# CommandBinding: oefeningen

Contents

[CommandBinding: oefeningen 1](#_Toc433202126)

[LandInformatie 1](#_Toc433202127)

[TFS integratie 1](#_Toc433202128)

[MVVM command binding 1 1](#_Toc433202129)

[MVVM command binding 2 2](#_Toc433202130)

[MVVM listener extra 2](#_Toc433202131)

[Opmerking: INestedPropertyChangedListener 2](#_Toc433202132)

[Opmerking: INestedPropertyChangedListener en events (niet te kennen voor het examen) 4](#_Toc433202133)

[Resumerend, om het overzicht te bewaren: 5](#_Toc433202134)

[StudentenApp: filter studenten 5](#_Toc433202135)

## LandInformatie

### TFS integratie

* Copieer uw LandInformatie oplossing (vorige week) naar een oplossing voor deze week (dan blijft uw oplossing van vorige week werken ..)
* Maak in TFS (de browser) een project LandInformatie aan;
* Koppel dit project met het zopas aangemaakt TFS project en checkin
  + Indien u niet meer weet hoe dit moet: kijk het na in de cursus OOP van vorig jaar

### MVVM command binding 1

* Installeer het MVVMLight NuGet package
* Voorzie op de master detail Page een delete knop om het geselecteerde landinfo object te verwijderen uit de Landen lijst
* Voorzie op het bijhorende ViewModel een property Delete
  + Return type RelayCommand
  + De command parameter in de ctor van RelayCommand voert de methode DeleteSelectedLandInfo uit (zie cursus word document)
  + De methode DeleteSelectedLandInfo werkt als volgt:
    - roep de static DeleteLandInfo methode in LandRepository op (die moet u zelf nog schrijven). Als parameter geven we het SelectedItem mee;
    - zet het SelectedLandInfo op null (het verwijderde item kan/mag niet meer geselecteerd zijn)
    - roep OnPropertyChanged op voor de nodige property namen
* Gebruik Blend om het Click event van de Delete button te koppelen aan de property Delete
* Test uw oplossing.
  + Indien de Landen verzameling een List<LandInfo> is zal het resultaat na het deleten niet correct zijn: in het beste geval verdwijnt het laatste element van de lijst (en niet het geselecteerde element).
  + Vervang zoals in de theorie les gedemonstreerd List<LandInfo> door ObservableCollection<LandInfo>. U zal dit op meerdere plaatsen moeten realiseren, de compiler helpt u met compileerfouten de aan te passen code terugvinden.

### MVVM command binding 2

We voorzien de mogelijkheid om in een gefilterde LandInfo lijst een element te verwijderen:

* Voorzie in de template een delete knop
* Voorzie een LandInfoVM viewmodel klasse als wrapper rond de LandInfo
* Definieer een Listener interface (InestedPropertyChangedListener in het theorie document) voor LandInfo en implementeer deze interface met het viewmodel waarin de LandInfoVM objecten worden gemaakt. Bij de creatie van dergelijk LandInfoVM object stel je het creeërende viewmodel in als listener van het LandInfoVM object (in de cursus tekst gebruiken we hiervoor LINQ)
* Voorzie in LandInfoVM een Delete property van type RelayCommand
* Gebruik Blend om de Delete knop te koppelen aan de Delete property
  + Omdat de DataTemplate geen ingestelde DataContext heeft (anders dan de Page van de vorige oefening) zal u de property niet in een lijst terugvinden maar Delete als custom path moeten voorzien
* Na het verwijderen van het LandInfo object verwittigen we de listener, die hierop gepast reageert (in dit geval: de gefilterde lijst herberekent).

### MVVM listener via interface INestedPropertyChangedListener

Indien de IsEuropees property van een LandInfoVM object wijzigt moeten we ook hier de Listener verwittigen zodat de gefilterde lijst kan aangepast worden

## Resumerend, om het overzicht te bewaren:

Page ViewModels met gewone properties en RelayCommand properties

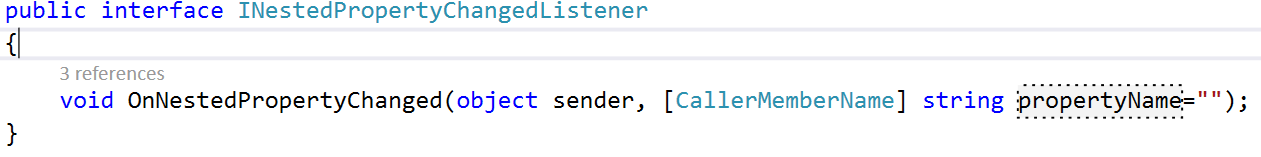
* TwoWay binding: vanuit de UI naar het ViewModel
* INotifyPropertyChanged vanuit het ViewModel naar de UI
* ObservableCollection voor lijsten die kunnen gewijzigd worden in het ViewModel
* RelayCommand properties om commandbinding mogelijk te maken

Model ViewModels met gewone properties en RelayCommand properties

* TwoWay binding: vanuit de UI naar het ViewModel
* INotifyPropertyChanged vanuit het ViewModel naar de UI
* ObservableCollection voor lijsten die kunnen gewijzigd worden in het ViewModel
* RelayCommand properties om commandbinding mogelijk te maken

Om communicatie vanuit model viewmodels naar hun page viewmodel mogelijk te maken:

* Definieer een listener interface



* Voorzie op het model viewmodel een listener property voor deze interface
* Implementeer in het Page ViewModel deze interface
* Wanneer het Page ViewModel de model ViewModel objecten aanmaakt stelt het Page ViewModel zichzelf in als listener
* Bij een property wijziging in het Model ViewModel wordt via de listener het Page ViewModel hiervan op de hoogte gebracht. Het Page ViewModel beslist wat hiermee te doen

## StudentenApp: filter studenten

Voorzie de mogelijkheid om de studenten te filteren op basis van een tekst:



* De zoektekst is een gewone property (TwoWay bound) in het ViewModel
* De zoek knop is een button met hierin een SymbolIcon. In Windows 10 wordt een ganse lijst standaard icoontjes (schaalbaar) aangeboden via de [Symbol enumeration](https://msdn.microsoft.com/en-us/library/windows.ui.xaml.controls.symbol.aspx).

U kan de content van een Button instellen op dergelijk icon op onderstaande manier (de behavior is hiervoor niet van belang, die wordt gebruikt om de command binding te realiseren (best via Blend))

