C# WPF – (New) simpel Data-binding

Data-binding er en metode der med fordel kan anvendes til at knytte værdier i en *class* til et element på brugergrænsefladen. I dette eksempel, bliver der gennemgået hvordan man laver *Data-binding* mellem en *TextBox* og en *Property* i en *class*.

For at løse opgaven, skal du benytte følgende solution *WpfAppDataBinding* som du finder i mappen [Opgaver](https://intranet.aspit.dk/Fagets_Univers/Softwarekonstruktion/Documents/Materialer/S1/Brugerflader/Cs%20WPF%20-%20Simpel%20Data-binding/WpfAppDataBinding.zip). Du skal kopier ZIP-mappen WpfAppDataBinding til din kodemappe på C-drevet og pakke den ud i din mappe med WPF projekter.

Dette lille simple program, har kun til formål, at demonstrer anvendelsen af simpel Data-binding, hvorfor funktionaliteten i programmet ikke har den store anvendelse i virkeligheden.

**Brugergrænsefladen**Brugergrænsefladen er i forhold til programmet, tilsvarende simpel.

Der er indsat to *GroupBox* elementer på formen – *Udregning* og *Lige eller ej.*

*Udregning* indeholder en *TextBox (textBoxNewNumber)*, en *Label (labelHT1)* og en *Label (labelResult).*

*Lige eller ej* indeholder en *Label (labelResText).*

I *TextBox (textBoxNewNumber)* kan brugeren indtaste et tal, som straks bliver ganget med tallet 1387 og der bliver kontrolleret og resultatet er et lige tal (Heltal der divideres med 2 og ikke giver nogen decimaler).

Resultatet af udregningen skal vises i *Label (labelResult)*.

Når udregningen er gennemført, skal der udføres en kontrol af, om tallet er et heltal og resultatet skal vises i *Label (labelResText)*.

|  |  |  |
| --- | --- | --- |
|  |  |  |

**2-lags arkitektur**

Applikationen er opbygget så den overholder betingelserne for en 2-lags arkitektur hvor der benyttes GUI og BIZZ.

|  |  |
| --- | --- |
| GUI laget indeholder:   * MainWindow.xaml * MainWindow.xaml.cs   BIZZ laget indeholder:   * ClassBIZ.cs   Repository laget indeholder:   * ClassNotify.cs * ClassTextRes.cs * ClassCalckRes.cs |  |

Opret referencerne mellem projekterne: GUI =>BIZ&Repository og BIZ=>Repository

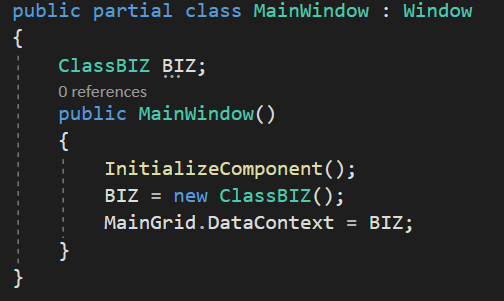
Opgaven

Du skal følge den nedforstående beskrivelse, hvor der ligeledes vil være forklaringer til de enkelte step.

**Step 1. – Knyt class filerne via oprettelse af instanser**

I class *MainWindow.xaml.cs*, skal du knytte class *MainWindow.xaml.cs* til class *ClassBizz.cs*.

Det gøres på følgende måde:



Lige over *public MainWindow()* skriver du:

*ClassBIZ BIZ;*

Derved laver du en tom instans af *ClassBIZ*

Inde i *public MainWindow()* skriver du:

*BIZ = new ClassBIZ();*

Derved bliver *BIZ* initialiseret, så du fra *MainWindow.xaml.cs* kan tilgå public elementer i class *ClassBIC.cs* via instansen *BIZ*

|  |
| --- |
|  |

Ved at benytte det *x:Name* der er givet til det overordnede grid og skrive følgende:

*MainGrid.DataContext = BIZ;*

knytter man *Binding* for alle objekter i *MainGrid* til *BIZ*.

**Step 2. Opret ClassNotify i project Repository**

Nu skal du i projekt Repository oprette en class med navnet ClassNotify.cs. Denne class skal håndtere og sikrer at ændringer i div. Propertys, opdateres i GUI via databinding.

I ClassNotify.cs skal vi benytte en reference til .Net classén *ComponentModel,* dette opnår vi at danne en reference med kommandoen *using System.ComponentModel;.*

I ClassNotify.cs skal der oprettes en direkte nedarvning fra *INotifyPropertyChanged,* dette gøres ved at man ud for classéns navn skriver ” *: INotifyPropertyChanged”.* [Her kan du læse lidt om INTERFACE](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/)

Nu skal der oprettes en lokal instans af *PropertyChangedEventHandler*, som knytter sig til classén og som navngives til *PropertyChanged* .

|  |
| --- |
|  |

Til sidst skal der laves en metode der skal sikrer at opdateringerne bliver ’broadcastet’ til hele applikationen. Dette gøres ved at implementere følgende metode i ClassNotify.cs.

|  |
| --- |
|  |

Denne metode modtager en string som indeholder et ID med et navnesammenfald til den *property* der blevet opdateret og til det element på brugergrænsefladen der skal knyttes til via dens *{Binding strCalckNumber}.*

Det modtagne ID udløser to handlinger:

1. Der foretages et kald til den property med samme navn for at hente dens ”data”
2. Den ’broadcaster’ ID og data til alle classés der er knyttet via *DataContext* angivelsen

For at oprette denne knytning mellem brugergrænsefladen og specifikke *property* i en given class, skal man i *konstructoren* til classén der knyttet til brugergrænsefladen, angive denne knytning.

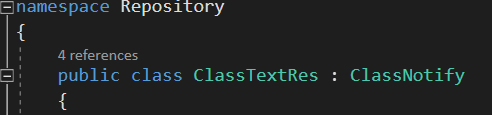
Hvis der på formen forefindes flere objekter, som f.eks. *GroupBox*, kan hvert objekt knyttes individuelt til hver sin *class* og dermed også til forskellige samlinger af data. Et objekt kan kun knyttes til en *class*, så hvis der skal knyttes til *property´s* fra forskellige *class´s*, skal de respektive elementer på brugergrænsefladen samles i selvstændige objekter. Det kan f.eks. være elementer som *GroupBox* eller *Grid*.

Til denne applikation benyttes der to *GroupBox´s* som knyttes til samme *class,* ved at knytte et overordnet objekt som indeholder begge *GroupBox’s* – nemlig *Maingrid.* I *konstructoren* der knytter sig til brugergrænsefladen, sikres denne knytning på følgende måde:

|  |
| --- |
|  |

**Step 3. – Opret ClassCalckRes & ClassTextRes i projekt Repository**

I projekt Repository skal der oprettes to nye class filer: *ClassTextRes.cs* og *ClassCalckRes.cs.*

Først skal *ClassTextRes.cs* oprettes og det skal angives at *ClassTextRes* direkte nedarver fra *ClassNotify,* på følgende måde:

Nu skal der laves et passende antal *metoder* og *property*´s til *ClassTextRes*, nemlig en af hver.

Der skal oprettes et *property,* med et tilhørende *private field* hvor navnet kun differencerenes med et ’\_’

|  |
| --- |
|  |
|  |

Bemærk at den tekststreng der parameteroverføres til metoden *Notify* er identisk med navnet på *property´n*, samme reference navn skal benyttes på det element i brugergrænsefladen der skal modtage værdien.

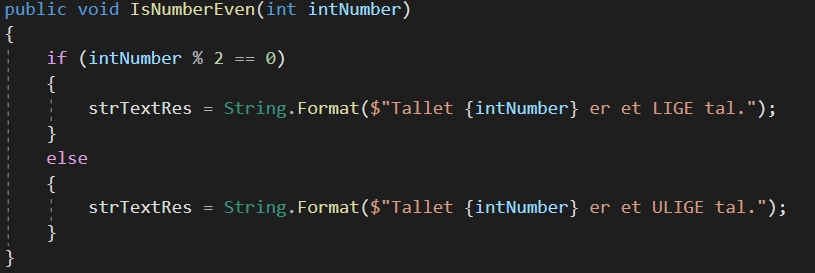
Hvert *property* der skal knyttes til brugergrænsefladen via Data-binding, består et *private* *field* og et *public* *property*. Den *property* der er public, er indgangen til at opdaterer værdien i det *private field*. Det *private field* benyttes som container der skal holde den sidst angivne værdi og kan derfor også benyttes som kontrolværdi for om det er nødvendigt at opdatere værdien.

Samme struktur kan udnyttes til at sikrer at der ikke kan indtastes og indsættes ulovlige værdier i en *property* og dermed også i det tilknyttede objekt på brugergrænsefladen, dette kommer vi nærmer ind på senere.

Da det ikke skal være muligt, at indsætte værdier i *strTextRes* fra metoder uden for *ClassTextRes*, gøres *set* til *private*.

Herefter skal der oprettes en metode med navnet *IsNumberEven,* hvis eneste formål er, at den ud fra en parameter af datatype int, skal afgøre om det parameteroverførte tal er lige eller ulige.

Herefter skal den sætte værdien i *strTextRes* til en tekst der viser tallet og om det er et lige eller ulige tal.



I *class*´s *constructor* skal/kan der angives en standard tekst for hvert *property*. Dette gøres på følgende måde:

|  |
| --- |
|  |

Så er *ClassTextRes* færdig.

Opret *ClassCalckRes* i projekt *Repository*.

Denne class skal også nedarve *ClassNotify*, lige som i *ClassTextRes*.

I denne *class* skal der oprettes 3 *propertys* med tilhørende *fields* og en *metode*.

|  |  |
| --- | --- |
| Der skal oprettes et *property* af datatypen *string* og navnet *strCalckRes*, hvor *get* er *public* og *set* er *private*.  Og et *property* af datatypen *string* og navnet *strCalckNumber* hvor både *get* og *set* er public.  Det sidste *property* skal være af datatypen *ClassTextRes*, hvilket jo er den class du lige har lavet. Derfor kalder man også sine class-filer for *simple datatyper*, da de kan anvendes som sådan.  Bemærk at der skal foretages kald til *Notify* metoden i alle *property* og vær særdeles opmærksom på, at det er nøjagtig samme navn der skrives i *Notify* parameteren, som navnet på det *property* den står i.  Bemærk, at der i *property* *strCalckNumber*, foretages et kald til en metode *CalckNumber*.    Nu skal du skrive metoden *CalckNumber* der skal  modtage et parameter af datatypen *string*.  Det modtagede parameter sendes til metoden  *IsNumberEven* som *parameter*.  Metoden *IsNumberEven* er tilgængelig gennem  parameteren *textRes* som er af datatypen  *ClassTextRes*, som indeholder metoden. |  |

**Step 4 - ClassBizz – Property og Constructor**

*ClassBiz* skal også nedarve *ClassNotify*.



Nu skal der laves et passende antal *property*´s og *metoder* til class *ClassBizz.*

Det bliver dejligt nemt, da der kun skal oprettes en *property*, nemlig *calckRes* af datatypen *ClassCalckRes.*

|  |
| --- |
|  |
|  |
|  |
|  |

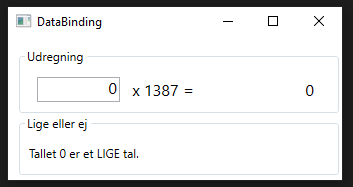
I alle *class*´s *constructor* i hele programmet, skal der angives en standard værdi for hvert property og hvor der anvendes en simple datatype man selv har skrevet skal den initialiseres med en ny instans. Dette gøres på en af følgende måder:

|  |
| --- |
|  |

**Step 5 – Binding af elementerne på brugergrænsefladen**

Der skal nu laves *binding* til de tre elementer i XAML koden, som skal knyttes til en Property i koden.

De tre elementer der skal knyttes ved brug af binding er:



En TextBox

En Label

En Label

De to label skal kun kunne ”læse” værdierne og knyttes på følgende måde:

|  |
| --- |
| I stedet for at angive en tekst i Content, skriver man: ”{Binding XXXXXXX}”.  De angivne X’er skal repræsentere en sti til det Property der holder den data objektet skal vise.  Når der står *calckRes.strCalckRes* betyder det, at data skal hentes fra property *calckRes* som er af datatypen *ClassCalckRes*, som indeholder et property *strCalckRes*.  Når der står *calckRes.TextRes.strTextRes* betyder det, at data skal hentes fra property *calckRes* som er af datatypen *ClassCalckRes*, som indeholder et property *textRes* som er af datatypen *ClassTextRes* der indeholder et property *strTextRes.*  Tekstboksen skal bindes på en lidt anden måde. Her har vi brug for at ændringer i tekstboksen overføres til vores kode og bliver behandlet.  Derfor skal man i stedet skrive: ”{Binding XXXXX , Mode=TwoWay , UpdateSourceTrigger=PropertyChanged}”  De angivne X’er skal repræsentere en sti til et property som beskrevet ovenfor.  Med angivelse af *Mode* som *TwoWay*, angiver jeg at opdateringen skal foretages fra både brugergrænsefladen og fra koden.  Med angivelse af *UpdateSourceTrigger* til *PropertyChanged*, angiver jeg hvornår der skal foretages en opdatering, altså når der ændres i teksten i tekstboksen, eller når der sker ændringer i det property den er knyttet til. |
|  |

|  |
| --- |
|  |

Så skulle alle elementer være på plads og du er klar til at teste – god fornøjelse ;O)

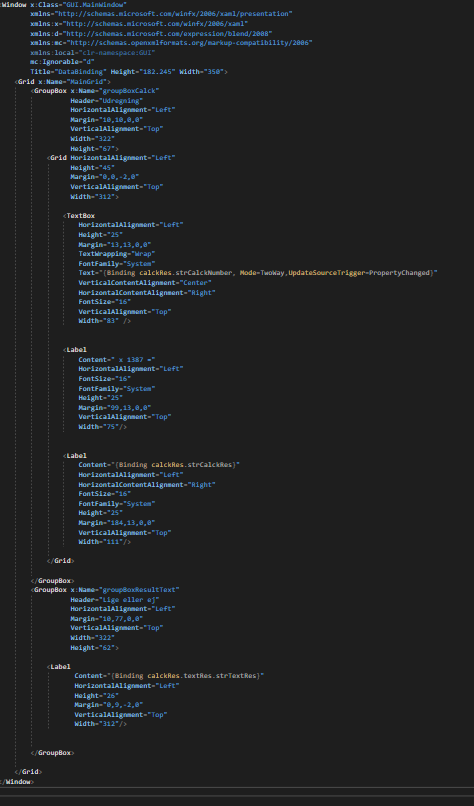
**Test opgave:**

Gennemfør en test af programmet for fejl.

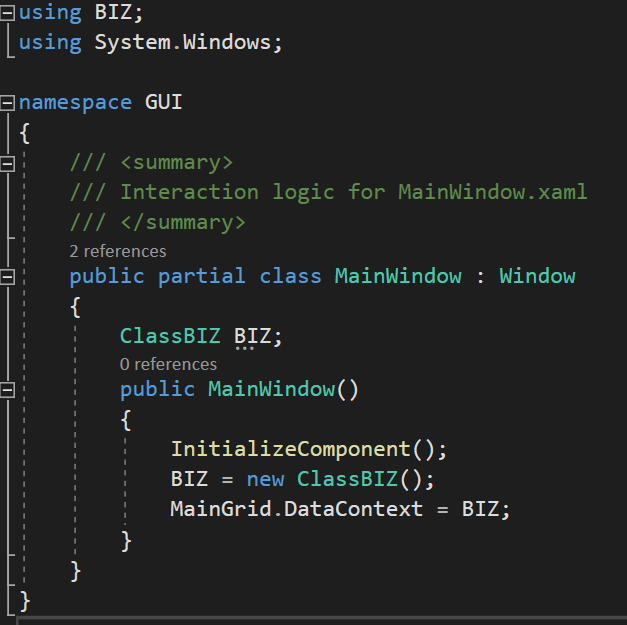
Beskriv eventuelle fejl og kom med et forslag til hvordan det rettes.

Samlet kode:

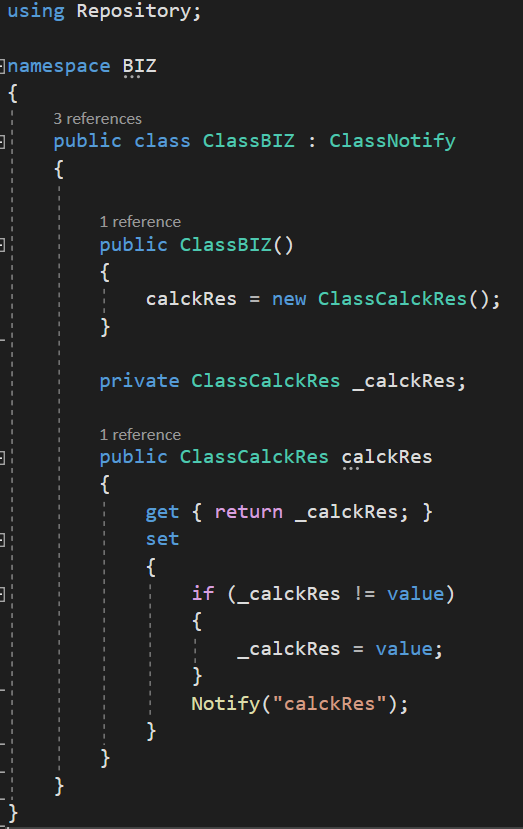
**MainWindow.xaml:**

****

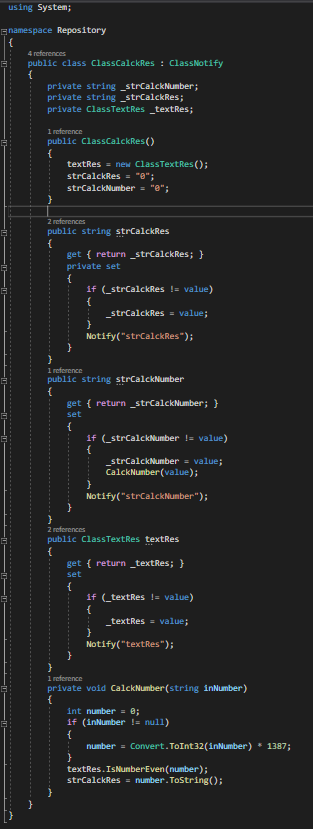
**MainWindow.xaml.cs:**

****

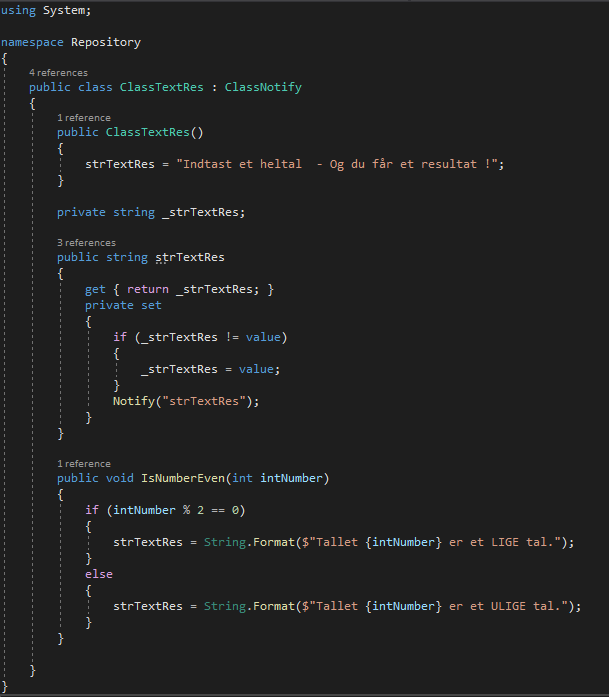
**ClassBIZ.cs:**

****

**ClassCalckRes.cs:**

****

**ClassTextRes.cs:**



**ClassNotify.cs:**

