

Multi-threading



C# threads

Concurrency

Creating and starting threads

Suspending, resuming and stopping threads

Background and foreground threads

Thread priorities

Concurrency – What?

Things happening at the same time.

(or switching tasks so fast, that we get the illusion
of things happening at the same time)



Concurrency – Why?

6861 1014

Go to www.menti.com and use the code 6861 1014

Why Concurrency?

Concurrency in C# - How?

Threads - today and the next week(s)

Tasks - when you chose Software Design in your 6th semester

Creating and starting threads

```
public class LotsOfWork
{
    public void DoLotsOfWork()
    {
        for (int i = 0; i < 1000; i++)
        {
            // TODO: add a lot of work here..

            Console.WriteLine("iteration: {0}", i);
        }
    }
}
```

Creating and starting threads

```
class Program
{
    static void Main(string[] args)
    {
        LotsOfWork work = new LotsOfWork();

        Thread myThread = new Thread(work.DoLotsOfWork);

        myThread.Start();

        System.Console.ReadKey();
    }
}
```


Creating and starting threads

```
class Program
{
    static void Main(string[] args)
    {
        LotsOfWork work = new LotsOfWork();

        Thread myThread = new Thread(work.DoLotsOfWork);

        myThread.Start();

        System.Console.ReadKey();
    }
}
```

```
public class LotsOfWork
{
    public void DoLotsOfWork()
    {
        for (int i = 0; i < 1000; i++)
        {
            // TODO: add a lot of work here..

            Console.WriteLine("iteration: {0}", i);
        }
    }
}
```

Passing parameters - Properties

```
class Program
{
    static void Main(string[] args)
    {
        LotsOfWork work = new LotsOfWork();

        Thread myThread = new Thread(work.DoLotsOfWork);
        work.NumberOfIterations = 500;

        myThread.Start();

        System.Console.ReadKey();
    }
}
```

```
public class LotsOfWork
{
    public int NumberOfIterations { get; set; } = 0;

    public void DoLotsOfWork()
    {
        for (int i = 0; i < NumberOfIterations; i++)
        {
            // TODO: add a lot of work here..

            Console.WriteLine("iteration: {0}", i);
        }
    }
}
```

Passing parameters – To the Start() method

```
class Program
{
    static void Main(string[] args)
    {
        LotsOfWork work = new LotsOfWork();

        Thread myThread = new Thread(work.DoLotsOfWork);

        myThread.Start(500);

        System.Console.ReadKey();
    }
}
```

```
public class LotsOfWork
{
    public void DoLotsOfWork(object parameter)
    {
        int numberOfIterations = (int) parameter;
        for (int i = 0; i < numberOfIterations; i++)
        {
            // TODO: add a lot of work here..

            Console.WriteLine("iteration: {0}", i);
        }
    }
}
```



Your turn

Solve exercises 1 and 2

and when done, jump to the
advanced exercises if you want to

Pausing a thread

```
public class LotsOfWork
{
    public void DoLotsOfWork(object parameter)
    {
        int numberOfIterations = (int) parameter;
        for (int i = 0; i < numberOfIterations; i++)
        {
            // TODO: add a lot of work here..

            Console.WriteLine("iteration: {0}", i);

            Thread.Sleep(50); // 50 milliseconds
        }
    }
}
```

The thread will sleep for ***at least*** 50 milliseconds.

Yield

```
public class LotsOfWork
{
    public void DoLotsOfWork(object parameter)
    {
        int numberOfIterations = (int) parameter;
        for (int i = 0; i < numberOfIterations; i++)
        {
            // TODO: add a lot of work here..

            Console.WriteLine("iteration: {0}", i);

            Thread.Sleep(0); // Yield
        }
    }
}
```


Sleep(0) means that the thread **yields**.

If another thread is ready to run, it will run.

If no other thread is ready to run, the thread that yielded will continue.

Don't use Thread.Suspend and Thread.Resume

C#

 Copy

```
[System.Obsolete("Thread.Suspend has been deprecated. Please use other classes  
in System.Threading, such as Monitor, Mutex, Event, and Semaphore, to  
synchronize Threads or protect resources. http://go.microsoft.com/fwlink  
/?linkid=14202", false)]  
public void Suspend ();
```

C#

 Copy

```
[System.Obsolete("Thread.Resume has been deprecated. Please use other classes  
in System.Threading, such as Monitor, Mutex, Event, and Semaphore, to  
synchronize Threads or protect resources. http://go.microsoft.com/fwlink  
/?linkid=14202", false)]  
public void Resume ();
```

<https://docs.microsoft.com/en-us/dotnet/api/system.threading.thread.suspend?view=net-5.0#System.Threading.Thread.Suspend>

<https://docs.microsoft.com/en-us/dotnet/api/system.threading.thread.resume?view=net-5.0#System.Threading.Thread.Resume>

When does the program exit?

```
class Program
{
    static void Main(string[] args)
    {
        LotsOfWork work = new LotsOfWork();

        Thread myThread = new Thread(work.DoLotsOfWork);
        myThread.Start();

        System.Console.WriteLine("Goodbye from main");

        System.Console.ReadKey();
    }
}
```

```
public class LotsOfWork
{
    public void DoLotsOfWork()
    {
        for (int i = 0; i < 1000; i++)
        {
            // TODO: add a lot of work here..

            Console.WriteLine("iteration: {0}", i);
        }
    }
}
```


Waiting for other threads to complete (join)

```
class Program
{
    static void Main(string[] args)
    {
        LotsOfWork work = new LotsOfWork();

        Thread myThread = new Thread(work.DoLotsOfWork);
        myThread.Start(50);

        myThread.Join();
        System.Console.WriteLine("Hello from main");

        System.Console.ReadKey();
    }
}
```

The `Join()` method blocks the caller until the thread on which join was called completes.

Waiting for other threads to complete (join)

```
class Program
{
    static void Main(string[] args)
    {
        LotsOfWork work = new LotsOfWork();

        Thread myThread = new Thread(work.DoLotsOfWork);
        myThread.Start(50);

        myThread.Join(1000); // continue if not joined after 1000 ms
        System.Console.WriteLine("Hello from main");

        System.Console.ReadKey();
    }
}
```

But you can specify a timeout.



Your turn

Solve exercises 3, 4, and 5

and when done, jump to the
advanced exercises if you want to

Background and foreground threads

```
class Program
{
    static void Main(string[] args)
    {
        LotsOfWork work = new LotsOfWork();

        Thread myThread = new Thread(work.DoLotsOfWork);
        myThread.IsBackground = true;

        myThread.Start(50);

        System.Console.ReadKey();
    }
}
```

The program will now terminate instantly, when a key is pressed, because myThread is a background thread.

Stopping a thread - gracefully

```
class Program
{
    static void Main(string[] args)
    {
        LotsOfWork work = new LotsOfWork();

        Thread myThread = new Thread(work.DoLotsOfWork);
        myThread.Start();

        System.Console.ReadKey();
        work.ShallStop = true;
    }
}
```

```
public class LotsOfWork
{
    public bool ShallStop { get; set; } = false;
    public void DoLotsOfWork()
    {
        int iteration = 0;
        while (!ShallStop)
        {
            Console.WriteLine("iteration: {0}", iteration);
            Thread.Sleep(50); // 50 milliseconds
            iteration++;
        }
        Console.WriteLine("Thread is done!");
    }
}
```

Aborting a thread (not so graceful)

```
class Program
{
    static void Main(string[] args)
    {
        LotsOfWork work = new LotsOfWork();

        Thread myThread = new Thread(work.DoLotsOfWork);
        myThread.Start();

        System.Console.ReadKey();
        myThread.Abort();
    }
}
```

```
public class LotsOfWork
{
    public bool ShallStop { get; set; } = false;
    public void DoLotsOfWork()
    {
        int iteration = 0;
        while (!ShallStop)
        {
            Console.WriteLine("iteration: {0}", iteration);
            Thread.Sleep(50); // 50 milliseconds
            iteration++;
        }
        Console.WriteLine("Thread is done!");
    }
}
```

Aborting a thread (not so graceful)

```
class Program
{
    static void Main(string[] args)
    {
        LotsOfWork work = new LotsOfWork();

        Thread myThread = new Thread(work.DoLotsOfWork);
        myThread.Start();

        System.Console.ReadKey();
        myThread.Abort();
    }
}
```

Avoid `Abort()` if you can.

`Abort()` throws an exception on the thread.

You don't know what the thread was doing, when it was aborted.

The exception can be caught by the thread (and the thread keeps running...).

Thread priorities - Danger, beware!

The .NET scheduler uses thread priorities when deciding which thread to run.

You almost never want to mess with this. So don't!



Your turn

Solve exercises 6, 7, and 8

and when done, jump to the
advanced exercises if you want to



AARHUS UNIVERSITY

References and image sources

Images:

Down the rabbit hole: <https://i.pinimg.com/originals/3b/a4/1c/3ba41c16b44edd7d9c5c9faeec965fad.gif>

Computer keyboard: http://stockmedia.cc/computing_technology/slides/DSD_8790.jpg

Nuclear explosion: <http://www.greenpeace.org/international/en/multimedia/photos/mushroom-cloud/>