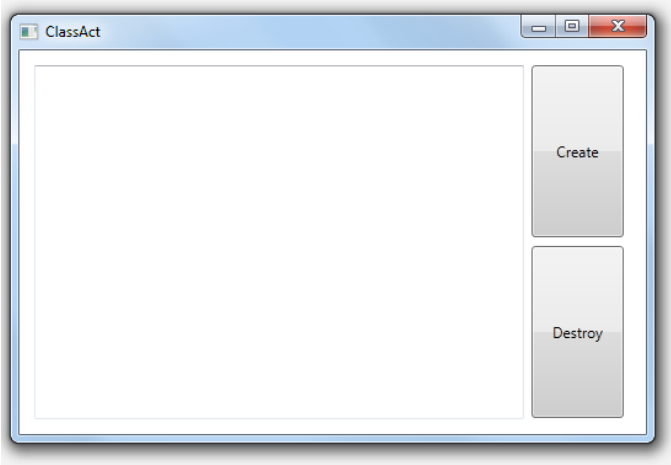
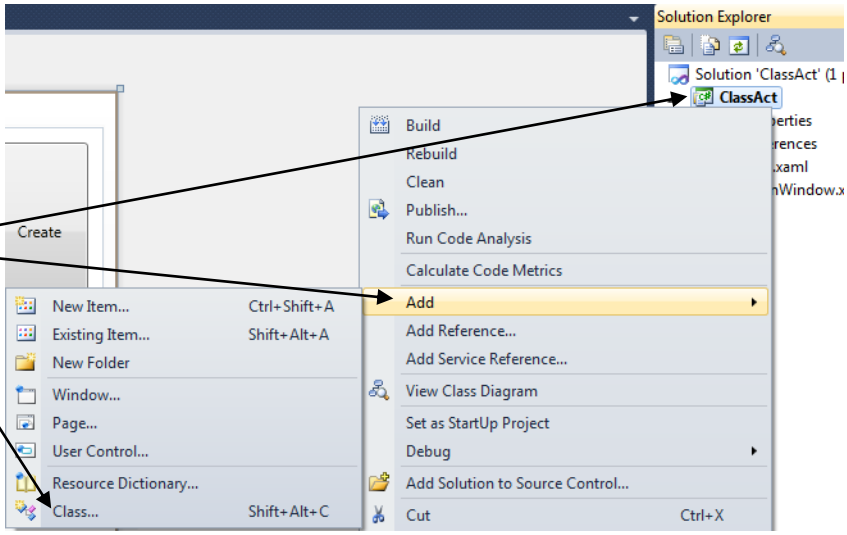
Du har indtil nu arbejdet med classes uden den store fokus på hvorledes disse hjælper dig i din programmering. Det skal vi lave lidt om på nu, vi skal nemlig prøve at oprette vores egne classes. Vi skal ligeledes se lidt på hvad der sker, når en class oprettes som en variabel.

Start med at oprette et nyt WPF projekt og tilføj en TextBox og to knapper som vist her:



Herefter skal der tilføjet endnu en class til projektet. Dette gøres ved at højre klikke på projektet i solution Explorer og vælg Add->Class:



I vinduet der kommer herefter omdøber du filnavnet fra Class1.cs til MyClass.cs

Dette navn overføres herefter til din nye class og filen der åbner i editoren skulle meget gerne se ud som herunder:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ClassAct
{
    class MyClass
    {
    }
}
```

Som du kan se er der default nogle using statements og samtidig bliver vores nye class "MyClass" automatisk tilføjet til vores namespace. (i dette tilfælde ClassAct)

Så er vi faktisk klar til at oprette et objekt af MyClass. Start med at tilføje eventhandlers til "Create" og "Destroy" knapperne og tilføj også følgende linje kode, lige over din MainWindow() Constructor:

```
public partial class MainWindow : Window
{
    MyClass myClass = null;
    public MainWindow()
    {
```

Dette laver en, for klassen "Global" variabel. Det vil sige at denne variabel kan tilgås fra samtlige funktioner i vores MainWindow Class.

Herefter skal vi have tilføjet koden der opretter et objekt af vores nye class i "Create" knappens eventhandler:

```
    private void btn_Create_Click(object sender, RoutedEventArgs e)
    {
        myClass = new MyClass();
    }
```

Og herefter kan du tilføje koden der frigiver objektet igen, til "Destroy" knappens eventhandler:

```
    private void btn_Destroy_Click(object sender, RoutedEventArgs e)
    {
        myClass = null;
    }
```

Husk på, at selv om *vi* skal oprette objektet, kan vi ikke selv bestemme hvornår det fjernes. Vi kan kun frigive vores variabel (myClass i dette tilfælde). Enten ved at det automatisk falder ud af scope. D.v.s. at vi forlader en funktion og derfor ikke længere har adgang til variablen, eller ved at vi tilskriver variablen med en ny værdi (I dette tilfælde null). Herefter overtager .NET GarbageCollectoren opgaven med at fjerne variablen fra hukommelsen.

Prøv om dit program kan kompilere og se hvad der sker når du trykker på knapperne... Ikke særlig spændende? Det skyldes at vi endnu ikke har tilføjet noget kode til vores class. Først skal vi have lavet en constructor.

Constructor

Det er sådan at alle classes der automatisk får en default constructor, af C# da det er denne der bruges når classen initieres via new keywordet. Denne kan vi overskrive ved at oprette vores egen.

Herunder er vist hvordan vi opretter en constructor og ligeledes en destructor som jeg vil komme ind på lidt senere.

Som du kan se, har jeg også tilføjet lidt Debug kode til begge to. Denne information kommer i output vinduet og du kan derfor se hvornår de enkelte funktioner kaldes.

```
class MyClass
{
    public MyClass()
    {
        Debug.Write("Dette er Constructoren!");
    }

    ~MyClass()
    {
        Debug.Write("Dette er Destructoren!");
    }
}
```

Hvis du starter projektet igen og samtidig åbner output vinduet (Via menuen View->Output) Kan du se at der sker noget når du trykker på henholdsvis Create og Destroy knapperne. Det kan dog godt tage nogen tid inden destructoren kaldes, da denne jo kun køres når Garbage collection er igang.

Selv om det frarådes fra Microsoft, kan du faktisk godt gennemtvinge GC'en ved at tilføje koden nedenfor:

```
    ~MyClass()
    {
        Debug.Write("Dette er Destructoren!");
        GC.Collect();
    }
```

Her "beder" du GC'en om at rydde op, men som du sikkert har lagt mærke til, opfører den sig dog stadig temmelig teenager artigt. Den starter nemlig ikke nødvendigvis lige med det samme...

Nu har vi oprettet en simpel klasse. Men udover at den viser os hvornår den oprettes og fjernes fra hukommelsen har den ikke meget funktionalitet. Så vi starter med at lave et par variabler (Fields på engelsk) i toppen af classen:

```
class MyClass
{
    int Tal;
    string Navn;

    public MyClass()
    {
```

Nu burde vi kunne bruge dem i vores MainWindow, lige efter at vi har oprettet klassen:

```
private void btn_Create_Click(object sender, RoutedEventArgs e)
{
    myClass = new MyClass();
    myClass.Navn = "Thomas";
}
```

Men prøver du dette melder Visual Studio at der er en fejl... Hvad kan dette skyldes? Og hvad skal der til for at rette fejlen?

Når du har rettet fejlen. Kan du jo prøve om du kan udskrive navnet til tekstboksen fra eventhandleren til "Destruct" knappen ***INDEN*** du tilskriver den med null.

Hvad sker der hvis du trykker to gange på Destroy knappen?

Kigger du efter, er destructoren på myClass objektet endnu ikke kørt, men du får stadig en fejl. Og dette skyldes ene og alene at vores "handle" myClass nu ikke længere peger på objektet. Og derfor ikke kan returnere variablen... Men data er der stadig! **Prøv at slette koden der skriver til tekstboksen igen og** tilføj herefter koden herunder til destructoren i MyClass:

```
if (Navn != null)
{
    Debug.Write("Hej " + Navn + ",  dette er Destructoren!");
}
```

Hvis du kører projektet og venter på GC'en vil du se at den skriver navnet ud, uden problemer.

Properties

Indtil videre har vi haft direkte adgang til de to variabler. Men dette anses ikke for god OOP stil. For hvad nu hvis det drejede sig om andet end simple typer. Lad os sige at Navn kom fra en database eller en tekstfil. Så kunne det, afhængig af koden i vores class, være at brugeren af vores class havde direkte adgang til logikken bagved. Og en enkelt fejl, hvad enten den var uskyldig eller ej, kunne have fatale konsekvenser for vores class. Derfor "gemmer" vi vores variabler. Dette kan gøres med simple funktioner som eksempelvis "SetNavn" og "GetNavn" men dette kan hurtigt blive temmelig rodet. Derfor har vi properties.

Du opretter nemmest en property via codesnippets: Du skriver "prop" i toppen af classen og trykker tab to gange. Herefter får du en default property af typen int forærende. Du kan ændre typen og navnet til hvad du ønsker. Så prøv at lave de to variabler om til rigtige properties:

```
class MyClass
{
    public int Tal { get; set; }
    public string Navn { get; set; }

    public MyClass()
    {
```

Resten af din kode virker stadig. Da funktionaliteten af en property er ganske som med en public variabel.

Private vs Public Properties

Resten af denne side og næste er kun for at give nogle eksempler på hvorledes properties kan benyttes. Du skal derfor ikke indtaste koden, men blot læse og forstå koden.

Hvad nu hvis ikke du ønsker at brugeren af din class skal kunne ændre i variablerne udefra?

Vi har allerede set hvor begrænset en private variabel er i forhold til en private. Og properties giver os faktisk mulighed for at begrænse adgangen i begge retninger:

Eksempelvis ville følgende kode give en property der kunne aflæses men ikke tilskrives:

```
        public int Tal { get; Private set; }
```

Og modsat, kan en private getter gøre det umuligt at læser fra propertien.

Koden vi har set for en property indtil nu er den absolut minimale. Og der er faktisk en del gemt "inde bag ved" Vi kan selv vælge at tilføje en komplet property. Og den ville i det tilfælde se ud som på næste side.

```
private int _tal;

public int Tal
{
    get
    {
        return _tal;
    }
    set
    {
        _tal = value;
    }
}
```

Og det er jo en del mere kode, men der er faktisk tilfælde hvor dette er nødvendigt. Forestil dig eksempelvis at vi har noget valideringskode i setteren:

```
public int Tal
{
    get
    {
        return _tal;
    }
    set
    {
        _tal = value;

        // Check om der er overløb
        if (_tal > 200)
        {
            _tal = 200;
        }
    }
}
```

Eller hvis der skulle returneres noget mere avanceret som her:

```
public string SvarTextBoxText
{
    get
    {
        return tb_Svar.Text;
    }
}
```

Her er det kun muligt at hente svaret fra tekstboksen, da setteren er undladt.

Non default constructore

Vi har kigget på hvorledes det er muligt at oprette properties. Samt hvorledes disse kan beskyttes på den ene eller anden måde. Men hvis man har en property der kun kan læses fra. Hvordan skal man så kunne skrive til den? Jo ser du, dette kan gøres på flere forskellige måder. Enten via intern kode i den enkelte class. Via en public funktion. Eller via constructoren. Den "gratis" version af constructoren kan ikke modtage noget input. Så vi har derfor mulighed for at tilføje en "overloadet" constructor.

Prøv at tilføje en overloaded constructor til MyClass som vist herunder:

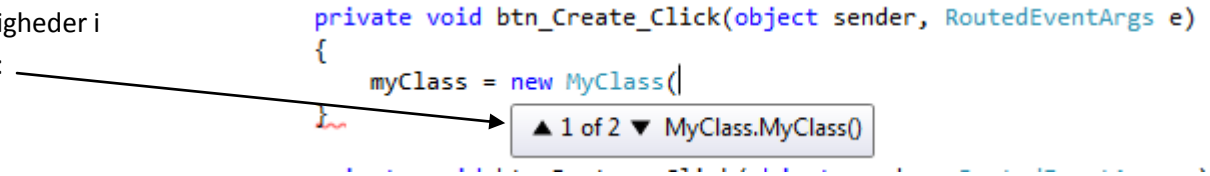
```
class MyClass
{
    // Properties med private settere
    public int Tal { get; private set; }
    public string Navn { get; private set; }

    public MyClass()
    {
        Debug.Write("Dette er Default Constructoren!");
    }

    public MyClass(int tal, string navn)
    {
        Tal = tal;
        Navn = navn;

        Debug.Write("Dette er Non-default Constructoren!");
    }
```

Hvis du går tilbage i "Construct" knappen og omskriver den linje hvori du oprettede myClass, vil du se at der nu er 2 muligheder i intellisence:



Den første er default constructoren. Og trykker du pil ned, får du mulighederne for at anvende vores non-default constructor (Også kaldet en overloaded constructor!)

De muligheder du har set i denne opgave gælder faktisk for samtlige (ikke statiske) classes i C#... Jeps også MainWindow... Prøv at tilføje følgende kode til din "construct" knap's eventhandler og kør programmet:

```
MainWindow mainWindow = new MainWindow();
mainWindow.Show();
```