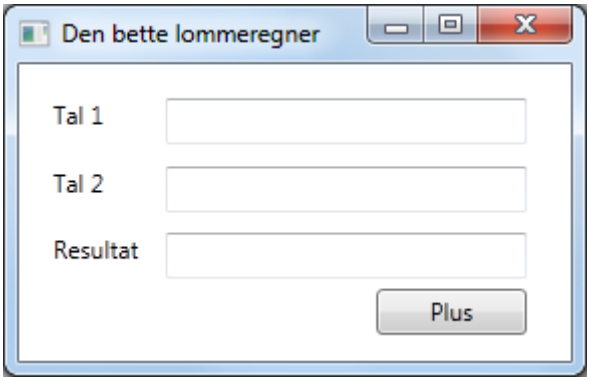


Vi skal i den følgende opgave lave en lille lommeregner som i sin grundform kan addere 2 tal. Billedet ovenfor viser hvordan den kunne se ud.

Start med at oprette et nyt C# WPF projekt kaldet ”DenBetteLommeregner” Tilføj derefter kontroller til brugerfladen som vist ovenfor.

Tilføj herefter et ”Click” event til knappen.

I event handleren (funktionen som håndterer Click eventet) skal du nu oprette tre variabler af typen double:

```
double tal1 = 0;
double tal2 = 0;
double resultat = 0;
```

Vores tekstbokse indeholder tekst af typen string. Og da vi kun kan regne med tal skal disse strings konverteres.

Det kan vi gøre på 2 måder:

Parse (Den lille hurtige)

Via funktionen parse() - Parse er en hjælpefunktion som er indbygget i samtlige simpel typer. Og den benyttes til at ”oversætte” en string til den variabel type Parse() Kaldes fra…

Et eksempel er vist på sin plads:

```
double tal1 = 0;
tal1 = double.Parse(tbTal1.Text);
```

destinations variable, type der skal konverteres til, string der skal konverteres

Parse er den hurtigste konvertering der findes i C#, men er begrænset til kun at konverterer fra string til den type funktionen Parse kaldes fra eks double.Parse()

Convert (Den tunge men alsidige)

Convert er en static class lavet udelukkende til konvertering imellem forskellige typer.

Et eksempel kunne være:

```
double tal1 = 0;
tal1 = Convert.ToDouble(tbTal1.Text);
```

destinations variable, type der skal konverteres til, Variabel der skal konverteres fra

Convert er i modsætning til Parse ikke begrænset til kun at konvertere fra strings, den kan også konvertere fra et tal til et andet. Og hvis du ikke passer på, smider den gerne decimaler væk…

Prøv at konvertere indholdet af de to tekstbokse's (**Text**) og tilskriv resultatet til henholdsvis tal1 og tal2.

Plus herefter de to tal og tilskriv variablen "resultat" med resultatet.

```
resultat = tal1 + tal2;
```

Herefter kan resultatet udskrives til "resultat" tekstboksen med følgende linje kode:

```
tbResultat.Text = resultat.ToString();
```

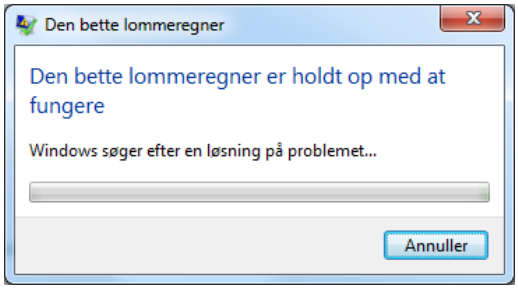
Hvor **tbResultat** er resultat tekstboksen.

Prøv først om du selv kan lave regne funktionen færdig. Hvis der opstår problemer, kan du se hele koden sidst i denne opgave.

Fejlhåndtering

Programmet virker problemfrit så længe brugeren ***kun*** indtaster tal. Men hvad sker der hvis der sniger sig et bogstav med ind i en af tekstboksene? (Prøv det på din egen applikation og noter resultatet, eks. Hvor fejlen opstår i koden)

Den uheldige fejlmeddelelse er endnu mere ulæselig hvis ikke programmet kører i debuggeren



Fejl håndtering i vores programmer er vigtigt af flere grunde, først og fremmest giver det en god bruger oplevelse når tingene virker. Også når brugeren gør noget forkert! Dernæst kan simpel fejlhåndtering være en nem måde at beskytte vores data på. Eksempelvis i forbindelse med fil håndtering eller databaseforbindelser. Forestil dig at brugeren vil gemme en fil men taster forkert.

I forbindelse med den simple regnemaskine vi sidder og udvikler er fejlhåndtering primært et værktøj til at fremme brugeroplevelsen. Skulle programmet crashe, kan brugeren jo bare starte forfra igen…

I C# bruger vi tre små simple kommandoer til at teste for og fange fejl.

Try/Catch strukturen ser i sin simpleste form ud som vist her til højre:

Test en eller flere statements

Fang eventuelle fejl

Udfør en handling i tilfælde af fejl

```
try
{

}
catch (Exception)
{
    throw;
}
```

I vores program kunne vi implementerer try/catch strukturen som vist herunder:

(**Skriv try og tryk tab tasten 2 gange for at auto genererer try/catch strukturen via *code snippets!***)

```
double tal1 = 0;
double tal2 = 0;
double resultat = 0;

try
{
   tal1 = Convert.ToDouble(tbTal1.Text);
   tal2 = Convert.ToDouble(tbTal2.Text);
   resultat = tal1 + tal2;
}
catch (Exception)
{
   Throw;
}
tbResultat.Text = resultat.ToString();
```

Det ville betyde at vi fangede eventuelle konverteringsfejl. Prøv at køre programmet i debuggeren endnu en gang og fremprovoker fejlen igen. Har tingene ændret sig?

Som du sikkert har lagt mærke til er fejlen flyttet fra selve konveteringen og ned til "throw" i "Catch" delen. Altså er der noget der virker… Vi tester for fejl og vi fanger den.

Men da Visual Studio ikke ved hvad den skal gøre med fejlen har den *(hvis du autogenerede try/catch statementen via code snippets)* indsat en "throw" statement som "kaster" fejlen videre til debuggeren. (eller windows hvis du kører uden debugger)

Vi skal derfor selv kode os ud af en eventuel fejlhåndtering. Den simple måde på det var at udskifte throw statementen med en tekstboks som advarer brugeren om fejlen og ikke andet:

```
catch (Exception)
{
    MessageBox.Show("Det er noget galt!", "Hov...");
}
```

Prøv programmet og se om det har den ønskede effekt? Nej vel?

Der kommer en fejlmeddelelse men ikke meget forklaring på hvad problemet er. Det kan vi heldigvis også lave om på. Prøv at fange "Exception" ved at navngive den som vist herunder:

```
catch (Exception exc)
```

Nu har vi mulighed for at bruge den fangede undtagelse direkte i vores kode, for ved at navngive den har vi konverteret den til et objekt vi kan behandle som alle andre objekter i vores kode.

```
MessageBox.Show(exc.Message, "Hov...");
```

Exception objektet

Message er integreret i objektet

Prøv at omskriv dit program så det benytter MessageBox sætningen vist herover, således det viser selve fejlmeddelelsen til brugeren.

Hvis det virker korrekt vil brugeren nu får en fin fejlbeskrivelse uden at programmet crasher. Han kan derefter rette fejlen og fortsætte sin udregning.

Den færdige regnefunktionen til Plus tasten

```
double tal1 = 0;
double tal2 = 0;
double resultat = 0;

try
{
   tal1 = Convert.ToDouble(tbTal1.Text);
   tal2 = Convert.ToDouble(tbTal2.Text);
   resultat = tal1 + tal2;
}
catch (Exception exc)
{
   MessageBox.Show(exc.Message, "How! Der er opstået en fejl...");
}
   tbResultat.Text = resultat.ToString();
```