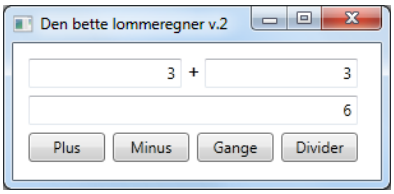
Side 1 og 2 i denne opgave er en introduktion til opgaven, så du bør læse siderne igennem inden du begynder at lave et nyt program…

Du skal i denne opgave modificere din lommeregner fra tidligere. Denne gang skal den dog også indeholder funktionalitet for minus, divider, gange. Og ligeledes tilføje et label som viser regnearten imellem tal1 og tal2 tekstboksen i stil med billedet her:



Funktionaliteten af de nye eventhandlers er ikke meget anderledes end den for plustasten. Dog skal regnetegnet jo ændres. Men med en hurtig omgang copy/paste er du snart færdig med opgaven…

Her til højre ses den kode der skal til for at få tingene til at virke... Det fylder jo en hel del, eller?

```
private void btn_Plus_Click(object sender, RoutedEventArgs e)
{
    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, "Hov! Der er opstået en fejl...");
    }
    tbResultat.Text = resultat.ToString();
}

private void btn_Minus_Click(object sender, RoutedEventArgs e)
{
    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, "Hov! Der er opstået en fejl...");
    }
    tbResultat.Text = resultat.ToString();
}

private void btn_Gange_Click(object sender, RoutedEventArgs e)
{
    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, "Hov! Der er opstået en fejl...");
    }
    tbResultat.Text = resultat.ToString();
}

private void btn_Divider_Click(object sender, RoutedEventArgs e)
{
    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, "Hov! Der er opstået en fejl...");
    }
    tbResultat.Text = resultat.ToString();
}
```

Og det her er ”kun” en simpel lommeregner, forestil dig at du udvikler et *rigtigt* stort og avanceret stykke software. Så ville det hurtigt blive uoverskueligt at skulle copy paste alt den kode. Og hvad hvis det skal rettes noget i kode? Et navn der skal ændres eller noget i den stil?

Der er heldigvis en løsning. Kodegenbrug! Vi samler det kodestumper som til forveksling ligner hinanden, og som vi ønsker at benytte igen og igen i metoder. Så skal vi for eftertiden kalde disse metoder når vi ønsker en given funktionalitet.

Tænk eksempelvis på Convert.ToDouble() her kaldte vi ”ToDouble” hver gang vi ønskede at konvertere noget tekst til tal.

Hvis vi lavede vores egen RegneFunktion() kunne vi kalde den hver gang vi ønskede at udregne resultatet af to tal. I sin simpleste form kunne den se ud som herunder:

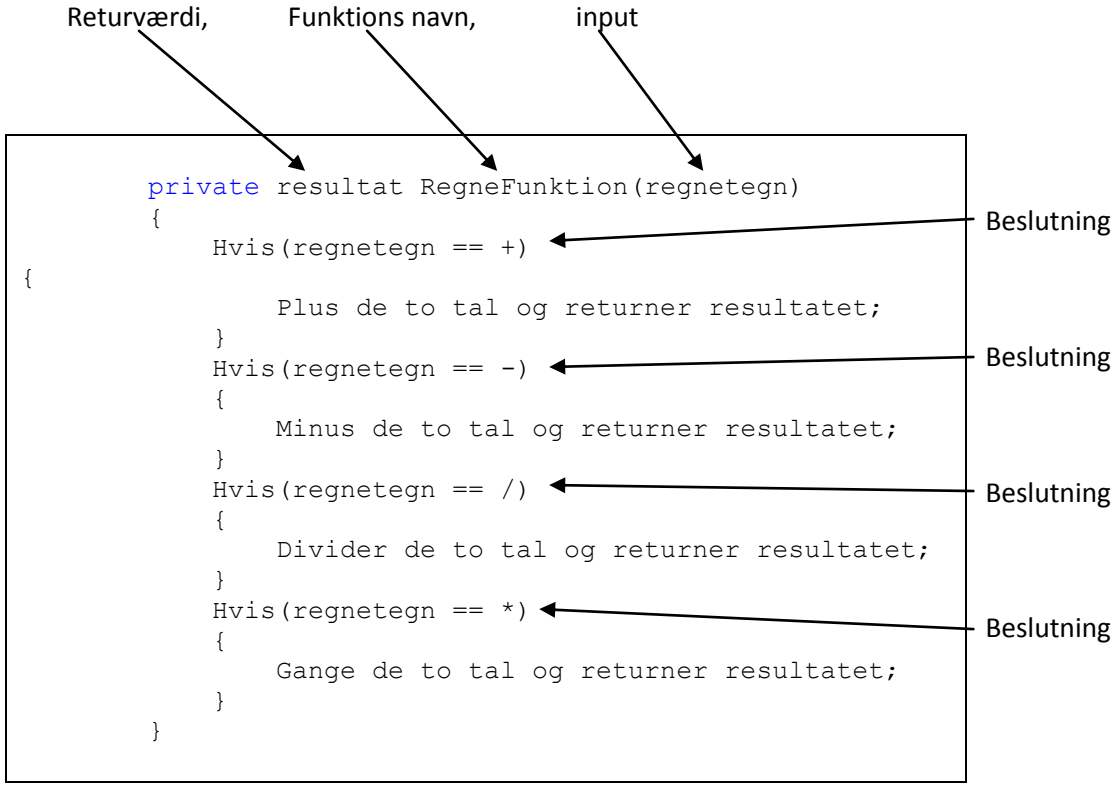
(Metodens Access modifier), Metodens returværdi(ingen i dette tilfælde),

```
private void RegneFunktion()
{

}
```

Funktionens scope, Funktionens navn, Funktionene input (ingen i dette tilfælde)

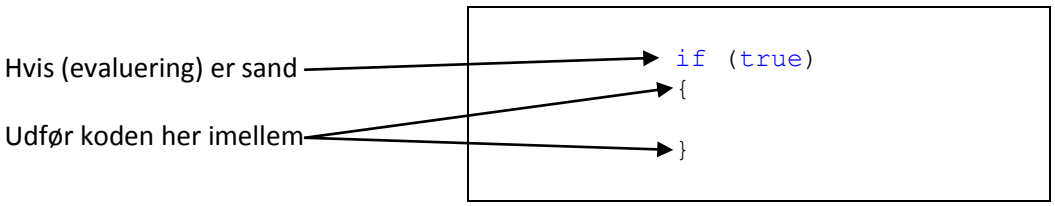
Men hvordan ved vi hvilken funktion vi skal kalde når vi trykker på plusknappen? Med den foregående funktion kunne vi ikke vide det … For at bestemme regnearten i den kaldte funktion, skal vi levere denne information med

Herunder er funktionen vist i ***pseudokode*** som gerne skulle illustrerer den funktionalitet vi ønsker:



Husk på at pseudokode *ikke er rigtig kode*! Det er blot et redskab til hurtigt at illustrere en given funktionalitet.

Beslutningerne i pseudokoden ovenfor er opbygget af IF sætninger og de virker som vist herunder:



Evalueringen skal resulterer i en boolesk ”true” altså 1==1 eller ’t’==’t’ osv. for at koden i de to tuborg klammer udføres, ellers springes dette kode over.

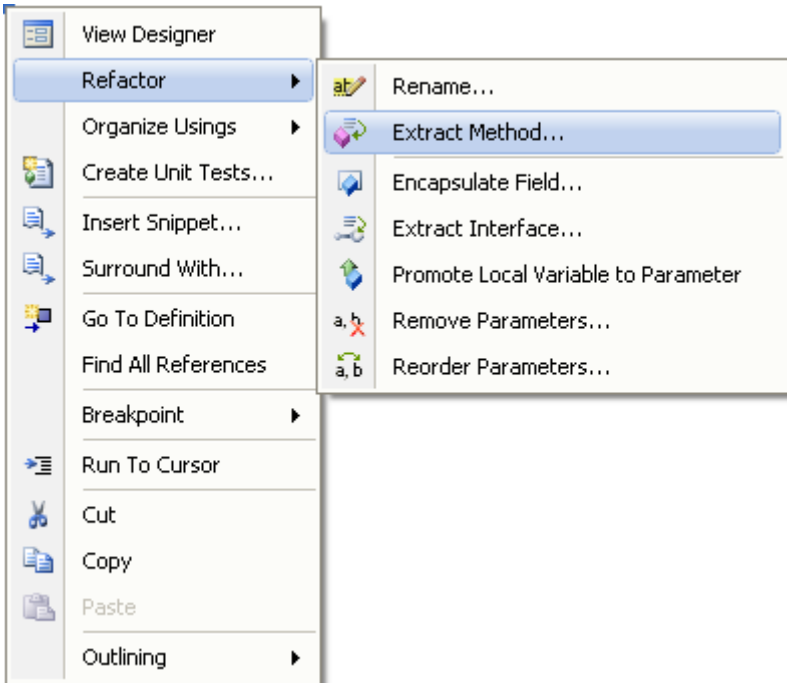
Vi kan tilføje en funktion til vores projekt på to måder.

Enten direkte via kode eller ved hjælp af Visual Studio. Vi vil for eksemplets skyld bruge den sidste metode:

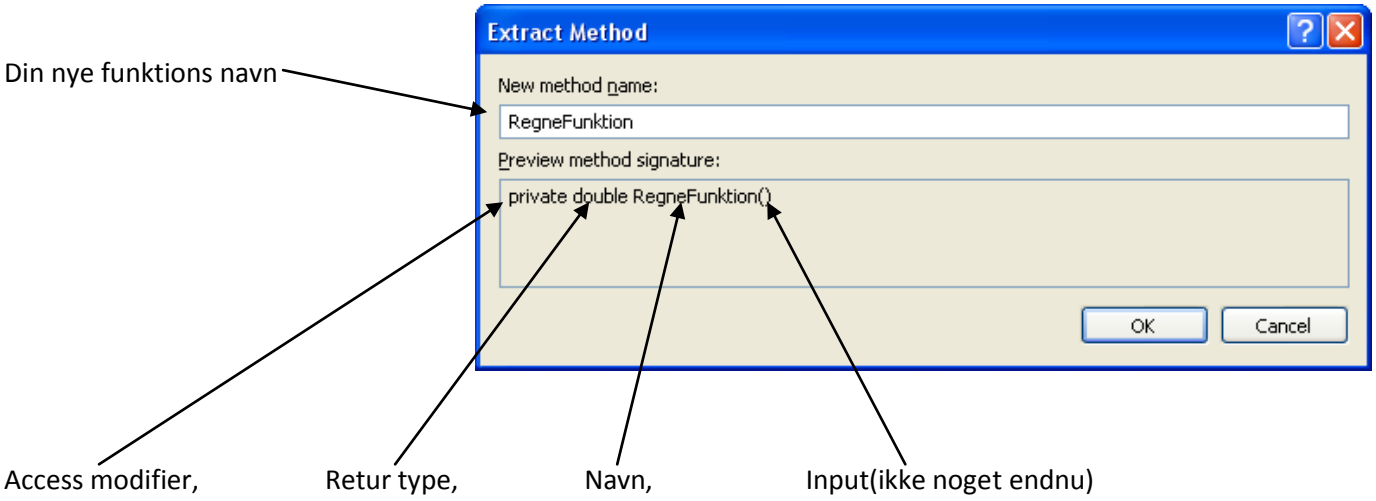
Start med at markerer koden i din btnPlus_Click eventhandler som vist herunder:

```
private void btn_Plus_Click(object sender, RoutedEventArgs e)
{
    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();
}
```

Højreklik på det blå-markerede tekst og vælg ”refactor” og herefter Extract Method:



Billedet på næste side viser wizarden der spørger efter et navn til din nye method. Jeg har meget originalt valgt at kalde den for RegneFunktion…



Når du trykker OK vil Visual Studio modificere din kildekode som herunder:

```
        private void btn_Plus_Click(object sender, RoutedEventArgs e)
        {
            double resultat = RegneFunktion();
            tbResultat.Text = resultat.ToString();
        }

        private double RegneFunktion()
        {
            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...");
            }
            return resultat;
        }
```

Som du kan se har du fået en funktion ved navn ”RegneFunktion” og alt logikken er flyttet herned. I din btn_Plus_Click funktion er koden til gengæld udskiftet med et kald til den nye funktion:

```
double resultat = RegneFunktion();
```

Hvis du prøver at kompilerer og køre dit nye program vil du se at ingen af funktionaliteten er ændret. Vi har simpelthen bare flyttet rundt på noget kode…

Men dette ændrer vi på nu! Vi skal nemlig have ændret koden til også at overfører et regnetegn når det kalder RegneFunktion. Det gør vi ved at ændre method signaturen som vist herunder:

```
private double RegneFunktion(char regneTegn)
                                    en char som input
```

Når vi kalder "RegneFunktion" fra vores btnPlus_Click eventhandler (og alle andre funktioner for den sags skyld) skal vi for eftertiden, altid huske at have et regnetegn med som vist herunder:

```
double resultat = RegneFunktion('+');
```

I RegneFunktion kan vi så efterfølgende benytte den overførte variabel som vist herunder:

```
if (regneTegn == '+')
{
    //Gør noget!
}
```

Nu kan du, som vist ovenfor, endelig få lavet den smule branching code der skal til for at skelne imellem de forskellige regnetegn:

Den endelige RegneFunktion kunne se ud som nedenfor:

```
private double RegneFunktion(char regneTegn)
{
    double tal1 = 0;
    double tal2 = 0;
    double resultat = 0;

    try
    {
        tal1 = Convert.ToDouble(tbTal1.Text);
        tal2 = Convert.ToDouble(tbTal2.Text);
    }
    catch (Exception exc)
    {
        MessageBox.Show(exc.Message, "Hov...");
    }

    if (regneTegn == '+')
    {
        resultat = tal1 + tal2;
    }

    return resultat;
}
```

Tilføj de sidste knapper til plus, minus og gange og tilføj den nødvendige kode for at få tingene til at virke.