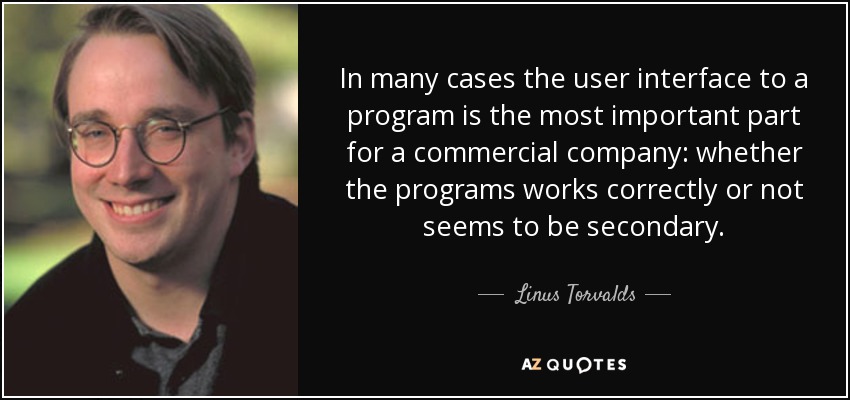
# WPF And MVVM 2

|  |  |
| --- | --- |
| **Læringsmål** | Du kan:   * **1Pf1**: anvende centrale metoder til at specificere og konstruere algoritmer […] * **1Pf2**: anvende centrale faciliteter i programmeringssproget til realisering af algoritmer, designmønstre, abstrakte datatyper, datastrukturer, designmodeller anvende centrale faciliteter i programmeringssproget til realisering af algoritmer, designmønstre, abstrakte datatyper, datastrukturer, designmodeller og brugergrænseflader. |
| **Forventet læringsudbytte [SOLO]** | Programmeringssporet:   * [Unistrukturel] Du kan identificere enkelte kendetegn ved **applikationstypen** GUI applikation – WPF, XAML, MVVM, UI-kontroller (ListBox, StackPanel, Grid, TextBox, Label, Button), databinding (inkl. interfacet INotifyPropertyChanged og ObservableCollection<T>-typen) * [Unistrukturel] Du kan identificere enkelte kendetegn ved **Arkitekturmønstre** – Model-View-ViewModel (MVVM) |
| **Din forberedelse** | Programmeringssporet:   * Læs (genlæs) nedenstående sektioner fra [The complete WPF tutorial](https://www.wpf-tutorial.com/):   + **List Controls** (fokusér på [ListBox](https://www.wpf-tutorial.com/list-controls/listbox-control/), men skimlæs gerne om de to andre)   + **Panels** (fokusér på [intro’en](https://www.wpf-tutorial.com/panels/introduction-to-wpf-panels/), [StackPanel](https://www.wpf-tutorial.com/panels/stackpanel/) og afsnit 6. til 11. om [Grid](https://www.wpf-tutorial.com/panels/grid/), skimlæs evt. det resterende i sektionen)   + **Databinding** (fokusér på de første 6 afsnit fra [int ro’en](https://www.wpf-tutorial.com/data-binding/introduction/) til og med [Responding to changes](https://www.wpf-tutorial.com/data-binding/responding-to-changes/) og evt. det sidste afsnit [Debugging data bindings](https://www.wpf-tutorial.com/data-binding/debugging/)) * Databinding:   + Evt. gense databinding-videoer fra forrige opgave   + Databind 5 – ListBox (video: 13:26; se planen i ItsLearning)   + Databind 6 – ListBox Selection (video: 10:22; se planen i ItsLearning)   + [Data binding of ListBox and string list in C#](http://wangxinliu.com/tech/program/WPF-DataBinding/) (fokusér på brugen af ObservableCollection<T> og ListBox-property’en SelectedItem )   + [List Vs ObservableCollection in WPF](https://www.youtube.com/watch?v=YW3gq-zyzoE) (video: 13:35) * Evt. genlæs: [Explain **INotifyPropertyChanged** In WPF - MVVM](https://www.c-sharpcorner.com/article/explain-inotifypropertychanged-in-wpf-mvvm/) |

I denne opgave fortsætter du med at se på databinding, men nu til en samling (Collection) af objekter og ikke et enkelt objekt. Du skal benytte dig af en ListBox-kontrol, som netop kan vise samlinger af objekter, og sørge for at skabe en databinding til dit ViewModel-objekt.

# Dagens ord

S

# Øvelse 1: Terminologi

Del teamet op i mindre grupper og brug **Ordet rundt** til at reflektere over begreberne ”MVVM”, ”XAML”, ”Data binding”, ”ListBox”, ”Grid” og ”StackPanel”. Sørg for at alle får mulighed for at tale.

*Tidsramme: 20 minutter*

**MVVM** stands for model-view-viewmodel and is a designpattern used for developing WPF applications where view is the part of the application that creates the UI, the model is the business logic and the viewmodel is responsible for presenting the needed data from the model to the view, View knows about viewmodel, viewmodel knows about model and none of the layers know about any of the layers above, e.g. the model doesn’t know about the viewmodel. It’s an example of layered architecture where the business logic code and the UI code is separated. This has benefits as developers being able to work on the layers separately and the model(business logic) can be used for other applicationtypes, e.g. ios app, webpage. Where the architecture and language for the UI is different. As opposed to MVC where eventhandlers is used alot, in MVVM databinding is used instead where applicable.

**XAML** is a UI markup language used to create the GUI and databindings for a WPF application.

**Data binding** is used to create synchronicity between two datapoints, often from the UI to the ViewModel, so that when one of the datapoints is changed the other one is too. The default binding mode is TwoWay in which both the source and the target (the data binding is often defined in the target or UI point) are updated the other one changes. Then there is OneWay which is from source to target or reverse OneWay which is from target to source.

**ListBox** is a WPF control in which a list of items is displayed and selectable, often connected to a list/observableCollection.

**Grid** is a control in which rows and columns can be defined to contain other controls.

**StackPanel** is a panel used to stack contained controls vertically by default or horizontally if defined. As opposed to WrapPanel(default is horizontally) where when the parent control runs out of space, the StackPanel doesn’t automatically continue of the next row/column but the truncated controls are not rendered.

**Benyt parprogrammering til alle følgende øvelser.**

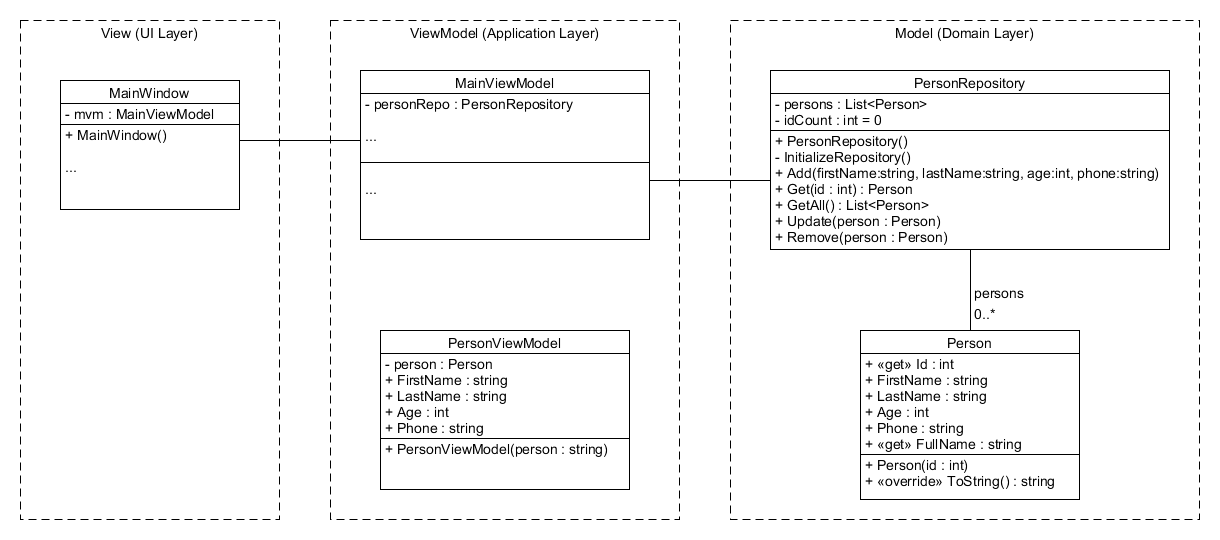
# Øvelse 2: WPF og MVVM

Når man udvikler en WPF-applikation, kan man ikke komme uden om MVVM, XAML og databinding. Disse begreber/teknikker er på forhånd designet til at fungere sammen, og denne opgave vil fortsætte med at afdække flere af disse sammenhænge ved at tage udgangspunkt i databinding.

## Øvelse 2.1: Klargøring

Udfør følgende:

* Hent den udleverede zip-fil ”Ex28-WPFAndMVVM2.zip”, der ligger i Git materiale-mappen for programmering (benyt git pull i materialemappen)
* Pak zip-filen ud, og åbn løsningen ”WPFAndMVVM2”, som er et projekt af typen ”WPF Application” med de 4 klasser MainWindow, MainViewModel, PersonRepository og Person som vist i følgende MVVM-lagdelte DCD (bemærk at PersonRepository nu er placeret i Model-laget):



Denne delvise WPF-løsning implementerer mere eller mindre hvert lag i en MVVM-arkitektur:

* **View-laget**: er defineret i XAML med et GUI-mønster (aka et master/detail-mønster)
* **ViewModel:** er kun rudimentært defineret
* **Model-laget**: indeholder en Person-klasse (er næsten færdigimplementeret) og en PersonRepository-klasse (færdigimplementeret)

Du skal sørge for at etablere den manglende XAML databinding mellem View-laget og ViewModel-laget, som mangler i løsningen, hvilket vil påvirke hvert lag efterhånden, som du får løst alle øvelserne i det følgende.

**Bemærk**: Alle klasser i View-laget (MainWindow og andre du laver) kender på ingen måde til klasserne i Model-laget, dvs. Person- eller PersonRepository-klasserne; View-laget kender kun til ViewModel-lagets klasser. Tilsvarende kender ViewModel-laget kun til Model-lagets klasser og ikke til dem i View-laget. Model-laget kender hverken til ViewModel-laget eller View-laget.  
**Disse krav skal opretholdes gennem hele din løsning jf. kravene i MVVM-arkitekturmønstret.**

## Øvelse 2.2: Inspicér den udleverede kode

Som det første:

* Kompilér løsningen, og kør den, så du er sikker på, at alt virker, inden du starter.

Der er ingen data at se i GUI’en på den kørende løsning endnu, men den udleverede kode implementerer et simpelt Master/Details-designmønster, som du nok kender fra File Explorer i Windows med en oversigt på venstre side (ListBox), hvor man kan vælge en overordnet visning af et punkt og en detaljeret visning i højre side af dette punkt (de 4 tekstbokse). Mellem oversigten og den detaljerede visning er der en ”splitter”, som gør det muligt at vise mere eller mindre af venstre/højre side.

**VIGTIGT**: Luk programmet, og udfør følgende inspektion:

* Undersøg først løsningen i Solution Explorer og se, hvorledes den er struktureret for at kunne repræsentere 3-lags arkitekturen i MVVM.
  + Overvej hvorfor den er organiseret, som den er i løsningen, og hvordan lagene er repræsenteret både fysisk med foldere og logisk med namespaces (se i klassefilerne).
  + Bemærk, hvordan kravene til MVVM-lagdelingen nævnt foroven på forhånd er opretholdt ved brug af namespaces.
* Undersøg dernæst de to klasser i Models-folderen. Du vil kunne genkende opbygningen i de to C#-filer ud fra tidligere opgaver:
  + Selvom **PersonRepository-klassen** er helt færdigimplementeret og ikke bør ændres overhovedet, så kig den nøje igennem. Alle CRUD-operationer er implementeret; måske er der inspiration at hente på en mulig implementering af et Repository-mønster til et fremtidigt projekt.  
    Bemærk at PersonRepository initialiseres med data i constructor’en (via kald til InitializeRepository()-metoden). Disse data, som er tilfældigt genereret, hentes fra en ekstern kommasepareret tekstfil Persons.csv i ”bin\Debug\ net5.0-windows”-folderen i projektet.
  + **Person-klassen** er rimelig nem at overskue; der burde ikke være nogen overraskelser her.  
    Bemærk: Denne klasse kommer du nok til at ændre lidt i, efterhånden som du kommer i gennem de næste øvelser.
* ViewModels-folderen indeholder kun en ViewModel til at starte med, nemlig **MainViewModel**:
  + MainViewModel-klassen indeholder kun et privat felt **personRepo** til at repræsentere den interne datastruktur i MainViewModel og lige så vigtigt forbindelsen fra ViewModel-laget til Model-laget, nemlig PersonRepository-klassen.
* Undersøg dernæst XAML-filen **MainWindow.xaml** i Views-folderen. Observér, hvordan den er opbygget med et Grid indeholdende en ListBox, en GridSplitter og et Grid med nogle TextBox-kontroller og Labels. Bemærk også StackPanel-kontrollen under den venstre ListBox; den får du brug for senere i opgaven. Tjek code-behind-filen **MainView.xaml.cs**, og bemærk at der ingen code-behind er.

Så kig på lagene, klasserne og koden i dem, så du er klar over og forstår udgangspunktet for opgaven.

## Øvelse 2.3: Dan forbindelse mellem View-laget og ViewModel-laget

Første trin er at sørge for forbindelse mellem MainWindow-klassen i View-laget til en instans af MainViewModel i ViewModel-laget, så du efterfølgende kan benytte dig af databinding.

Udfør følgende:

* Implementér forbindelsen til MainViewModel i MainWindow som vist i forberedelsen.
* Kør programmet, og tjek om MainViewModel-objektet er instantieret og initialiseret.   
  *Vink: med lidt debugging kan du tjekke om personRepo i MainViewModel faktisk indeholder data.***Bemærk: du må godt ændre i MainWindow-constructor’en i code-behind-filen, men du må IKKE oprette event-handlers i MainWindow-klassen til at løse opgaven.**

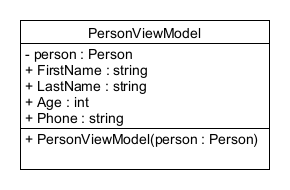
## Øvelse 2.4: Dan ViewModels for hver person

En ViewModel’s rolle er primært at iscenesætte data fra Model-laget, så det egner sig til præsentation i GUI’en. Det betyder, at kun de data, der er nødvendige for præsentation, skal repræsenteres i en ViewModel. Du har allerede en MainViewModel, som giver adgang til alle personer i PersonRepository. Men det er ikke nok.

Du skal også håndtere hvert enkelt Person-objekt (fra Model-laget via personRepo) og sørge for, at kun de informationer for en person, der skal præsenteres i GUI’en, bliver gjort tilgængelige til GUI’ens kontrollere.   
Derfor skal du også implementere en PersonViewModel-klasse, som komplementerer Person-klassen, til at håndtere personens iscenesættelse af data. Dvs. for hver Person-objekt i personRepo skal der instantieres et PersonViewModel-objekt. Dette indebærer flere trin.

Udfør følgende:

* Implementér først en PersonViewModel-klasse i C# som overholder designklassen:



Bemærk følgende iscenesættelse i PersonViewModel:

* + En PersonViewModel-instans, som eksisterer og skal bruges i ViewModel-laget, refererer til dets tilsvarende Person-objekt i Model-laget via det private **person**-felt. Dette person-felt er altid privat, fordi View-laget ikke må få adgang til Model-laget. PersonViewModel-instansen har dog brug for forbindelsen, hvilket overholder MVVM-kravene.
  + View-laget (GUI’en) har derimod brug for at kunne se personens informationer såsom fornavn, efternavn, alder og telefonnummer, hvorfor **FirstName**, **LastName**, **Age** og **Phone** alle er public og (meget vigtigt) properties jf. den anvendte konvention.
  + View-laget skal ikke have adgang til Person-objektets **Id**-property, da det er helt uinteressant for brugeren.  
    *Hermed er Person-klassens data iscenesat i PersonViewModel*.
* Vær opmærksom på implementeringen af PersonViewModel-klassens constructor, som skal sørge for at initialisere alle klassens medlemmer.

Med PersonViewModel-klassen parat skal du nu sørge for at implementere en datastruktur i MainViewModel, der rummer alle PersonViewModel-objekter og gør dem tilgængelige for MainWindow i dit View-lag.

Udfør følgende:

* I MainViewModel opret en public property **PersonsVM** af typen **List<PersonViewModel>** til at repræsentere PersonViewModels for alle Person-objekter.
* Initialisér PersonsVM-property’en i MainViewModel’s constructor, dvs. sørg for i PersonsVM at oprette et korrekt initialiseret PersonViewModel-objekt for hvert Person-objekt i personRepo.
* Kør programmet, og tjek via debugging, at PersonsVM-listen er korrekt initialiseret med et PersonViewModel-objekt for hvert Person-objekt i personRepo.

Du har nu udført de indledende trin til at databinde og udfylde din ListBox i GUI’en med den relevante personinformation.

## Øvelse 2.5: Vis data i ListBox (Master-delen)

Du skal nu sørge for, at GUI’ens ListBox (defineret i XAML) viser det fulde navn (fornavn + efternavn) for alle de personer i PersonsVM i MainViewModel, dvs. ViewModel-iscenesættelsen af personRepo. Der er 100 personer i datasættet.

Dette indebærer at:

* Du skal udelukkende konfigurere og indsætte **databindings** i ListBox-kontrollen i XAML-filen, så den viser det fulde navn for alle personer i datasættet (du må ikke bruge Person.ToString()-metoden).
* Kør programmet, og tjek om ListBox-kontrollen bliver korrekt udfyldt.

## Øvelse 2.6: Vis data i tekstbokse (Details-delen)

Når du klikker på en bestemt person i ListBox’en, så skal alle denne person’s oplysninger vises i de fire tekstbokse på højre side, dvs. på Details-siden.

Dette indebærer at:

* Du skal i MainViewModel registrere, hvad der er udvalgt i ListBox’en.
* Du skal via databindings i XAML-filen sørge for, at de fire tekstbokse kan hente sine data via det registrerede valg i MainViewModel af den valgte person i ListBox’en.  
  *Vink: det kræver nok en ny property i MainViewModel, der afspejler den valgte person i ListBox-kontrollen (se i forberedelsen for inspiration).*

## Øvelse 2.7: Redigér data i Details-delen

I Details-delen af brugergrænsefladen kan du redigere en person oplysninger via de 4 tekstbokse. Hvis du ændrer fornavn og/eller efternavn på en udvalgt person i Details-delen, så ændres det fulde navn ikke øjeblikkeligt på tilsvarende plads i ListBox-kontrollen i Master-delen.

Udfør følgende:

* Du skal nu sørge for, at en ændring i for-/efternavn på en udvalgt person også afspejles øjeblikkeligt i personens fulde navn på tilsvarende plads i ListBox’en.

# Øvelse 3: Tilføj eller slet person

Du vil som det næste gerne kunne tilføje nye personer til dit repository og slette eksisterende personer derfra. Det skal du se på nu. x

## Øvelse 3.1: Klargøring

Udfør følgende:

* Udvid brugergrænsefladen med 2 nye knapper med titlerne ’New’ og ’Delete’, så de er placeret inden i StackPanel-kontrollen lige under ListBox-kontrollen.   
  Giv New-knappen navnet btnNewPerson og Delete-knappen navnet btnDeletePerson.

Knappernes funktionalitet er:

* New-knappen tilføjer en ny person til systemet og viser personen som den valgte i ListBox-kontrollen.
* Delete-knappen fjerner den valgte person fra systemet og fra ListBox-kontrollen.

## Øvelse 3.2: Opret Person

Oprettelse af en ny person via GUI’en kræver lidt arbejde både i MainWindow- og MainViewModel-klasserne. Den nye person, der skal oprettes, har per default oplysningerne ”Specify FirstName” som fornavn, ”Specify LastName” som efternavn, 0 som alder og ”Specify Phone” som telefonnummer:

### MainViewModel-klassen:

Udfør først følgende i MainViewModel-klassen:

* Opret en ny metode til at håndtere GUI-operationen at tilføje en default person med signaturen:

**void AddDefaultPerson()**

Metoden skal sørge for at:

* + PersonsVM og personRepo i MainViewModel er opdateret med de korrekte informationer for den nye person ud fra metodens parametre.
  + Den nye tilføjede person også er den valgte person i GUI’ens ListBox.   
    Men da AddDefaultPerson()-metoden i MainViewModel jo ikke har direkte adgang til MainWindow i View-laget jf. MVVM-kravene, kan du i stedet for sætte den property, du tilføjede i øvelse 2.6 til at afspejle den valgte Person i ListBox-kontrollen.

### MainWindow-klassen:

Udfør dernæst følgende i MainWindow-klassen:

* Dan en Click-eventhandler for New-knappen i code-behind
* Kald AddDefaultPerson()-metoden i MainViewModel, som du oprettede før.

### Afprøv:

Kør nu programmet, og afprøv om det virker. Benyt debugging.

* Bliver ListBox-kontrollen opdateret med den nye person?   
  Hvis ikke, hvad tænker du kunne være problemet?

**Vink**: Brug *ObservableCollection<T>* i stedet for *List<T>* til PersonsVM-property’en i MainViewModel.   
Læs i forberedelsen, hvorfor det er en god idé.

* Prøv igen, og reflektér over, hvad der nu sker. Det skulle gerne virke.

Men er den nye tilføjede person nu den valgte i ListBox? Du har sat property’en for den valgte person, men ændringen er måske ikke slået igennem. Det er der råd for:

* Lad MainViewModel implementere INotifyPropertyChanged-interface (se Ex27-WPFAndMVVM for inspiration), og sørg for at kalde OnPropertyChanged(”…”) når du ”**setter**” din property for valgt person.
* Afprøv, og ret til, hvis det ikke virker.

**Vink**: Hvis du gerne vil have, at den nye person blive gjort synlig i din ListBox, så snart den er indsat, så kan du kalde ListBox-metoden ScrollIntoView() lige efter kaldet til AddDefaultPerson i din event-handler. Angiv Listbox-kontrollerens SelectedPerson-property som parameter til ScrollIntoView.

## Øvelse 3.3: Slet Person

Sletning af den valgte person i GUI’ens ListBox kræver også lidt arbejde både i MainWindow- og MainViewModel-klasserne samt PersonViewModel-klassen.

### MainViewModel-klassen:

Udfør først følgende i MainViewModel-klassen:

* Opret og implementér en ny metode til at slette en valgt person med signaturen:

**void DeleteSelectedPerson()**

Metoden skal sørge for at:

* + Den (i ListBox-kontrolleren) valgte person slettes fra personRepo (se PersonViewModel-klassen forneden) og fra PersonsVM i MainViewModel.
  + Den slettede person forsvinder fra ListBox-kontrollen.

### PersonViewModel-klassen:

Det er kun denne klasse, som har adgang til den konkrete person, som skal slettes fra personRepo. Undersøg først, hvordan en person slettes fra PersonRepository-klassen via Remove()-metoden.

Udfør dernæst følgende:

* Opret og implementér en ny metode til at slette en person med signaturen:

**void DeletePerson(…)**

Der er dog brug for én parameter til metoden. Overvej, hvilken information (som denne klasse ikke allerede kender) der er nødvendig at overføre til metoden for, at person-objektet kan slettes fra personRepo.

### MainWindow-klassen:

Udfør dernæst følgende i MainWindow-klassen:

* Dan en Click-eventhandler for Delete-knappen i code-behind
* Kald DeleteSelectedPerson()-metoden i MainViewModel, som du oprettede før.

### Afprøv:

Kør nu programmet, og afprøv om det virker. Benyt debugging.

* Bliver den valgte person slettet fra personRepo?
* Bliver den valgte person slettet fra PersonsVM?
* Bliver den valgte person fjernet fra ListBox-kontrollen?
* Hvis ikke, så tilret og prøv igen.