Threads

Beteiligte Typen (Auszug)

```
public sealed class Thread {
    public static Thread CurrentThread { get; }      // static methods
    public static void Sleep(int milliSeconds) {...}
    ...
    public Thread(ThreadStart startMethod) {...}     // thread creation

    public string Name { get; set; }                 // properties
    public ThreadPriority Priority { get; set; }
    public ThreadState ThreadState { get; }
    public bool IsAlive { get; }
    public bool IsBackground { get; set; }
    ...
    public void Start() {...}                        // methods
    public void Suspend() {...}
    public void Resume() {...}
    public void Join() {...}                         // caller waits for the thread to die
    public void Abort() {...}                        // throws ThreadAbortException
    public void Interrupt() {...}                    // callable in WaitSleepState
    ...
}

public delegate void ThreadStart();                  // parameterlose void-Methode

public enum ThreadPriority {Normal, AboveNormal, BelowNormal, Highest, Lowest}
public enum ThreadState {Unstarted, Running, Suspended, Stopped, Aborted, ...}
```

Beispiel

Zwei Threads, die ständig Zeichen auf dem Bildschirm ausgeben

```
using System;
using System.Threading;

class Printer {
    char ch;
    int sleepTime;

    public Printer(char c, int t) {ch = c; sleepTime = t;}

    public void Print() {
        for (int i = 1; i < 100; i++) {
            Console.Write(ch);
            Thread.Sleep(sleepTime);
        }
    }
}

class Test {

    static void Main() {
        Printer a = new Printer('.', 10);
        Printer b = new Printer('*', 100);
        new Thread(new ThreadStart(a.Print)).Start();
        new Thread(new ThreadStart(b.Print)).Start();
    }
}
```

Das Programm läuft so lange, bis der letzte Foreground-Thread beendet ist.

Unterschiede zu Java

SSW

C#

```
void P() {
    ... thread actions ...
}

Thread t = new Thread(new ThreadStart(P));
```

- Erfordert keine Unterklasse von *Thread*
- Beliebige Methode kann als Thread gestartet werden.
- *Abort*-Methode => *ThreadAbortException* Kann abgefangen werden, wird aber am Ende von catch automatisch wieder ausgelöst, außer man ruft *ResetAbort* auf. Alle finally-Blöcke werden ausgeführt, auch das Exit aus einem Monitor.

Java

```
class MyThread extends Thread {
    public void run() {
        ... thread actions ...
    }
}

Thread t = new MyThread();
```

- Eigener Thread muß Unterklasse von *Thread* sein.
- Thread-Aktionen müssen in Methode *run* stecken.
- *stop*-Methode ist deprecated weil gefährlich (gibt u.U. Monitor in inkonsistentem Zustand frei).

Thread-Zustände

```
Thread t = new Thread(new ThreadStart(P));
Console.WriteLine("name={0}, priority={1}, state={2}", t.Name, t.Priority, t.ThreadState);
t.Name = "Worker"; t.Priority = ThreadPriority.BelowNormal;
t.Start();
Thread.Sleep(1);
Console.WriteLine("name={0}, priority={1}, state={2}", t.Name, t.Priority, t.ThreadState);
t.Suspend();
Thread.Sleep(1);
Console.WriteLine("state={0}", t.ThreadState);
t.Resume();
Console.WriteLine("state={0}", t.ThreadState);
t.Abort();
Thread.Sleep(1);
Console.WriteLine("state={0}", t.ThreadState);
```

Ausgabe

```
name=, priority=Normal, state=Unstarted
name=Worker, priority=BelowNormal, state=Running
state=Suspended
state=Running
state=Stopped
```

Zustandsübergänge

Suspended

Safepoint erreicht

t.Resume

SuspendRequested

t.Suspend

Unstarted

t.Start

Running

t.Abort

AbortRequested

Thread.Sleep
other.Join
Wait(obj)

time over
other joined
Pulse(obj)

t.Interrupt

t.Abort

Exception behandelt,
finally abgearbeitet

WaitSleepJoin

Stopped

Beispiel für Join

SSW

```
using System;
using System.Threading;

class Test {

    static void P() {
        for (int i = 1; i <= 20; i++) {
            Console.Write('-');
            Thread.Sleep(100);
        }
    }

    static void Main() {
        Thread t = new Thread(new ThreadStart(P));
        Console.Write("start");
        t.Start();
        t.Join();  // wartet auf t
        Console.WriteLine("end");
    }
}
```

Ausgabe

start--------------------end

Behandeln von Abort

Abort löst Exception aus, die abgebrochener Thread behandeln kann.

```
using System; using System.Threading;

class Test {

    static void P() {
        try {
            try {
                try {
                    while (true) ;
                } catch (ThreadAbortException) { Console.WriteLine("-- inner aborted"); }
            } catch (ThreadAbortException) { Console.WriteLine("-- outer aborted"); }
        } finally { Console.WriteLine("-- finally"); }
    }

    static void Main(string[] arg) {
        Thread t = new Thread(new ThreadStart(P));
        t.Start(); Thread.Sleep(1);
        t.Abort(); t.Join(); Console.WriteLine("done");
    }
}
```

Ausgabe

```
-- inner aborted
-- outer aborted
-- finally
done
```

Wechselseitiger Ausschluß

SSW

lock-Anweisung

lock(*Variable*) *Statement*

Beispiel

```
class Account {                    // diese Klasse stellt einen Monitor dar
    long val = 0;

    public void Deposit(long x) {
        lock (this) { val += x; }  // jeweils nur 1 Thread darf diese Anweisung ausführen
    }

    public void Withdraw(long x) {
        lock (this) { val -= x; }
    }
}
```

Lock läßt sich auch auf beliebiges anderes Objekt setzen

```
object semaphore = new object();
...
lock (semaphore) { ... critical region ... }
```

Keine synchronized-Methoden wie in Java, aber entsprechendes Attribut

```
[MethodImpl(MethodImplOptions.Synchronized)]
public void Deposit(long x) {...}
```

Klasse Monitor

```
lock(v) Statement
```

ist Kurzform für

```
Monitor.Enter(v);
try {
    Statement
} finally {
    Monitor.Exit(v);
}
```

Wenn Thread während der Ausführung von *Statement* mit Abort abgebrochen wird, wird trotzdem finally ausgeführt und der Monitor freigegeben.

Wait und Pulse

Monitor.Wait(lockedVar);	≈ wait() in Java (in Java ist *lockedVar* immer *this*)
Monitor.Pulse(lockedVar);	≈ notify() in Java
Monitor.PulseAll(lockedVar);	≈ notifyAll() in Java

Ablauf an Hand eines Beispiels:

Thread A

```
1 lock(v) {
      ...
    2 Monitor.Wait(v); 5
      ...
  }
```

Thread B

```
3 lock(v) {
      ...
    4 Monitor.Pulse(v);
      ...
  } 6
```

1. A kommt zu *lock(v)* und erhält Eintritt, da kein anderer Thread im gesperrten Bereich ist.
2. A kommt zu *Wait*, legt sich schlafen und gibt die Sperre frei.
3. B kommt zu *lock(v)* und erhält Eintritt, da die Sperre frei ist.
4. B kommt zu *Pulse* und weckt damit A. Es kann (aber muß nicht) ein Thread-Wechsel stattfinden.
5. A versucht, die Sperre wieder zu erlangen, was aber nicht gelingt, da B sie noch hat.
6. B gibt am Ende des gesperrten Bereichs die Sperre frei; A kann nun wieder eintreten und weiterlaufen.

Wait und Pulse (Forts.)

SSW

Beachte

- *Wait(v)* und *Pulse(v)* dürfen nur in einer Anweisungsfolge aufgerufen werden, die mit *lock(v)* geschützt wurde.

- Zwischen *Pulse(v)* und dem dadurch geweckten Thread können andere Threads zur Ausführung gelangen, die in der Zwischenzeit versucht haben, die Sperre zu bekommen (d.h. die Bedingung von *Pulse* muß nach *Wait* nicht mehr gelten!)

  Daher sollte man die *Wait*-Bedingung immer in einer Schleife prüfen:

  ```
  while (condition false) Monitor.Wait(v);
  ...
  make condition true;
  Monitor.Pulse(v);
  ```

- *PulseAll(v)* weckt alle in *v* wartenden Threads, aber nur einer darf in den gesperrten Bereich. Der nächste darf erst rein, wenn der Vorgänger die Sperre freigibt.

Beispiel Pufferverwaltung

```
class Buffer {
    const int size = 4;
    char[] buf = new char[size];
    int head = 0, tail = 0, n = 0;

    public void Put(char ch) {
        lock(this) {
            while (n == size) Monitor.Wait(this);
            buf[tail] = ch; tail = (tail + 1) % size; n++;
            Monitor.PulseAll(this);
        }
    }

    public char Get() {
        lock(this) {
            while (n == 0) Monitor.Wait(this);
            char ch = buf[head]; head = (head + 1) % size;
            n--;
            Monitor.PulseAll(this);
            return ch;
        }
    }
}
```

Ablauf, wenn Producer schneller

Put
Put
Put
Put
Get
Put
Get
...

Ablauf, wenn Consumer schneller

Put
Get
Put
Get
...