.NET og C#

- Extension methods
- Events og callbacks
- Delegates
- Anonymous Methods
- Lambda expressions

EXTENSION METHODS

Extension methods – ikke muligt i Java

- Fra .NET 3.0 har det været muligt at "tilføje" en metode til en klasse *uden at skulle lave en subklasse* og *uden at skulle tilføje den direkte i source koden* for klassen
- Der er tale om en static metode og noget syntaktisk sukker for at det ser ud som om metoden er defineret på klassen. Men der er ikke nogen dynamisk binding.
- Det følgende eksempel tilføjer en WordCount() metode til string Klassen.
- WordCount returnerer antallet af ord i stringen

Definition af extension metoden

```
namespace ext  //normalt vil man samle extensions i en fil
{
  public static class StrExt
  {
    // new method in string class
    // the first parameter must be "this string"
    public static int WordCount(this string str)
    {
      string[] words = str.Split(new char[] { ' '});

      return words.Length;
    }
  }
}
```

Brug af extension method

```
using ext;  //bemærk namespace
...
static void Main(string[] args)
{
  string s = "Once Upon a Time in the West";

  int n = s.WordCount();
  Console.WriteLine("Number of Words: "+n);
}
...
```

Extension method med en parameter

```
public static string ExtractWord(this string str, int n)
{
  string[] words = str.Split(new char[] { ' ' });

  if (n < 0 || words.Length <= n)
    throw new ArgumentException("Wordindex out of bounds");
  else
    return words[n];
}
```

Call:

```
string w2 = s.ExtractWord(2); //nul-indiceret
```

Extension methods

Det er ren syntaktisk sukker, man kunne lige så godt kalde det som

```
string w2 = KlasseMedExtensionMetode.ExtractWord(s, 2);
```

DELEGATES

Delegates – metoder som typer/variabler!

Problem:

En metode skal kalde en anden metode, og den anden metode kendes først på runtime-tidspunktet.

Bemærkning:

I f.eks. C++ anvendes en memory pointer til at pege på metoden, hvilket er type unsafe.

Java: definer et interface med en metode, f.eks. Runnable, Comparator

C# Løsning:

Delegates, hvormed vi kan lave type sikre variable, som peger på en metode og som kan ændres på runtime, ligesom en variabel

Delegates – metoder/funktioner som variabler!

```
public delegate int MyDelegate(string s);  //så skal modtage en string og give en int tilbage

public int SomeMethod(string txt) {  //opfylder dette krav!
    if (txt is null)
        return -1;
    return txt.Length;
}

public void Run() {
    // Fundamental idea!
    MyDelegate f1 = new MyDelegate(SomeMethod); //these two ways are the same.
    MyDelegate f2 = SomeMethod;

    // Using the delegates – the two lines are doing 100 % the same thing!
    Console.WriteLine("From function f1 : " + f1.Invoke("hi there"));  // Invoke
    Console.WriteLine("From function f2 : " + f2("hi there"));         // Not using Invoke
}
```

Delegates

- Et delegate-objekt kan pege på enhver metode (statisk eller instans) som passer med delegate-definitionen:
  - samme returtype
  - samme antal og type af parametre
- Constructor'en for en delegate har præcis én parameter:
  - en metode, som passer med delegate-definitionen – f.eks. : `MyDelegate f1 = new MyDelegate(SomeMethod);`
  - `Eller blot MyDelegate f1 = SomeMethod;`

Delegates

- Vi er vant til at arbejde med variable, som indeholder data.
- Med delegates kan vi lave variabler, **som indeholder metoder**.
- Under afviklingen af programmet kan en delegate-variabel sættes til at indeholde den metode, som er relevant i den aktuelle situation, og variablen kan tildeles forskellige metoder på forskellige tidspunkter.

At en variabel kan indeholde/pege på ”kode” (dvs metoder/funktioner) er en del af funktionel programmering.
I C++ kaldes delegates også for *function references*

Delegates

Der findes to slags delegates (koden fra før er den første type herunder):

- `public abstract class Delegate : ICloneable, ISerializable`
- `public abstract class MulticastDelegate : Delegate`

Bemærk: De er begge to abstract, så vi kan ikke direkte lave objekter fra dem.

Singlecast og multicast delegates

- Et *singlecast* delegate-objekt indeholder en reference til **én metode**.
- Et *multicast* delegate-objekt (kan) indeholder referencer til **flere metoder**.
- Alle metoderne i et multicast delegate-objekt udføres, når objektet bruges som metode.
- Metoderne i en multicast delegate bør normalt have returtypen void, da returværdien bestemmes af den sidst kaldte metode.
- P.t. er alle delegates Multicast, og kaldes blot singlecast, hvis de kun indeholder en metode

Delegates "ligner" klasser

- En klasse definerer, hvordan objekter skal se ud (data og metoder).
- En delegate definerer, hvordan metoder skal se ud (returtype og parametre).
- En klasse bruges til at skabe objekter.
- En delegate bruges til at skabe delegate-objekter, som indeholder metoder.

- **Delegates og klasser er ikke det samme**, men de har lidt samme træk – enten at definere hvordan objekter eller metoder skal se ud.

Delegates, metoder og properties

Delegates indeholder bla. følgende metoder:

- `Invoke`, som benyttes til udførelse her og nu.
- `BeginInvoke` og `EndInvoke`, som benyttes til asynkron udførelse.
- `GetInvocationList`, som benyttes til aflæsning af en multicast delegates metoder.

Property:

- `Method` indeholder en singlecasts metode.

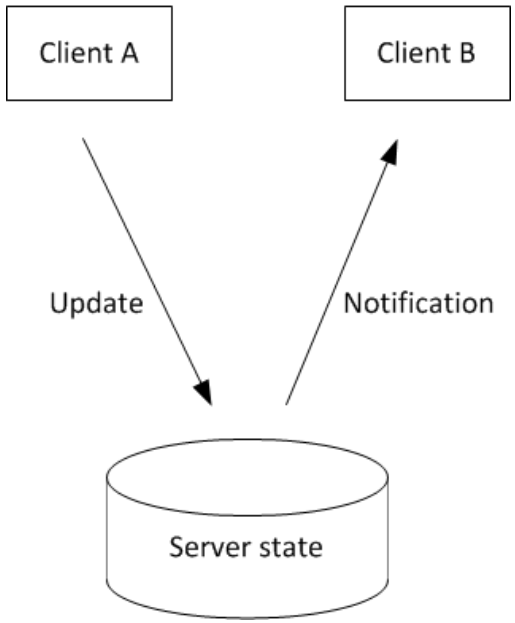
Sådan anvendes delegates

- `public delegate int Transformer(int x);`
- Metode: `private static int Square(int x) {...}`
- Metode, der anvender delegate:
- `public static void Transform(int[] values, Transformer t)`
- Kald af metoden: `Util.Transform(values, Square);`
- Multicast delegates: Får man hvis man tilføjer flere metoder:
- `Transformer t = Square;`
- `t += AddOne;`
- MEN det virker ikke så godt i forhold til returværdier, da der så bare returneres værdien af den sidste delegate.

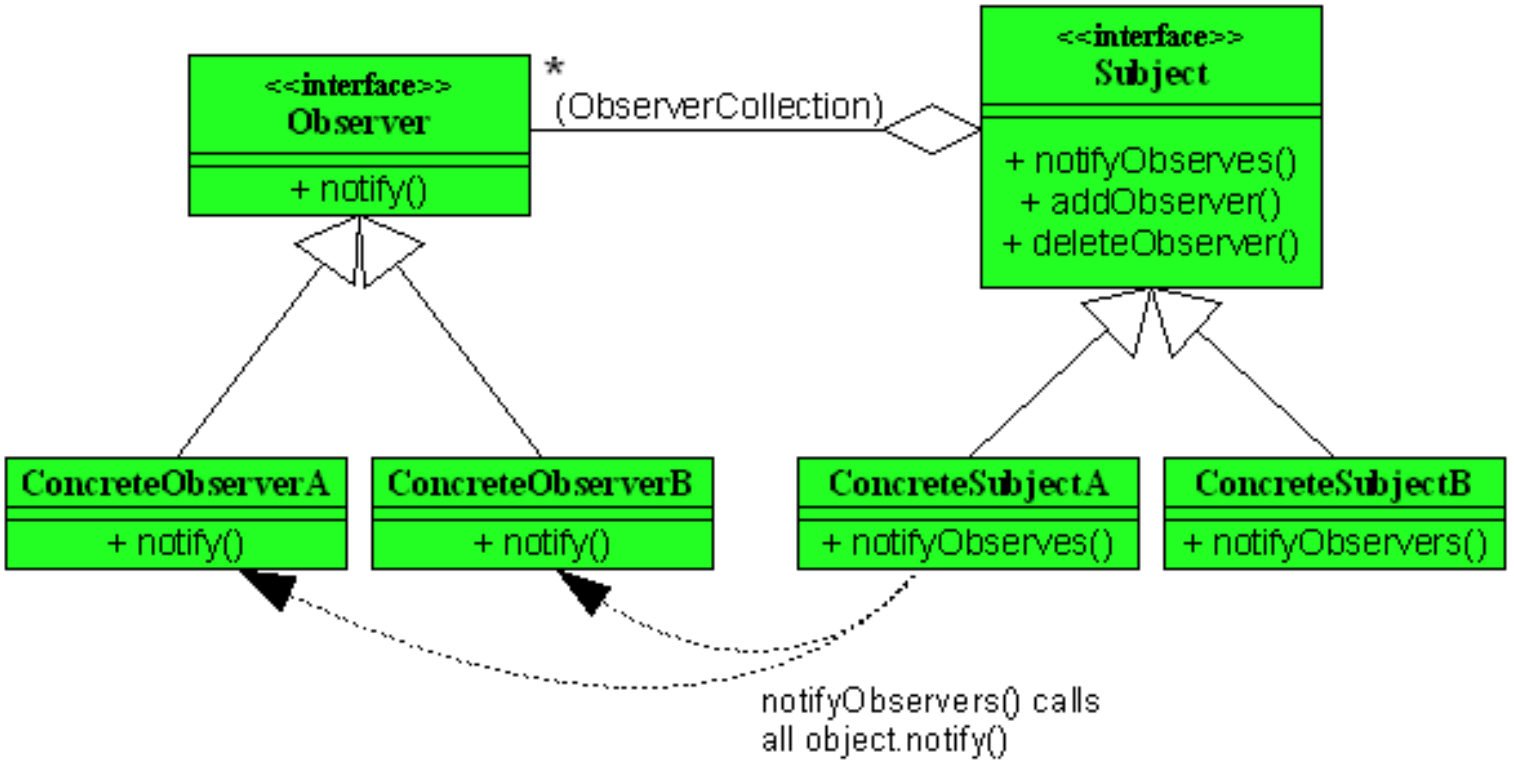
EVENTS OG CALLBACKS

Callback

- Typisk use-case er at en client skal notificeres at data er ændret.
- For eksempel i et client-server system som på tegningen:



Klassisk Observer pattern callback (Observer og Subject)



Events og Observer-pattern

- events i C# er en elegant og fleksibel facilitet til at implementere Observer-pattern
- **kræver ikke to interfaces og implementationer af disse, som man normalt ville gøre**.
- men kun
  - én linies erklæring af en **delegate type**
  - én simpel erklæring af en event variabel med delegatens type

EVENTS OG DELEGATES

(OBSERVER PATTERN 2.0)

Events (Observer pattern er en del af C#)

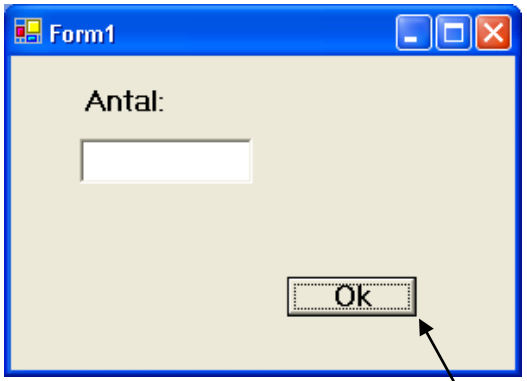
```
public delegate void MySimpleEvent(string s);
public event MySimpleEvent EventListener;  //event er en multi-cast delegate

public void EventListener1(string s) {  //bemærk – samme type som delegate
    Console.WriteLine("EventListener1 took action on : " + s);
}
public void EventListener2(string s) {  //bemærk same type som delegate
    Console.WriteLine("EventListener2 took action on : " + s);
}
public void Notify(string s) {
    if (EventListener != null)
        EventListener(s); //udfører eventhandlerne, som har typen MySimpleEvent
}
public void Run() {
    if(EventListener == null) {
        EventListener += EventListener1;
        EventListener += EventListener2;
    }
    Notify("Hello world");
}
```

Events

- **Events er lavet for at lave en abstraktion omkring delegates, et beskyttelseslag, så den kaldende kode har sværere ved at misbruge.**
- **Se f.eks. https://stackoverflow.com/questions/29155/what-are-the-differences-between-delegates-and-events for et eksempel**

Programmering med events – UI eksempel

- Et objekt kan udløse en hændelse (*raise an event*).
- Når en event udløses, vil en eller flere metoder blive kaldt; disse metoder kaldes *eventhandlere*.



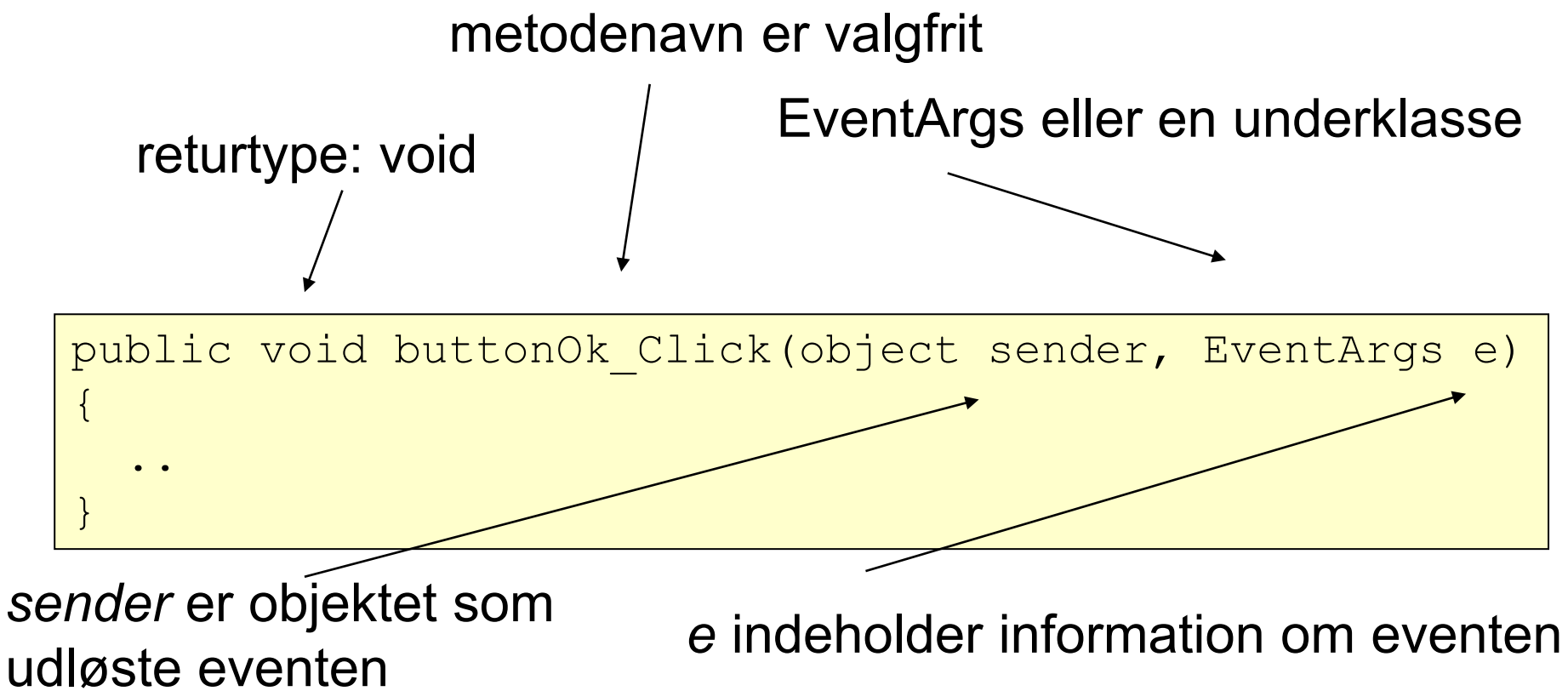

```
public class Form1 {
  ..
  public void buttonOk_Click(..) {
    ..
  }
}
```

buttonOk udløser Click-eventen, når en bruger klikker på knappen

buttonOk_Click() kaldes, når buttonOk udløser Click-eventen

Eventhandlere i .NET GUI'en

- Eventhandlere i .NET GUI'en følger en fastlagt skabelon



”Normale” metoder ved delegates

En ikke-anonym (”Normal”) delegate i GUI:

```
public partial class Form1 : Form
{
  // This method connects the event handler.
  public Form1()
  {
    InitializeComponent();
    button1.Click += new EventHandler(ButtonClick);
  }

  // This is the event handling method.
  private void ButtonClick(object sender, EventArgs e)
  {
    MessageBox.Show("You clicked the button.");
  }
}
```

Delegaten er erklæret som

```
public delegate void EventHandler ( object sender, EventArgs e )
```

Anonyme metoder ved delegate

Her er et eksempel på anvendelse af en anonym delegate:

```
public partial class Form1 : Form
{
  public Form1()
  {
    InitializeComponent();
    button1.Click += delegate(object sender, EventArgs e)
    {
      // følgende kode er del af en anonym metode
      MessageBox.Show("You clicked the button, and " +
        "This is an anonymous method!");
    };
  }
}
```

Gui-events og Visual Studio

Hver GUI-klasse har sine egne events og mange fælles.
Nedenfor er nogle af disse angivet:

- Button har Click, Enter
- Textbox har TextChanged, Enter, Leave
- ListBox har SelectedIndexChanged, SelectedValueChanged
- Form har Load, Move, FormClosing
- MenuItem har Click, DropDownOpened
- RadioButton har CheckedChanged
- ProgressBar har GiveFeedback

Gui-events og Visual Studio

Vælg ved hjælp af dobbeltklik en event i Property-vinduet i Visual Studio:

- Visual Studio genererer en tom eventhandler med den korrekte signatur. Eventhandlerens navn er en sammensætning af navnet på objektet, som udløser eventen, og eventens navn (f.eks. `buttonOk_Click()`)

Museklikket udløser kald af eventen Click, som er defineret i Button-klassen:

- `Button_Ok.Click(sender,e);//ikke synlig kode`

Events og .NET GUI'en

- Al interaktion mellem brugeren og programmets GUI udløser events.
- Programmøren skal skrive (udfylde) eventhandlere, som behandler events.

LAMBDA UDTRYK

(VI VIL ARBEJDE VIDERE MED DETTE I NÆSTE UGE – HER EN INTRO)

Anonymous Methods – kort erklæring af delegate, hvor kroppen er defineret uden et metode navn

```
public delegate int MyDelegate(string s);

public void Run() {

    MyDelegate f1 = delegate (string s) { return s.Length; };
    MyDelegate f2 = delegate (string s) { return (s is null) ? -1 : s.Length; };

    int res1 = f1("Hello world");
    int res2 = f2(null);

    Console.WriteLine("f1 returned " + res1);
    Console.WriteLine("f2 returned " + res2);}
```

Lambda expressions

- Et lambda udtryk (expression) er en kompakt måde at skrive en delegate funktion på.
- Et lambda udtryk har de samme input parametre og return parametre som signaturen på delegaten.
- Lambda udtryk foretrækkes frem for anonymous methods

Læsning af et lambda udtryk

- Lambda expression

```
i => i>=10   (hvad er input og output type)?
```

- Læses som input i **går til** output i>=10.

- Basalt set er et lambda udtryk blot en funktion som tager et input (af en type) og giver et output (af den samme eller anden type) (præcis som en delegate!)

Lambda expressions – eksempler.

```
//anonymous method
//MyDelegate f1 = delegate (string s) { return s.Length; };
//lambda
//MyDelegate f1 = (s) =>  { return s.Length; };
//lambda short notation
MyDelegate f1 = (s) => s.Length;
//anonymous
//MyDelegate f2 = delegate (string s) { return (s is null) ? -1 : s.Length; };
//lambda
MyDelegate f2 = (string s) => { return (s is null) ? -1 : s.Length; };

int res1 = f1("Hello world");
int res2 = f2(null);

Console.WriteLine("f1 returned " + res1);
Console.WriteLine("f2 returned " + res2);

Console.ReadLine();
```

Lambda expressions – eksempler.

```
public delegate int MyDelegate(string s);

MyDelegate f = (s)  => { return s.Length; };

Console.WriteLine("From f : " + f("Hello world"));

public MyDelegate GetNewDelegate() {
    int local_variable = 666;
    return (string arg) => { return local_variable; };
}

MyDelegate f7 = GetNewDelegate(); // Vi laver en ny delegate.
Console.WriteLine("From function f7 : " + f7("Hello world 5555"));

//Hvad udskriver dette program? Tænk lidt over koden. Der er to Writelines!
```


The Lambda expression

En fordel ved lambda udtryk: shortwriting

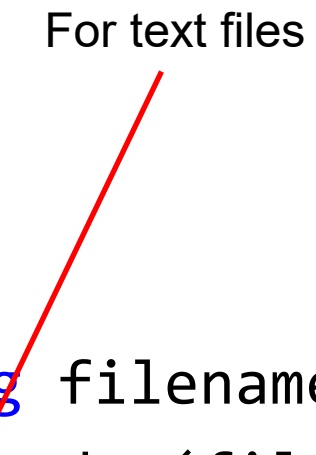
```
// find even numbers
List<int> even = list.FindAll(i => i % 2 == 0);  // hvad er typen på lambda udtrykket?

// find numbers with more than one digit
List<int> big = list.FindAll(i => i >= 10);  // hvad er typen på lambda udtrykket?
```

Så vi kan give et lambda udtryk som en parameter og det er så i dette tilfælde en slags filtreringsfunktion, som kun udvælger de elementer vi er interesserede i i listen (af int her).

Dette vil vi bruge i næste uge også, når vi skal snakke LINQ

PRAKTISKE ”VÆRKTØJER” (TIL BRUG I OPGAVERNE)

Læsning af Filer




```
using System.IO;

public static void ReadFile(string filename) {
    using (var file = new StreamReader(filename)) {
        string line;
        while ((line = file.ReadLine()) != null) {
            Console.WriteLine(line);
        }
    }
}
```

Splitting af strings

```
public static void SplitString(string txt) {
    var res = txt.Split(';');
    foreach (var s in res) {
        //Console.WriteLine("<" + s + ">");
        Console.WriteLine("<" + s.Trim() + ">");
    }
}
```

Remove white-spaces
from both ends.

IComparer<> interface and sorting Lists

Hvordan skal man sortere disse objekter?

```
public class DataBlock {
    public int    Id { get; set; }
    public double Number { get; set; }
}
```

By Number? By Id?

IComparer<> interface and sorting Lists

Lad os først lave den fulde klasse med en constructor og ToString

```
public class DataBlock {
    public DataBlock(int id, double number) {
        Id = id;
        Number = number;
    }
    public int    Id { get; set; }
    public double Number { get; set; }
    public override string ToString() {
        return $"Id:{Id}, Number:{Number}";
    }
}
```

IComparer<> interface and sorting Lists

Vi kan sortere ved at implementere interfacet IComparer:

```
public class SortById : IComparer<DataBlock> {
    public int Compare(DataBlock x, DataBlock y) {
        return x.Id.CompareTo(y.Id);
    }
}
```

Interface with Compare method, so we can call Sort

Use CompareTo on the System.Int32 object

IComparer<> interface

Nu kan vi sortere med vores ny klasse:

```
List<DataBlock> dataBlocks = GetListOfDataBlocks(); //fra fil/db etc.

dataBlocks.Sort(new SortById()); //her bruge vi vores nye klasse

foreach(var db in dataBlocks)
    Console.WriteLine(db);
```

Opgaver!