# WPF, MVVM og Command binding

# 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: 15 minutter*

**Benyt parprogrammering til alle følgende øvelser.**

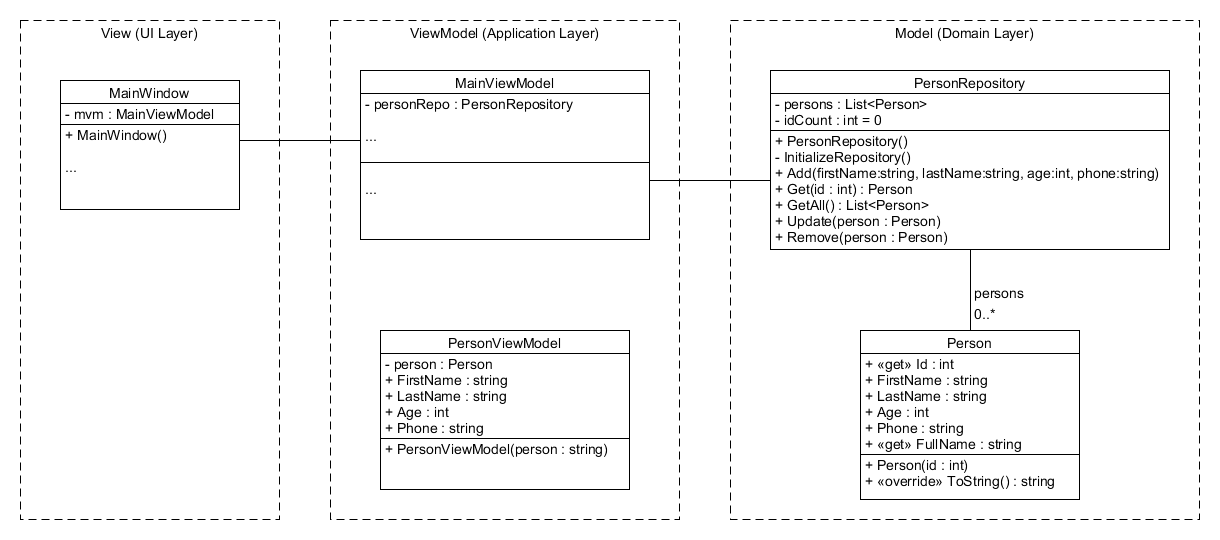
# Øvelse 2: WPF og MVVM

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 ”Uge20\_WPFAndMVVM2.zip”, der ligger sammen med denne opgave på itsLearning.
* Pak zip-filen ud, og åbn løsningen ”WPFAndMVVM2” (tryk ’Trust and Continue’ om nødvendigt), 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\ net6.0-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 igennem 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 giv dig god tid til at inspicere 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.**

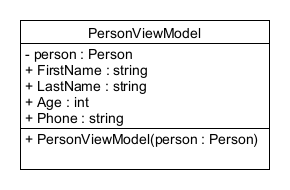
## Øvelse 2.4: Dan ViewModels for hver person

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 (breakpoint), 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 persons 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.

## 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 som standard 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 Pr26\_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.

# Øvelse 4: Terminologi

Del teamet op i mindre grupper, og brug **Ordet rundt** til at reflektere over begreberne ”Command binding”, ”ICommand”.  
Sørg for at alle får mulighed for at tale.

*Tidsramme: 15 minutter*

**Benyt parprogrammering i alle følgende øvelser**.

# Øvelse 5: Command binding i XAML

WPF har indbygget et alsidigt kommando-koncept, som ikke alene skaber en løs kobling mellem View-laget og ViewModel-laget på samme måde som databinding gør, men også via XAML centraliserer bl.a. aktivering/deaktivering af kommandoer (fra knapper, menupunkter, toolbar-ikoner, keyboard shortcuts). Command binding minimerer desuden (eller fjerner helt) kode i code-behind. Du skal i denne øvelse implementere en kommando i en ViewModel og etablere en binding til den fra en knap i XAML i View-laget.

## Øvelse 5.1: Arkitekturovervejelse

Overvej følgende i en passende CL-struktur i dit team:

* I hvilket MVVM-lag hører en kommando-klasse til?
* Hvorfor mener du, kommando-klassen hører netop til i dette lag?
* Hvilke andre MVVM-lag må en kommando-klasse kende til?

## Øvelse 5.2: Vælg en tidligere løsning

Udfør følgende:

* Nu vil du anvende command binding i stedet for en eventhandler i code-behind for at udføre knappens funktionalitet. Et krav er, at det giver mening, at knappen både kan aktiveres og deaktiveres.
* Opdatér din DCD for dit udvalgte GUI-eksempel, så din kommando-klasse indgår i den med tilhørende attributter, operationer og associationer.

## Øvelse 5.3: Opret en kommando-klasse, og etablér en command binding for den

Som du har set i forberedelsen, implementerer en kommando-klasse ICommand-interfacet og er designet til at udføre en bestemt operation (forretningslogik), som aktiveres via en UI-kontrol (f.eks. en knap) i din GUI. I MVVM er der løs kobling mellem den UI-kontrol, der aktiverer kommandoen og selve implementeringen af kommandoen i kommando-klassens Execute()-metode.

Udfør følgende for dit udvalgte eksempel (som ikke nødvendigvis ligner nedenstående kode):

* Slet din eksisterende eventhandler for din knap i code-behind, f.eks.:

private void Button\_Click(object sender, RoutedEventArgs e)

{

mvm.MyLabelText = DateTime.Now.ToString();

}

* Slet dernæst referencen til eventhandler’en i XAML-elementet for knappen  
   <Button Content="Update Label" ... Click="Button\_Click"/>  
  Dvs. slet attributten: Click="Button\_Click"

Du er nu parat til at implementere en kommando samt oprette en command binding fra din ”Update Label”-knap (eller hvad nu din kommando hedder i dit eksempel) til denne kommando.   
**Bemærk: tilstræb at code-behind filen i din løsning ikke indeholder nogen som helst kode udover din MainWindow-constructor, som sætter DataContext til en instans af MainViewModel.**

Udfør:

* Implementér en kommando (via en C#-klasse, der implementerer ICommand-interfacet), der udfører den samme operation, som din knap gjorde før. Du vælger selv navnet på C#-klassen til kommandoen.
* Opret en property i din MainViewModel, som initialiseres til en instans af din nye kommando-klasse
* I XAML opret til sidst en command binding fra din knap til din kommando-property (husk du kan også overføre kommando-parametre fra XAML).
* Kør dit program, og ret i koden, indtil det virker.

## Øvelse 5.4: Aktivér/deaktivér knap via CanExcute()

En knap kan aktiveres/deaktiveres via CanExecute()-metode i din kommando-klasse. Du skal nu sørge for at styre denne aktivering på en passende måde for dit udvalgte eksempel.

Udfør følgende:

* Overvej, hvornår din knap skal være aktiveret og deaktiveret
* Ret Execute()-metoden i din kommando-klasse, så din knap (de)aktiveres korrekt
* Ret evt. CanExecuteChanged-eventet som angivet i forberedelsesvideoen ’Commandbind 3 - Button’, hvis din knap ikke responderer som forventet

# Øvelse 6: Databinding og command binding for ListBox

Du skal i det følgende øve både databinding og command binding for en ListBox-kontrol i GUI’en.

## Øvelse 6.1: Klargøring

Udfør følgende:

* Opret en ny WPF-løsning i Visual Studio med en mappe til hvert lag i MVVM-arkitektur. Du vælger selv navnet på løsningen.
* I WPF-løsningens mappe for Model-laget, opret en Product-klasse med følgende properties:
  + Id af typen int
  + Name af typen string
  + ProductType af typen string
  + Price af typen double

Opret dernæst en passende ProductRepo-klasse til at repræsentere dine produktdata:

* Overvej, hvilket lag dit repository skal placeres i
* Initialisér ProductRepo-listen med følgende 3 produkter:
  + ”Apple” med prisen 7.95
  + ”Orange” med prisen 5.50
  + ”Banana” med prisen 8.25

I din mappe for ViewModel-laget:

* Opret en passende ProductViewModel-klasse, som iscenesætter din Product-klasse kun med de to properties Name og Price
* Opret tillige en MainViewModel-klasse med en ny property med navnet ”ProductsVM” af typen ObservableCollection<ProductViewModel> samt et privat felt (indkapsling) for din ProductRepo-klasse.

## Øvelse 6.2: Databind fra ListBox til en Collection

Udfør følgende:

* Opret nu en ListBox-kontrol i GUI’en, dvs. i View-laget
* I XAML databind ListBox-kontrollen til ProductsVM-listen
* Kør programmet, og tjek om det virker, som det skal. Ret, hvis ikke.

## Øvelse 6.3: Brug af DataTemplate i en ListBox

Måske viser ListBox-kontrollen ikke lige den mest brugervenlige tekst, så overvej hvordan du kan vise en tekst for hvert produkt, hvor både produktets navn og pris indgår (vink: det er vist noget med en DataTemplate i XAML)

* Ret koden, så ListBox-kontrollen viser en mere forståelig tekst for hvert produkt.

## Øvelse 6.4: Visning af det valgte element i ListBox-kontrollen

Du vil nu gerne kunne vælge et produkt i ListBox-kontrollen og gøre det muligt at tilgå hver af de to oplysninger for et produkt.

Udfør følgende:

* Opret to TextBox-kontroller i GUI’en (den ene over den anden) til højre for din ListBox. Den øverste TextBox skal vise navnet på et produkt og den nederste prisen på et produkt. Indsæt evt. nogle labels, så du kan se, hvad der er hvad. Det kunne se ud som nedenstående:



* Opret nu en property f.eks. med navnet SelectedProduct i din MainViewModel af type ProductViewModel.
* I XAML opret en databinding fra det udvalgte element i din ListBox (vink: SelectedItem-attributten i XAML) til ovenstående property SelectedProduct.
* I XAML etablér dernæst databindings, så navn og pris for det udvalgte produkt i ListBox-kontrollen bliver vist i de to TextBox-kontroller
* Kør dit program, og ret koden, indtil det virker.

## Øvelse 6.5: Redigering af produkt

Nu hvor du har to TextBox-kontroller for produktnavn og -pris, så vil du gerne kunne redigere navn og/eller pris via disse for et udvalgt produkt i listen.

Undersøg følgende:

* Kør dit program, og vælg et produkt i listen.
* Ret produktets navn eller pris i den tilsvarende TextBox.
* Observér om produktet ændrer navn eller pris i ListBox-kontrollen efter din ændring
* Ret koden, hvis det ikke virker, som ønsket.

## Øvelse 6.6: Indsæt nyt produkt

Du vil nu gerne kunne indsætte et nyt produkt og automatisk få ListBox-kontrollen i GUI’en opdateret.

Udfør følgende:

* Implementér en ny C#-klasse ”NewProduct”, der implementerer ICommand-interfacet og sørger for at indsætte et nyt produkt i ProductRepo-listen samt ProductsVM-listen, når kommandoens Execute()-metode kaldes. Lad det nye produkt få navnet ”New Product” med default-prisen 0.0.
* Udvid din MainViewModel med en kommando-property (med et passende navn), der instantierer den nye kommando.
* I XAML indsæt en knap under ListBox-kontrollen med teksten ”New Product”.
* I XAML etablér command binding for denne knap til din nye kommando-property i MainViewModel.
* Kør dit program, og afprøv din nye kommando.
* Kører det som det skal? Kommer der er nyt produkt ind i ListBox-kontrollen, når du klikker på knappen?
* Overvej, hvorfor det fungerer, som det gør.

Din ListBox indeholder en databinding (via ItemsSource) til en ProductsVM-liste (Collection) i MainViewModel. Hvis der sker en ændring i Product-listen, f.eks. der indsættes et nyt produkt, eller slettes et eksisterende produkt, via C#-kode (f.eks. via NewProduct.Execute()-metoden), så skal ListBox-kontrollen notificeres om denne ændring.

* Overvej hvorledes ListBox-kontrollen bliver notificeret, når den liste, den er databindet til, ændres
* Overvej hvad ObservableCollection sørger for, som List ikke gør i forbindelse med en ændring af listen

## Øvelse 6.7: Slet valgt produkt

Du vil også kunne slette produkter fra din ProductRepo og ProductsVM-liste i MainViewModel og automatisk få opdateret din ListBox.

Udfør følgende:

* Opret og implementér en ny metode til at slette et produkt i ProductViewModel-klassen med signaturen:

**void DeleteProduct(…)**

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 product-objektet kan slettes fra productRepo’et.

Implementér dernæst den samlede funktionalitet via en ”Delete Product”-knap i GUI’en med inspiration fra forrige øvelse. Vær opmærksom på knappens funktionalitet, hvis der ikke er valgt et produkt fra ListBox-kontrollen, eller listen er tom.