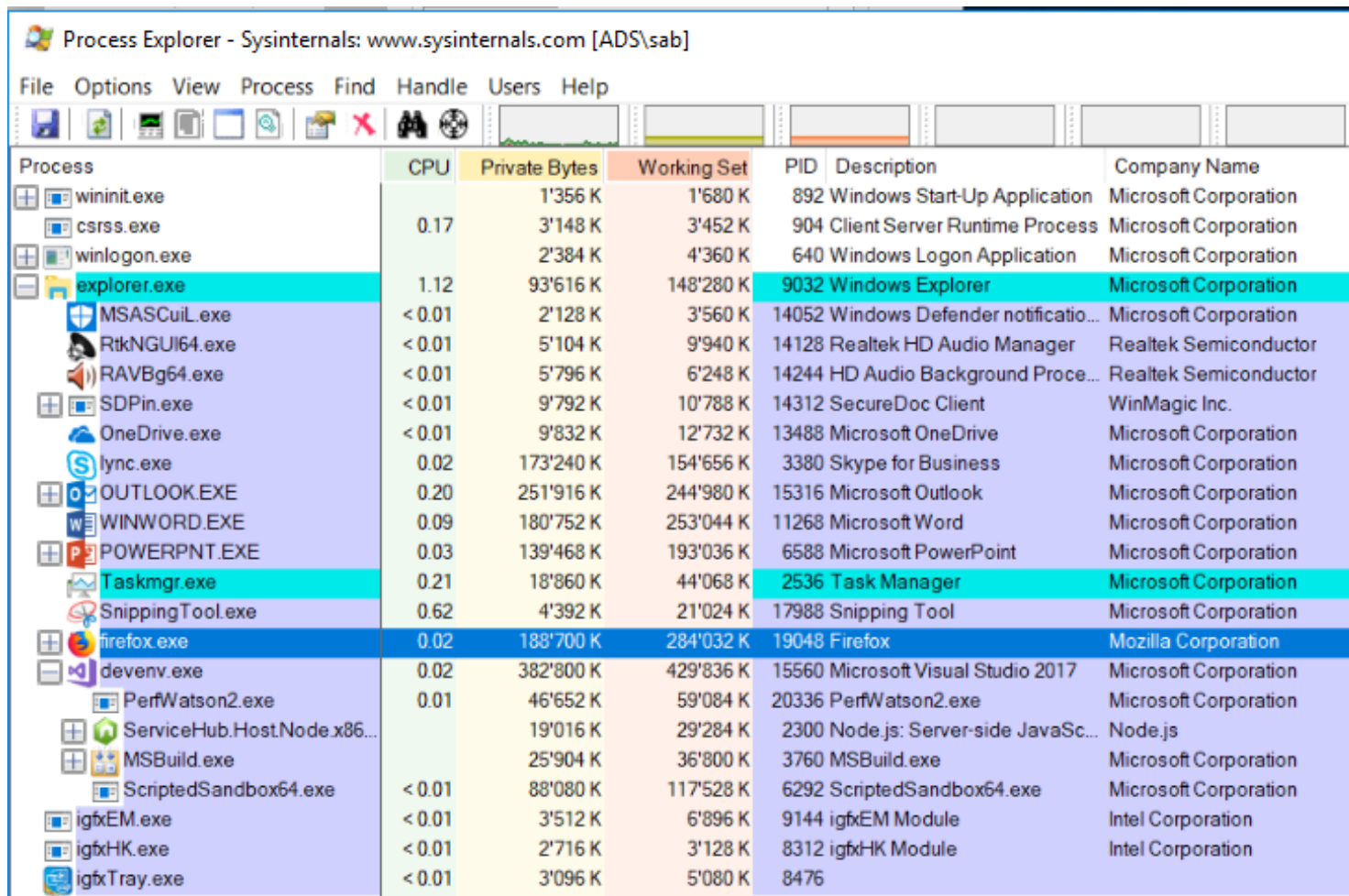
- `Increment()` // 1...1000 auf Konsole schreiben
- `Decrement()` // 1000...1 auf Konsole schreiben

B) Starte die beiden Methoden in zwei separaten Threads

Threads in .NET

Process Explorer von Sysinternals

■ Prozesse und Threads visualisieren



The screenshot shows the Process Explorer application window. The title bar reads 'Process Explorer - Sysinternals: www.sysinternals.com [ADS\sab]'. The menu bar includes 'File', 'Options', 'View', 'Process', 'Find', 'Handle', 'Users', and 'Help'. The toolbar contains icons for file operations and process management. The main window displays a table of running processes. The table has columns for 'Process', 'CPU', 'Private Bytes', 'Working Set', 'PID', 'Description', and 'Company Name'. The 'Process' column shows a tree view of processes, with 'explorer.exe' and 'Taskmgr.exe' highlighted in blue. The 'CPU' column shows values like 0.17, 1.12, and 0.02. The 'Private Bytes' column shows values like 3'148 K, 93'616 K, and 188'700 K. The 'Working Set' column shows values like 3'452 K, 148'280 K, and 284'032 K. The 'PID' column shows values like 904, 640, and 9032. The 'Description' column shows descriptions like 'Client Server Runtime Process', 'Windows Logon Application', and 'Windows Explorer'. The 'Company Name' column shows names like 'Microsoft Corporation', 'Realtek Semiconductor', and 'Mozilla Corporation'.

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
wininit.exe		1'356 K	1'680 K	892	Windows Start-Up Application	Microsoft Corporation
csrss.exe	0.17	3'148 K	3'452 K	904	Client Server Runtime Process	Microsoft Corporation
winlogon.exe		2'384 K	4'360 K	640	Windows Logon Application	Microsoft Corporation
explorer.exe	1.12	93'616 K	148'280 K	9032	Windows Explorer	Microsoft Corporation
MSASCuiL.exe	< 0.01	2'128 K	3'560 K	14052	Windows Defender notificatio...	Microsoft Corporation
RtkNGUI64.exe	< 0.01	5'104 K	9'940 K	14128	Realtek HD Audio Manager	Realtek Semiconductor
RAVBg64.exe	< 0.01	5'796 K	6'248 K	14244	HD Audio Background Proce...	Realtek Semiconductor
SDPin.exe	< 0.01	9'792 K	10'788 K	14312	SecureDoc Client	WinMagic Inc.
OneDrive.exe	< 0.01	9'832 K	12'732 K	13488	Microsoft OneDrive	Microsoft Corporation
lync.exe	0.02	173'240 K	154'656 K	3380	Skype for Business	Microsoft Corporation
OUTLOOK.EXE	0.20	251'916 K	244'980 K	15316	Microsoft Outlook	Microsoft Corporation
WINWORD.EXE	0.09	180'752 K	253'044 K	11268	Microsoft Word	Microsoft Corporation
POWERPNT.EXE	0.03	139'468 K	193'036 K	6588	Microsoft PowerPoint	Microsoft Corporation
Taskmgr.exe	0.21	18'860 K	44'068 K	2536	Task Manager	Microsoft Corporation
SnippingTool.exe	0.62	4'392 K	21'024 K	17988	Snipping Tool	Microsoft Corporation
firefox.exe	0.02	188'700 K	284'032 K	19048	Firefox	Mozilla Corporation
devenv.exe	0.02	382'800 K	429'836 K	15560	Microsoft Visual Studio 2017	Microsoft Corporation
PerfWatson2.exe	0.01	46'652 K	59'084 K	20336	PerfWatson2.exe	Microsoft Corporation
ServiceHub.HostNode.x86...		19'016 K	29'284 K	2300	Node.js: Server-side JavaSc...	Node.js
MSBuild.exe		25'904 K	36'800 K	3760	MSBuild.exe	Microsoft Corporation
ScriptedSandbox64.exe	< 0.01	88'080 K	117'528 K	6292	ScriptedSandbox64.exe	Microsoft Corporation
igfxEM.exe	< 0.01	3'512 K	6'896 K	9144	igfxEM Module	Intel Corporation
igfxHK.exe	< 0.01	2'716 K	3'128 K	8312	igfxHK Module	Intel Corporation
igfxTray.exe	< 0.01	3'096 K	5'080 K	8476		

Threads in .NET

Thread-IDs

1. Thread-ID innerhalb des Prozesses:

- Eindeutige Thread-ID innerhalb des Prozesses

```
Thread currentThread = Thread.CurrentThread;  
int threadID = currentThread.GetHashCode();
```

2. Eindeutige ID des verwalteten Threads:

```
Property: Thread.ManagedThreadId;
```

- Beispiel: Thread-ID des aktuellen Threads ermitteln

```
int threadID = Thread.CurrentThread.ManagedThreadId;
```


Ausführung pausieren

- Ein Thread kann mit der Methode `Thread.Sleep()` «schlafen gelegt» werden
- Mit `Sleep()` kann der Thread nur sich selbst schlafen legen, womit er den Prozessor von sich aus wieder frei gibt

```
public static void Sleep(int millisecondsTimeout);  
public static void Sleep(TimeSpan timeout);
```

Threads in .NET

Threads debuggen (1/3)

The screenshot shows the Visual Studio IDE with the following components:

- Source Code:** `Class1.cs` showing the `IncrementerSuspended3()` method. A breakpoint is set on the `Thread.Sleep(1);` line. The `for` loop variable `inx` is highlighted in yellow.
- Solution Explorer:** Shows the project `Threads1` with files `App.ico`, `AssemblyInfo.cs`, and `Class1.cs`.
- Autos Window:** Displays local variables: `Thread.CurrentThread` (Type: `System.Threading.Thread`), `Thread.CurrentThread.Name` (Value: `"IncThread"`), `inx` (Value: `1`), and `this` (Type: `Threads1.ThreadTest`).
- Threads Window:** Lists active threads:

ID	Name	Location	Priority	Suspend
2196	<No Name>			0
288	IncThread	Threads1.ThreadTest.Increm	Normal	0
2544	DecThread	Threads1.ThreadTest.Decrem	Normal	0
- Console Output:** Shows the output of `Console.WriteLine`:

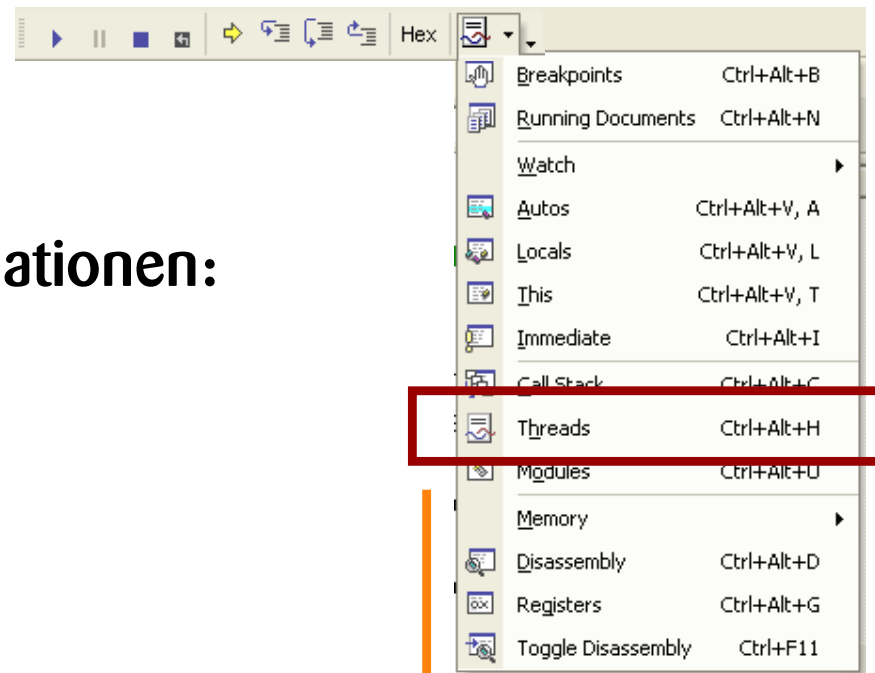
```
Incrementer {0}: {1}
Decrementer {0}: {1}
```

Threads in .NET

Threads debuggen (2/3) - Das «Threads»-Debug-Fenster

Enthält aktuelle Informationen:

- Thread ID
- Name
- Location
- Priority
- Suspend



Der Menüeintrag «Threads» ist nur sichtbar, wenn man sich im Debug-Modus befindet

Threads					
ID	Name	Location	Priority	Suspend	
2196	<No Name>			0	
288	IncThread	Threads1.ThreadTest.Increm	Normal	0	
2544	DecThread	Threads1.ThreadTest.Decrem	Normal	0	

Threads in .NET

Threads debuggen (3/3) – Threads suspendieren / aktivieren

The image illustrates the process of suspending and then activating a thread during debugging in Visual Studio.

Suspendieren (Suspending):

- The **Threads** window shows three threads: ID 924 (<No Name>), ID 3560 (IncThread), and ID 3160 (DecThread).
- A context menu is opened for the selected thread (ID 3560), and the **Freeze** option is highlighted.
- An arrow points from the **Freeze** option to the **Suspendieren** label.
- The **Threads** window is shown again, where the **Suspend** column for thread ID 3560 has changed from 0 to 1.
- An arrow points from the **Suspendieren** label to a console window.
- The console window shows the output of a program with two threads: an incrementer (ID 3560) and a decrementser (ID 3160). The output shows the incrementer's value increasing from 11 to 12, while the decrementser's value remains at 970.

Aktivieren (Activating):

- The **Threads** window is shown again, where the **Suspend** column for thread ID 3560 has changed from 1 back to 0.
- A context menu is opened for the selected thread (ID 3560), and the **Thaw** option is highlighted.
- An arrow points from the **Thaw** option to the **Aktivieren** label.

Übung 5.2



Worker-Methoden erweitern

- A) Erweitere die Aufgabe 5.1, so dass die beiden Threads nach jeder Ausgabe auf die Konsole den Prozessor wieder frei geben.
- B) Ordne jedem Thread einen Namen zu.
- C) Die Thread-ID soll mit ausgegeben werden.
- D) Benutze den Debugger und das «Thread»-Debug-Fenster, um die Threads zu debuggen.

Möglichkeiten, um Parameter zu übergeben

- Die Parameterübergabe kann mittels der Definition von Eigenschaften in der Klasse erfolgen, welche die Thread-Methode enthält
- Durch Konstruktor und Delegate mit Parametern

```
public delegate void ParametrizedThreadStart(object obj);
```

- Ausführen eines Lambda-Ausdruckes, welches die gewünschte Methode mit den Argumenten aufruft

Threads in .NET

Parameter übergeben

- Beispiel mit `ParameterizedThreadStart (object obj)`

```
namespace StartThreadParameter
{
    class Program
    {
        static void Main(string[] args)
        {
            Thread t = new Thread(PrintMessage);
            t.Start("Ich bin ein Thread mit Parametern");
        }

        static void PrintMessage(object msgObj)
        {
            string message = (string)msgObj;    // cast
            Console.WriteLine(message);
        }
    }
}
```

Threads in .NET

Parameter übergeben

■ Beispiel mit Lambda-Ausdruck

```
class Program
{
    static void Main(string[] args)
    {
        Thread t = new Thread( () => PrintMessage("Hallo") );
        t.Start();
    }

    static void PrintMessage(string msg)
    {
        Console.WriteLine(msg);
    }
}
```


Übung 5.3



Worker-Methode mit Parameterübergabe

- Erweitere die Aufgabe 5.2 so, dass Minimalwert und Maximalwert der Schleifen für beide Threads vor dem Start gesetzt werden können.

Threads in .NET

Wichtige Thread-Methoden (nicht vollständig)

Methode	Zweck / Anmerkungen
Abort()	Übliche Methode zur Beendigung eines Threads. Löst <code>ThreadAbortException</code> aus.
Interrupt()	Überführt einen anderen Thread, der sich im <code>WaitSleepJoin</code> -Zustand befindet, in den <code>Started</code> -Zustand zurück.
Join()	Blockiert den aufrufenden Thread bis zum Beenden des angegebenen Threads.
Resume()	Nimmt die Ausführung eines anderen, angehaltenen Threads wieder auf.
Sleep (int milliseconds)	Statische Methode! Blockiert den aufrufenden Thread für die angegebene Dauer in Millisekunden. Ist die Dauer 0, wird die Durchführung anderer, wartender Threads gewährleistet. Ist die Dauer <code>Infinite</code> , wird der Thread auf unbestimmte Zeit blockiert.
Start()	Ein anderer Thread wird auf den <code>Running</code> -Zustand gesetzt.
Suspend()	[Obsolete] Hält einen anderen oder den eigenen Thread an.

Threads terminieren (1/2)

- Mit der Methode `Abort()` kann ein Thread zur Beendigung aufgefordert werden

```
// Beispiel: threadB soll beendet werden:  
  
threadB.Abort();
```

- Die Methode `Abort()` wirft dem zu beendenden Thread eine Exception vom Typ `ThreadAbortException` zu

Threads terminieren (2/2)

- Der zu beendende Thread kann die Exception fangen und im catch-Block abhandeln, resp. aufräumen und sauber beenden

```
// Beispiel: Catch ThreadAbortException:  
void MyThreadMethod()  
{  
    try  
    {  
        while (<Bedingung>)  
        {  
            <Arbeit>  
        }  
    }  
    catch (ThreadAbortException exception)  
    {  
        <Aufräumarbeit>  
    }  
}
```

Übung 5.4



Threads terminieren

- Erweitere die Aufgabe 5.3 so, dass der eine Thread den anderen nach einer gewissen Zeit terminiert.

Threads Joining

- Mit der Methode `Join()` kann ein Thread blockiert werden bis ein bestimmter Thread terminiert. Der blockierte Thread läuft erst weiter, nachdem der angegebene Thread beendet ist.

```
// Beispiel: warten bis threadB beendet ist:  
threadB.Join();
```

Übung 5.5



Threads Incrementer / Decrementer Erweitert 4. Stufe

- Erweitere die Aufgabe 5.4 so, dass der eine Thread nach einer gewissen Zeit (Zählerstand) blockiert, bis der andere terminiert worden ist.

Threads in .NET

Thread-Prioritäten

```
// public enum ThreadPriority:  
  
    Highest  
    AboveNormal  
    Normal  
    BelowNormal  
    Lowest
```

- Wichtig: Die Thread Priorität muss vom Betriebssystem nicht gezwungenermassen befolgt werden!

Threads in .NET

Thread-Zustände

Unstarted:

- Ein Thread beginnt seine Lebensdauer durch den Aufruf seines Konstruktors. Er befindet sich dann im Zustand `Unstarted`.

Started:

- Durch die Thread-Methode `Start()` wird er in den Zustand `Started` versetzt. In diesem Zustand ist der Thread ablaufbereit, er besitzt jedoch noch nicht die wichtigste Ressource dazu, nämlich den Prozessor selbst.

Running:

- Der Prozessor selbst wird durch das Betriebssystem dem Thread zugeteilt. Der Thread befindet sich dann im Zustand `Running`.

Thread-Zustände - Running (Fortsetzung):

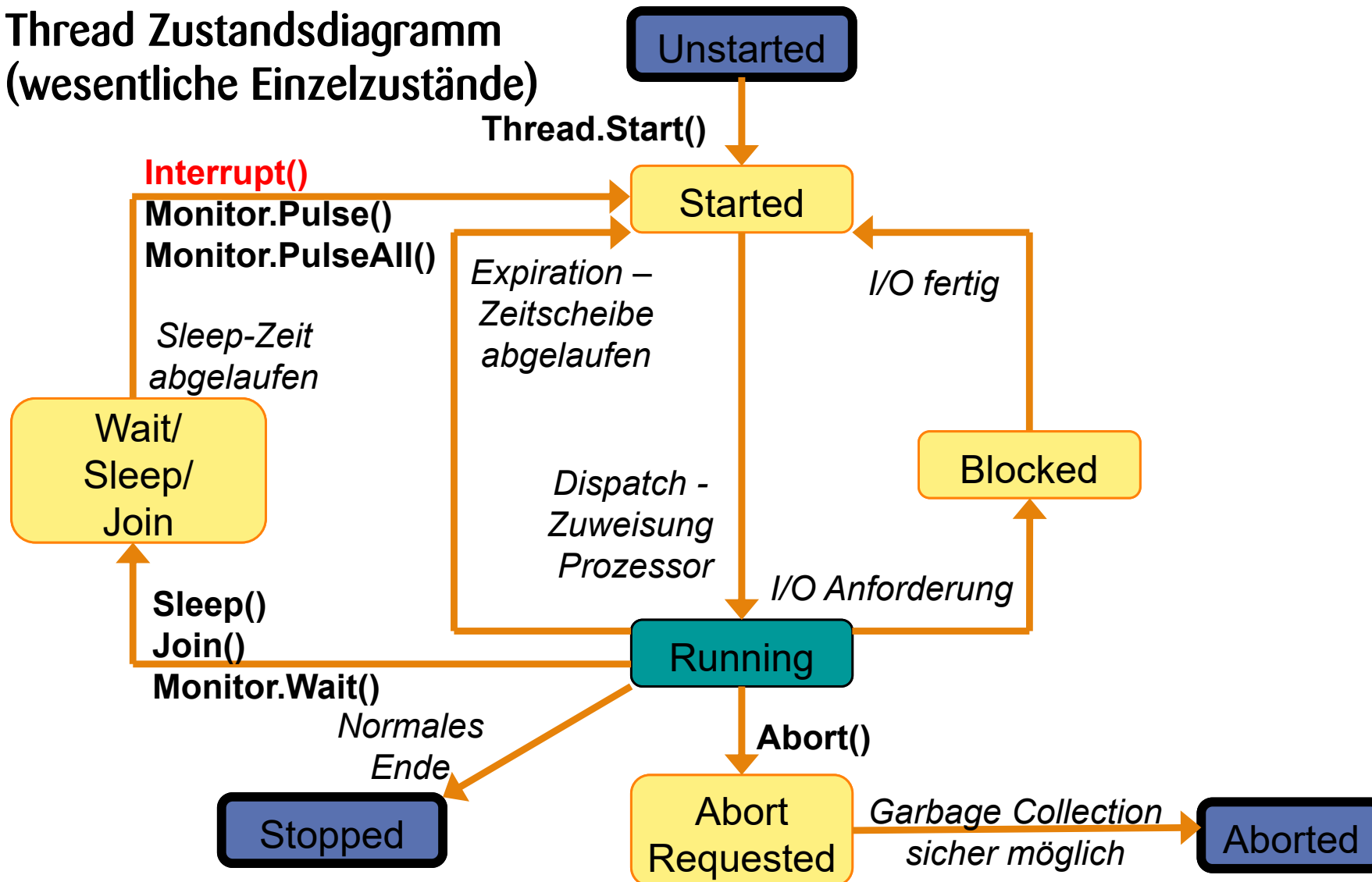
- Der Übergang aus dem Zustand `Running` in einen der anderen Zustände erfolgt in folgenden Fällen:
- Wenn das Betriebssystem den Ablauf der dem Thread zugeordneten Zeitscheibe feststellt, überführt das Betriebssystem den Thread wieder in den Zustand `Started`.
- Durch die Thread-Methode `Suspend()` wird er in den Zustand `SuspendedRequested` übergeführt, der dann nach Abschluß der dort zu erledigenden Arbeiten durch den Zustand `Suspended` abgelöst wird.
- Durch die Thread-Methoden `Sleep()` oder `Join()` bzw. durch die Monitor-Methode `Wait()` wird er in den Zustand `WaitSleepJoin` versetzt.
- Durch eine Aufforderung zum I/O hingegen wird der Thread in den Zustand `Blocked` (gehört nicht zu den eigentlichen Thread-Zuständen) versetzt.

Thread-Zustände

- Aus dem Zustand Blocked wird der Thread durch eine Fertigstellung des I/O wieder in den Zustand Started überführt.
- Durch die Thread-Methode `Resume()` wird der Thread vom Zustand Suspended wieder in den Zustand Started überführt.
- Durch die Thread-Method `Interrupt()`, durch die Monitor-Methoden `Pulse()` oder `PulseAll()` oder durch die Feststellung, dass die Sleep-Zeit abgelaufen ist, wird der Thread vom Zustand WaitSleepJoin wieder in den Zustand Started überführt.
- Wird die Methode, die in der Thread-Methode `Start()` als Thread-Routine angegeben wird, normal beendet, erfolgt der Übergang in den Zustand Stopped.
- Durch Aufruf der Thread-Methode `Abort()` wird er in den Zustand AbortRequested übergeführt, der dann nach Abschluß der dort zu erledigenden Arbeiten durch den Zustand Aborted abgelöst wird

Threads in .NET

Thread Zustandsdiagramm (wesentliche Einzelzustände)



Threads in .NET

Threads synchronisieren – Das Deadlock-Problem

- Wenn sich Threads Ressourcen teilen und für sich beanspruchen, kann es zu Deadlocks führen.
- Ein Deadlock führt zur Blockierung der beteiligten Threads.

Beispiel Code:

```
static void Main(string[] args)
{
    Object obj1 = new Object();
    Object obj2 = new Object();

    new Thread (() => {
        lock (obj1)
        {
            Thread.Sleep(1000);
            lock (obj2); // Deadlock
        }
    }).Start();

    lock (obj2)
    {
        Thread.Sleep(1000);
        lock (obj1); // Deadlock
    }
}
```

Threads in .NET

Threads synchronisieren – `lock()`

- In .NET können Ressourcen z.B. mit dem Ausdruck `lock()` gesichert werden.

```
// Beispiel: Objekt outputfile sichern:  
  
lock(outputfile)  
{  
    outputfile.Write(...);  
}
```

Threads synchronisieren - Monitor

- Eine weitere Möglichkeit Ressourcen zu sichern, bietet die Klasse `Monitor`. Die Klasse `Monitor` sichert das gewünschte Objekt.
- Die statischen Methoden `Enter()` zum Sperren und `Exit()` zum Freigeben bilden zusammen ein Paar.

```
// Beispiel: Objekt outputfile sichern:
```

```
// sichern
```

```
Monitor.Enter(outputfile);
```

```
...
```

```
outputfile.Write(...);
```

```
...
```

```
// freigeben
```

```
Monitor.Exit(outputfile);
```

Threads in .NET

Threads synchronisieren - Monitor

■ Beispiel mit `try`- und `finally`-Block:

```
// Beispiel: Objekt obj sichern:  
object obj;  
...  
try  
{  
    Monitor.Enter(obj);  
    // kritischer Abschnitt  
}  
finally  
{  
    Monitor.Exit(obj);  
}
```


Übung 5.6



Threads mit gemeinsamer Ressource - File

- A) Erweitere die Aufgabe 5.5 so, dass beide Threads (Incrementer und Decrementer) ihre Ausgabe in die gleiche Datei schreiben.

Für jeden Char im Ausgabestring soll die Methode `Write()` aufgerufen werden. Teste das Programm und kontrolliere den Dateiinhalt.

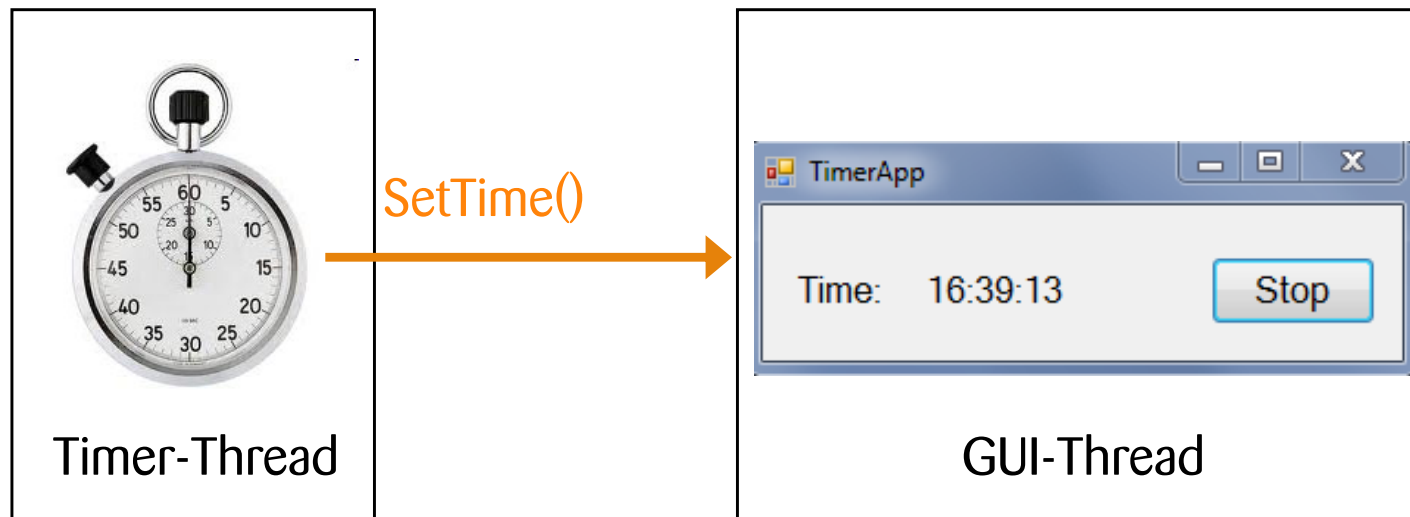
- B) Schütze den kritischen Bereich mit einer bekannten Methode.

Methoden in anderen Threads aufrufen

GUI-Controls aktualisieren aus Timer

■ Situation:

- «Windows Forms»-GUI-Controls sind nicht thread-safe
- `System.Timers.Timer` läuft in einem Timer-Thread



Methoden in anderen Threads aufrufen

GUI-Controls aktualisieren aus Timer ist „unsafe“

■ Direkter Zugriff auf GUI-Control

```
public partial class ThreadUnsafeExample : Form
{
    System.Timers.Timer timer;

    public ThreadUnsafeExample()
    {
        InitializeComponent();
        timer = new System.Timers.Timer();
        timer.Interval = 1000;
        timer.Elapsed += new System.Timers.ElapsedEventHandler(timer_Elapsed);
    }

    private void btnStartStop_Click(object sender, EventArgs e)
    {
        if (timer.Enabled == false)
        {
            timer.Start();
            btnStartStop.Text = "Stop";
        }
        else
        {
            timer.Stop();
            btnStartStop.Text = "Start";
        }
    }

    void timer_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
    {
        //
        lblTime.Text = String.Format("{0:D2}:{1:D2}:{2:D2}",
            DateTime.Now.Hour, DateTime.Now.Minute, DateTime.Now.Second);
    }
}
```

InvalidOperationException was unhandled by user code

Cross-thread operation not valid: Control 'lblTime' accessed from a thread other than the thread it was created on.

Troubleshooting tips:

- [How to make cross-thread calls to Windows Forms controls](#)
- [Get general help for this exception.](#)

[Search for more Help Online...](#)

Actions:

- [View Detail...](#)
- [Enable editing](#)
- [Copy exception detail t...](#)

Direkter Zugriff auf Label führt zu einer Exception

Methoden in anderen Threads aufrufen

GUI-Controls aus Timer aktualisieren ist „safe“

```
public delegate void UpdateDisplay();

public partial class Form1 : Form
{
    System.Timers.Timer timer;
    public UpdateDisplay myUpdateMethod;

    public Form1()
    {
        InitializeComponent();

        myUpdateMethod = new UpdateDisplay(SetTime);

        timer = new System.Timers.Timer();
        timer.Interval = 1000;
        timer.Elapsed += new System.Timers.ElapsedEventHandler(timer_Elapsed);
    }

    // läuft im Timer Thread
    void timer_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
    {
        this.Invoke(myUpdateMethod);
    }

    private void btnStartStop_Click(object sender, EventArgs e)
    {
        // Delegate Method
        void SetTime()
        {
            lblTime.Text = String.Format("{0:D2}:{1:D2}:{2:D2}",
                DateTime.Now.Hour, DateTime.Now.Minute, DateTime.Now.Second);
        }
    }
}
```

Zugriff auf Label via
Invoke und Delegate

Übung 5.7:

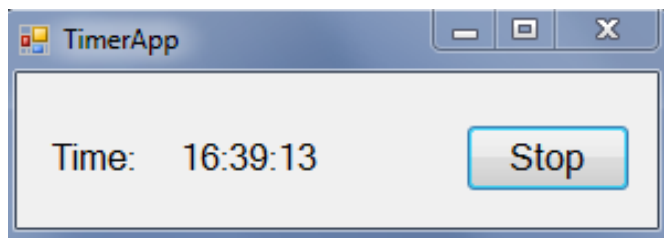


System.Timers.Timer- und Invoke-Methode

- Erstelle eine einfache «Windows Forms»-Applikation, um einen Timer zu starten und stoppen.

Verwende die Timer Klasse aus `System.Timers` und aktualisiere die Uhrzeit mittels der Methode `Invoke()`.

- Beispielapplikation:



Fore-/Background-Threads

Eigenschaften

- Standardmässig werden Threads als Foreground-Threads instanziiert
- Eine Applikation wird erst beendet, wenn alle Foreground-Threads beendet sind
- Sind alle Foreground-Threads beendet werden allfällig laufende Background-Threads abgebrochen (Aufräumen nicht möglich!) und die Applikation beendet
- Thread als Background definieren:
 - Eigenschaft: `IsBackground = true`

```
Thread worker = new Thread(() => Console.ReadLine());  
if (args.Length > 0) worker.IsBackground = true;
```

Fore-/Background-Threads

Beispielcode:

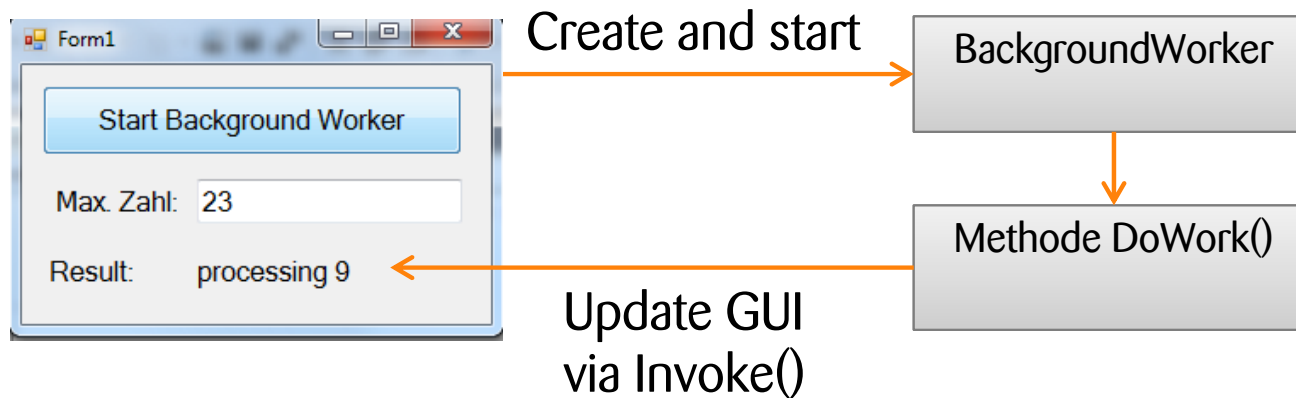
```
static void Main(string[] args)
{
    Thread worker = new Thread(() => Console.ReadLine());
    if (args.Length > 0) worker.IsBackground = true;
    worker.Start();
}
```

- Wird das Konsolenprogramm ohne Parameter gestartet, resultiert ein Foreground-Thread und die Applikation beendet erst nach Drücken der Enter-Taste
- Wird das Konsolenprogramm mit Parameter gestartet, resultiert ein Background-Thread. Die Applikation wird beendet, ohne dass auf die Eingabe gewartet wird, da der Background-Thread abgebrochen wird.

BackgroundWorker

Einführung

- BackgroundWorker ist eine Hilfsklasse im Namespace `System.ComponentModel`
- Vereinfacht das Ausführen einer länger dauernden Aufgabe im Hintergrund



BackgroundWorker

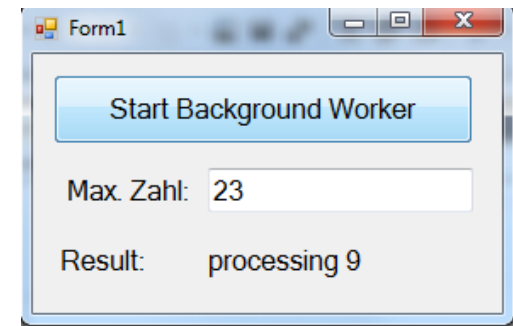
Beispiel

- BackgroundWorker instanziiieren

```
BackgroundWorker worker = new BackgroundWorker();
```

- DoWork () -Methode

```
private void DoWork(object sender, DoWorkEventArgs e)
{
    try
    {
        for (int i = 0; i <= (int)e.Argument; i++)
        {
            this.Invoke(new EventHandler(delegate
            {
                lblOutput.Text = string.Format("processing {0}\r\n", i.ToString());
            }));
            Thread.Sleep(200);
        }
    }
    catch (InvalidOperationException ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```



BackgroundWorker

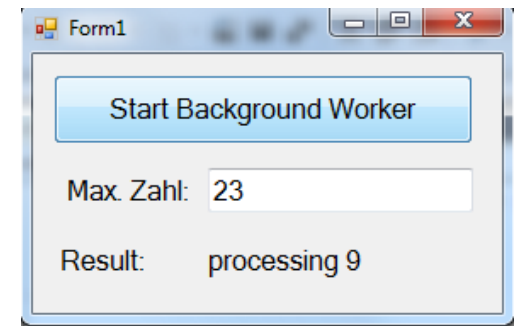
Beispiel

- Dem BackgroundWorker die DoWork-Methode zuweisen

```
worker.DoWork += new DoWorkEventHandler(DoWork);
```

- DoWork() im Background asynchron ausführen

```
private void btnStart_Click(object sender, EventArgs e)
{
    int zahl;
    Int32.TryParse(tbxMaxZahl.Text, out zahl);
    worker.RunWorkerAsync(zahl);
}
```



Übung 5.8:



BackgroundWorker

- Erstelle eine einfache «Windows Forms»-Applikation, welche eine rechenintensive Aufgabe im UI-Thread ausführt. Problem: Das UI reagiert nicht mehr auf Benutzereingaben.
- Lass die rechenintensive Aufgabe durch einen BackgroundWorker ausführen. Effekt: Das UI reagiert auf Benutzereingaben und keine UI-Thread-InvalidOperationException-Exception.