Haute École Léonard de Vinci
SCIENCES ET TECHNIQUES

BINV314A
.NET Outils et Concepts d'Application d'Entreprise

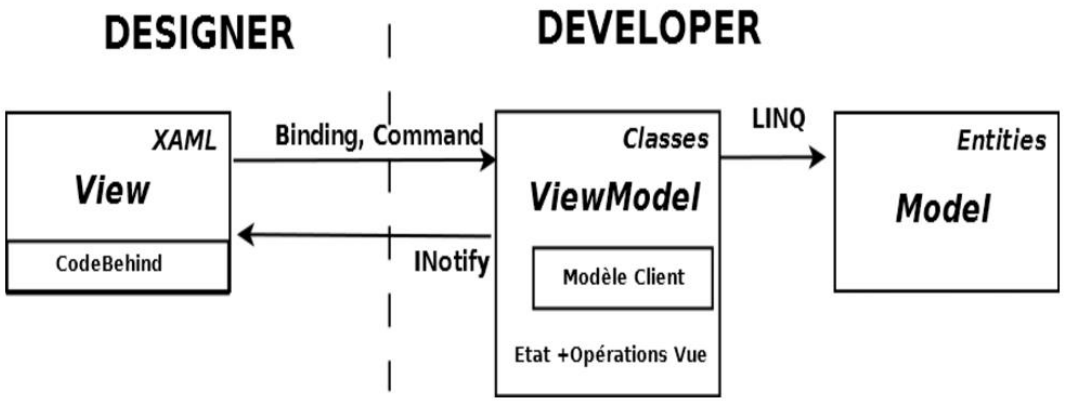
Semaine 10

WPF - MVVM - LINQ

Sommaire : WPF - MVVM - LINQ

- Notifications
- Command
- ObservableCollection

MVVM : Séparation en couches



MVVM : Notifications

- Interface INotifyPropertyChanged
- Event PropertyChangedEventHandler
- Création d'une méthode générique OnPropertyChanged

MVVM : Notifications

```
class LegumeModel : INotifyPropertyChanged {

   // Property changed standard handling

        public event PropertyChangedEventHandler PropertyChanged;

        // La view s'enregistera automatiquement sur cet event

        protected virtual void OnPropertyChanged(string propertyName)

        {

            if (PropertyChanged != null)

            {

                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));

            }

        }

…

set { _legume.name = value; OnPropertyChanged("Name"); }
```

MVVM : Command

- Gestion des événements

```
<Button Click="Button_Click"
```

  - EventHandler
  - Problème -> le nom de la méthode de traitement 'Button_Click' est dans le XAML et pire le code de traitement est dans le fichier xaml.cs

MVVM : Command

- Inconvénients des EventHandler traditionnels (Event Click par exemple)
  - Liaison forte entre le code qui utilise (s'abonne) et le code qui propose l'événement
  - Le code de traitement est dans le fichier xaml.cs

D'où l'idée d'utiliser des **Command**

MVVM : Command

- <Button Command="...."
- Interface ICommand
  - Execute(Object)
  - CanExecute (Object)
  - Event handler CanExecuteChanged

MVVM : Command

- Interface ICommand
  - Ok mais alors pour chaque command (bouton), je dois créer une classe implémentant ICommand ?
  - En théorie oui, seulement c'est ingérable en pratique
  - En pratique, nous créons une seule classe (Delegate) implémentant ICommand à laquelle nous passerons la méthode à exécuter

MVVM : Command

```
public class DelegateCommand : ICommand
  {
     private Action _executeMethod;
     public event EventHandler CanExecuteChanged;
     public DelegateCommand(Action executeMethod) {
          _executeMethod = executeMethod;
     }
     public bool CanExecute(object parameter)  { return true;}
     public void Execute(object parameter)  {
        _executeMethod.Invoke(); }
  }
```

MVVM : Command

- Ok mais comment lier tout ceci ?
  - <Button Command="…."
  - Binding → DelegateCommand → Trt

MVVM : Command

- Exemple
  - XAML : <Button Command="{Binding delCommand} »
  - ViewModel : propriété delCommand qui instancie une delegateCommand à laquelle on donne la méthode à exécuter (del)
  - ViewModel : définition d'une méthode del effectuant le traitement

ObservableCollection

- Problème : maj dans le ViewModel d'une liste utilisée dans la vue → notifier la vue
  - Solution 1 : Notifications (InotifyPropertyChanged et/ou InotifyCollectionChanged)
  - Solution 2 : ObservableCollection (Implémente déjà InotifyCollectionChanged)

Tests WPF

- Juste pour info … je ne demanderai pas cela à l'examen
- Tests unitaires
  - Facile à mettre en œuvre,il suffit de tester le(s) viewmodels principal(aux)
- Tests fonctionnels
  - https://docs.microsoft.com/fr-fr/visualstudio/test/use-ui-automation-to-test-your-code?view=vs-2019
  - Visual Studio **Entreprise** requis