



{POO}

## Socle commun sur la programmation orientée objet

### ➤ Objectifs de la formation :

- Donner (ou rappeler) les notions de base de la programmation orientée objet
- Faciliter l'apprentissage de nouveaux langages ou nouvelles technologies
- Partager les mêmes conventions et les mêmes façons de programmer
- Faciliter le partage de code et l'entraide au sein de la communauté

### ➤ Méthode :

- A partir d'un exemple en langage C# et Framework Xamarin
- Pour :
  - Concrétiser les notions
  - Avoir une première approche de Xamarin

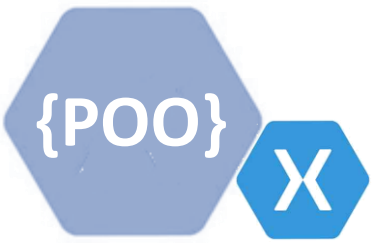


INRAE



Cati Sicpa

Socle commun sur la programmation orientée objet - Alexandre Journaux  
Ce document est mise à disposition selon les termes de la  
[Licence Creative Commons Attribution 4.0 International](https://creativecommons.org/licenses/by/4.0/)



# Programmation orientée objet avec un projet en Xamarin

En partant d'un exemple de programme à réaliser

Liste d'élevages ▼

Numéro animal OK

Poids animal

Enregistrer

Afficher historique

## Objectif du programme :

Enregistrer la pesée d'un animal

## Étapes :

- 1/ Sélectionner un élevage
- 2/ Saisir un numéro animal
- 3/ Valider la saisie (= vérifier qu'il existe dans l'élevage sélectionné)
- 4/ Saisir le poids
- 5/ Enregistrer (simuler l'enregistrement en affichant les infos saisies)
- 6/ Afficher l'historique d'un animal (ses pesées et autres mesures)



# Programmation orientée objet avec un projet en Xamarin


## Rappel des concepts de la programmation orientée objet

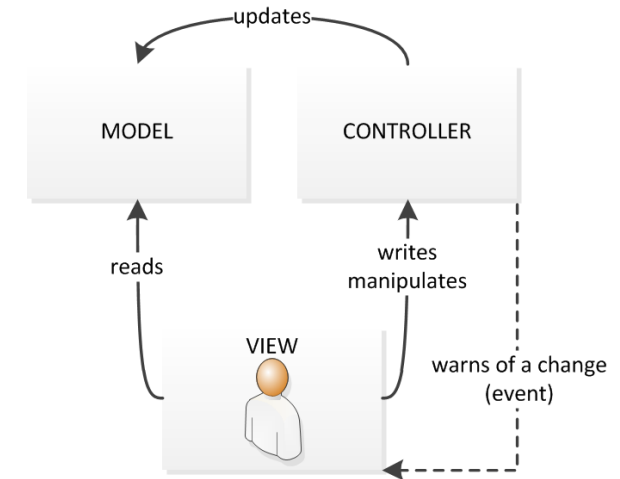
- La **Classe** : modèle de définition pour des objets ayant une même structure (**attributs**) et même comportement (**méthodes**)
- **L'objet** : représentation dynamique d'une classe (**instanciation**)  
Exemple : `Animal a = new Animal();` // a est un objet qui instancie la classe Animal
- **L'encapsulation** : regroupement des données dans une classe en empêchant l'accès aux données par un autre moyen que les services proposées
- **L'abstraction** : masquer le détail et n'exposer que le nécessaire à l'utilisateur de la classe
- **L'héritage** : Permet de créer une classe à partir d'une autre en récupérant la même structure et les mêmes comportements  
Exemple : `public class Bovin : Animal` // La classe Bovin hérite de la classe Animal
- Le **polymorphisme** : Possibilité pour une méthode de prendre plusieurs formes ou avoir un comportement spécifique  
Exemple :  
`Animal a = new Animal(); a.PeseAnimal(); a.PeseAnimal("2021/08/13");` // Soit date par défaut, soit date spécifiée  
`Bovin b = new Bovin(); b.PeseAnimal();` // Soit comportement hérité, soit comportement spécifique si surcharge de la méthode

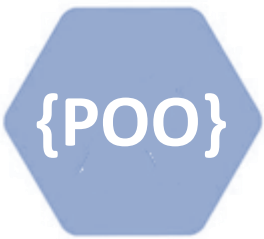


# Programmation orientée objet avec un projet en Xamarin

## Modèle de programmation par couche

- Chaque couche a des responsabilités différentes
- Exemple : MVC, MVVM 
- Les classes d'une couche ne doivent traiter que ce qui correspond à leurs rôles
- Par exemple
  - Dans une classe modèle, on ne doit pas trouver d'accès à la vue
  - Dans une vue, on ne doit pas trouver d'accès à la base de données





# Programmation orientée objet avec un projet en Xamarin

La programmation par couche avec Xamarin

Liste d'élevages ▼

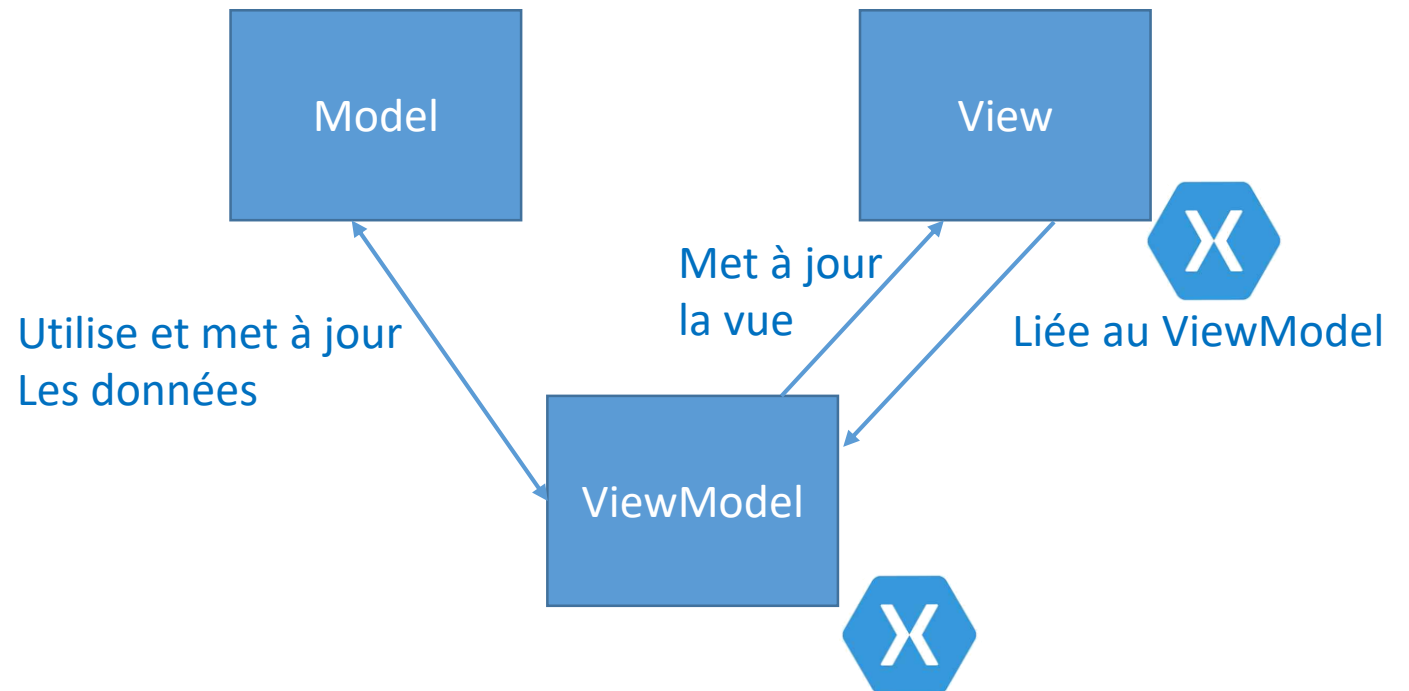
Numéro animal  OK

Poids animal

Enregistrer

Afficher historique

Programmer avec Xamarin c'est utiliser le pattern MVVM





# Programmation orientée objet avec un projet en Xamarin

La programmation par couche avec Xamarin

Liste d'élevages ▼

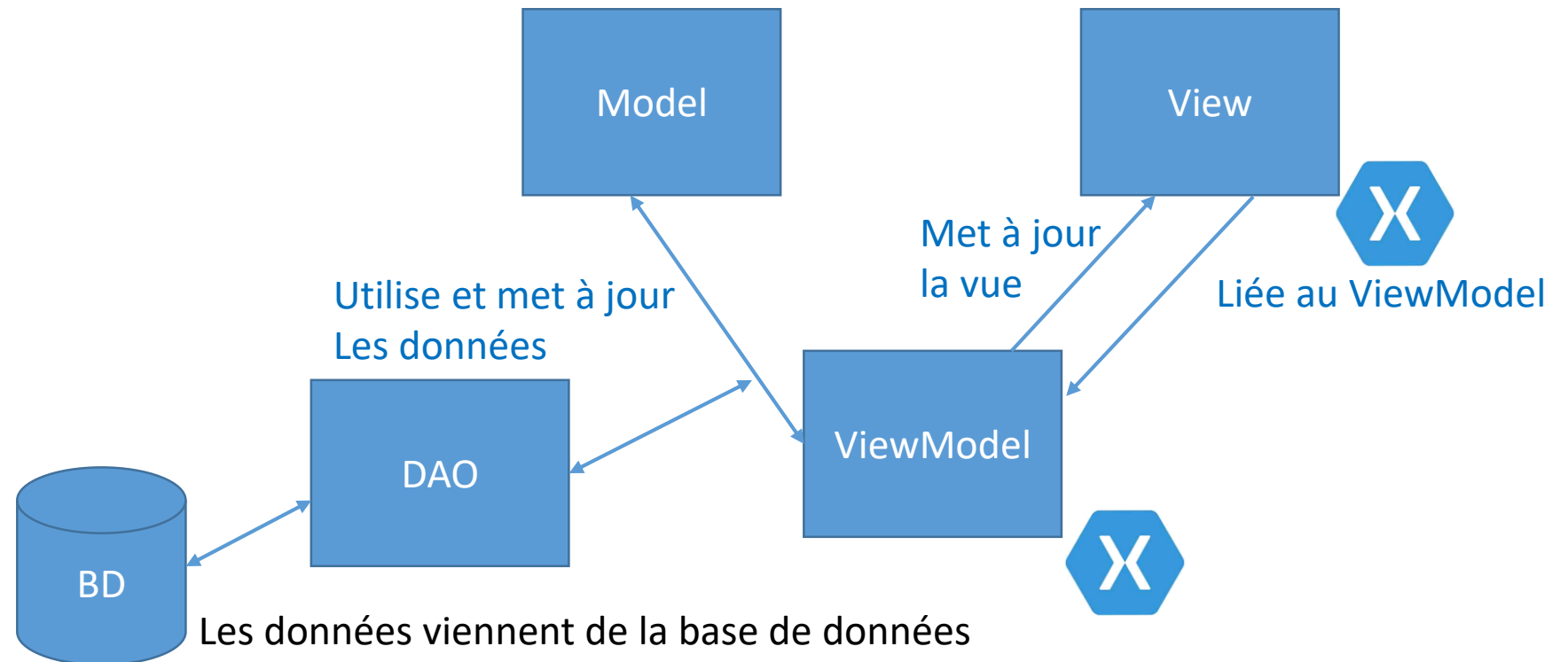
Numéro animal  OK

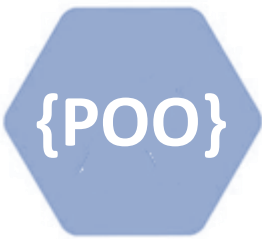
Poids animal

Enregistrer

Afficher historique

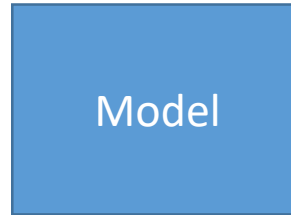
Pour l'accès aux données, ajout du pattern DAO





# Programmation orientée objet avec un projet en Xamarin

Modèle de données

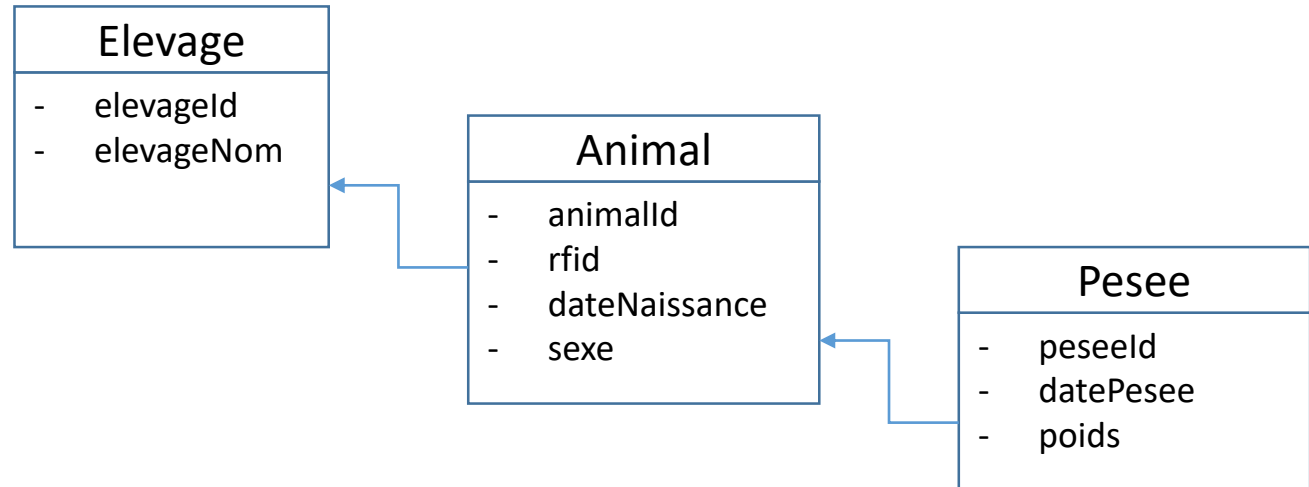


Liste d'élevages ▼

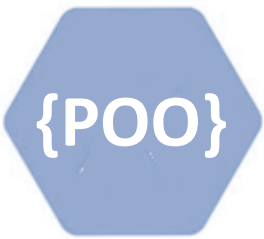
Numéro animal

Poids animal

Diagramme de classes :



Donc 3 classes à écrire dans le dossier Model



# Programmation orientée objet avec un projet en Xamarin

## Création de la classe Elevage

### ➤ Petits rappels

- Une classe est constituée de données qu'on appelle des attributs
- De procédures et/ou de fonctions qu'on appelle des méthodes
- Toute classe hérite de la classe **object**
  - Et donc des 3 méthodes *ToString()*, *Equals()* et *GetHashCode()*

### ➤ Quelques conventions à respecter

- Un attribut doit être privé (Principe « objet » de l'encapsulation)
  - Il va s'écrire en minuscule (éventuellement avec \_ devant)
- L'accès aux attributs va se faire par des méthodes publiques qu'on appelle des accesseurs
  - Le principe c'est que chaque classe contrôle l'accès et la mise à jour de ses attributs
  - En java, c'est les méthodes getters et setters
  - En C#, c'est les propriétés
    - Elles vont s'écrire avec une Majuscule en première lettre

### ➤ Convention de nommage (conseil)

- Utilisation du CamelCase

NomDeMaClasse, nomDeMonAttribut, NomDeMaPropriete, NomDeMaMethode, nomDeMaVariable



Raccourci VS : taper uniquement les majuscules d'un nom et VS complètera





# Programmation orientée objet avec un projet en Xamarin

## Création de la classe Elevage

➤ Quelles sont les bonnes façons d'écrire la classe Elevage ?

```
public class Elevage
{
    public int elevageId;
    public String elevageNom;
}
```



```
public class Elevage
{
    private int elevageId;
    private String elevageNom;
}
```



```
public class Elevage
{
    private int elevageId;
    private String elevageNom;

    public int getElevageId()
    {
        return this.elevageId;
    }
    public void setElevageId(int elevageId)
    {
        this.elevageId = elevageId;
    }

    public String getElevageNom()
    {
        return this.elevageNom;
    }
    public void setElevageNom(String elevageNom)
    {
        this.elevageNom = elevageNom;
    }
}
```





# Programmation orientée objet avec un projet en Xamarin

## Création de la classe Elevage

➤ Quelles sont les bonnes façons d'écrire la classe Elevage ?

```
public class Elevage
{
    private int elevageId;
    private String elevageNom;
    public int ElevageId
    {
        get
        {
            return this.elevageId;
        }
        set
        {
            this.elevageId = value;
        }
    }
    public String ElevageNom
    {
        get
        {
            return this.elevageNom;
        }
        set
        {
            this.elevageNom = value;
        }
    }
}
```

```
public class Elevage
{
    private int elevageId;
    private String elevageNom;

    public int ElevageId { get => elevageId; set => elevageId = value; }
    public string ElevageNom { get => elevageNom; set => elevageNom = value; }
}
```

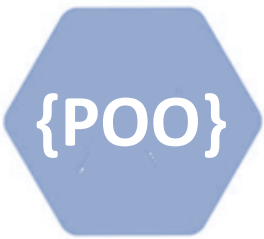
```
public class Elevage
{
    public int ElevageId { get; set; }
    public String ElevageNom { get; set; }
}
```



En Xamarin, on verra que l'écriture complète peut être nécessaire



Raccourci VS : propfull - Tab - Tab

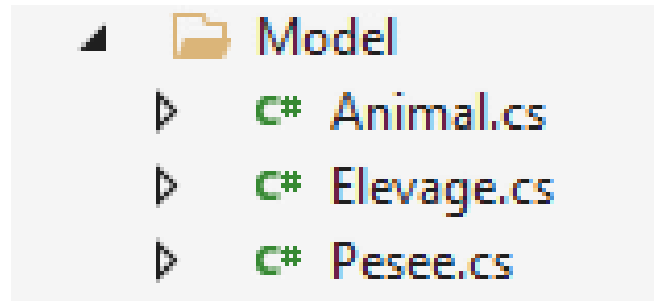
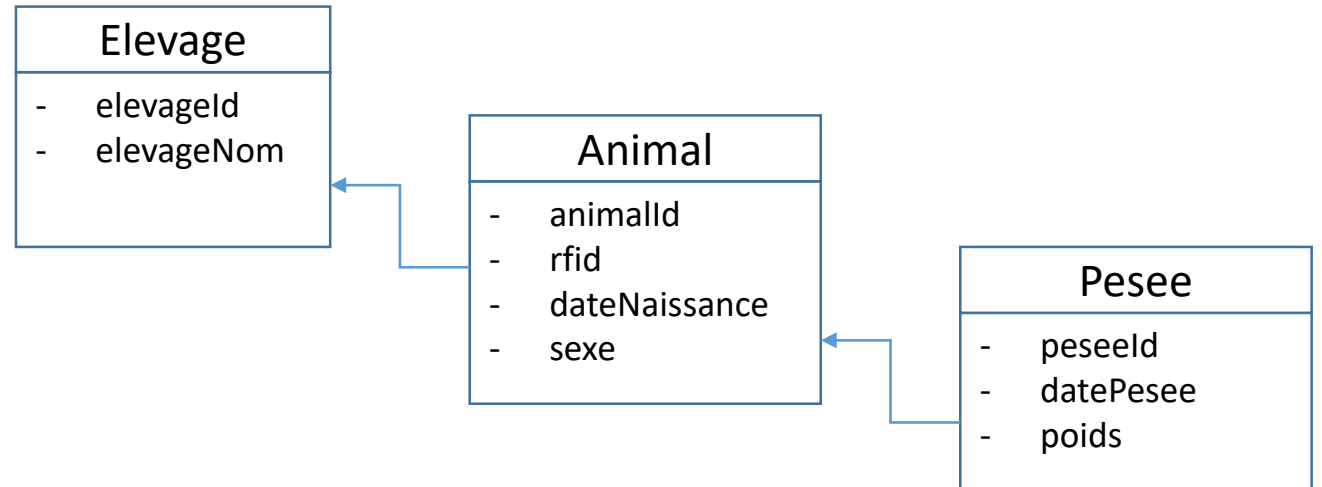


# Programmation orientée objet avec un projet en Xamarin

TP : Création des 3 classes du modèle



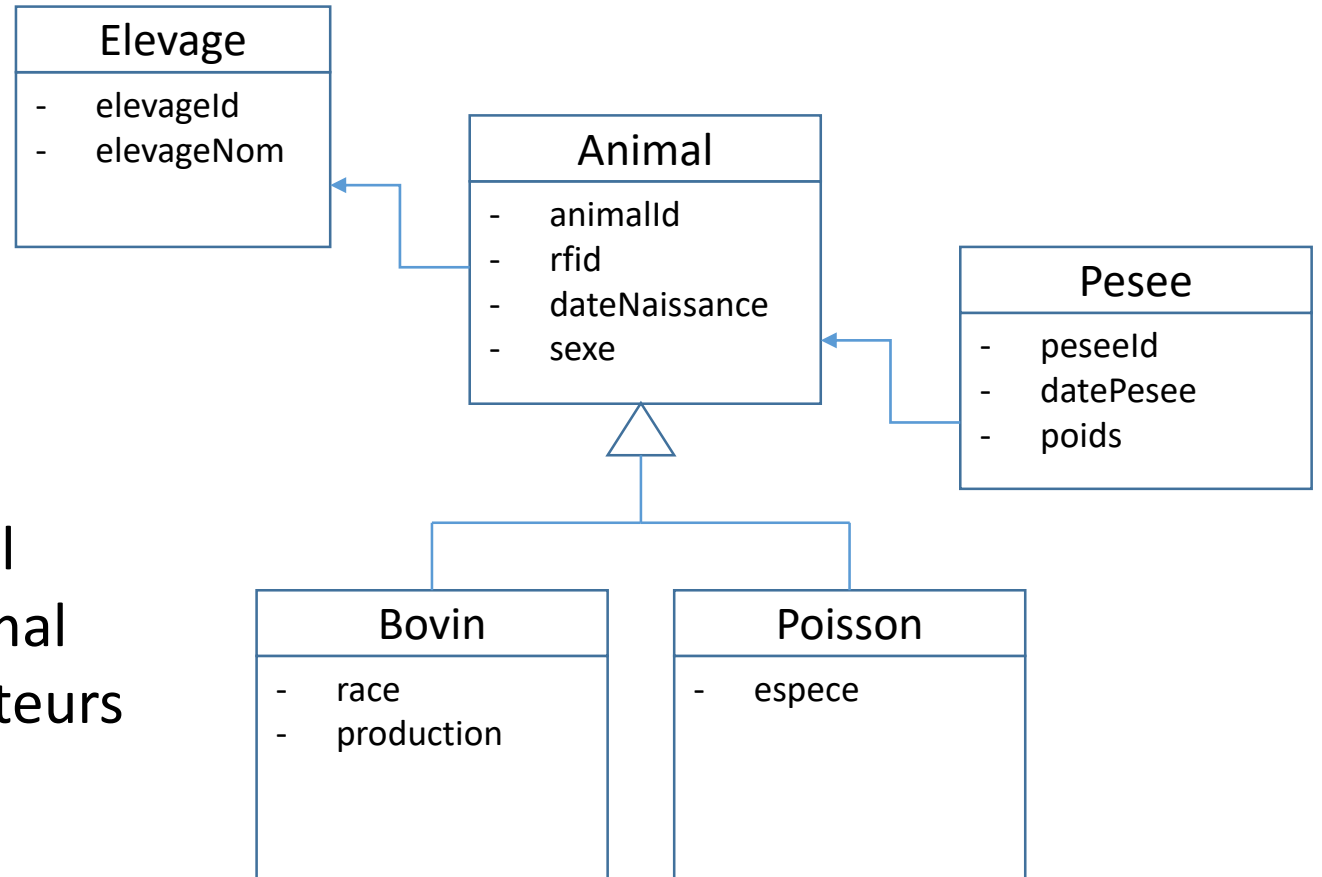
- Créer le dossier Model
- Créer la classe Elevage
- Créer la classe Animal
- Créer la classe Pesee





# Programmation orientée objet avec un projet en Xamarin

TP : Ajout de 2 classes filles de Animal



- Créer la classe Bovin qui hérite de Animal
- Créer la classe Poisson qui hérite de Animal
- Utiliser le mot clé `base` pour les constructeurs



# Programmation orientée objet avec un projet en Xamarin

## Création d'une interface

- Qu'est-ce qu'une interface ?
  - Sorte de classes abstraites sans aucune méthode implémentée
  - Intérêt
    - Etablie un contrat avec la classe qui l'implémente
    - En C#, une classe ne peut pas hériter de plusieurs classes  
par contre, une classe peut implémenter plusieurs interfaces
    - Une interface peut étendre plusieurs interfaces
  - Particularités
    - Une interface ne possède pas d'attribut
    - Les interfaces ne sont pas instanciables (comme les classes abstraites)
  - Exemple
    - La classe Bovin pourrait implémenter les interfaces IMammifere, IPature, ISerializable, IComparable, ...



Convention de nommage : INomInterface

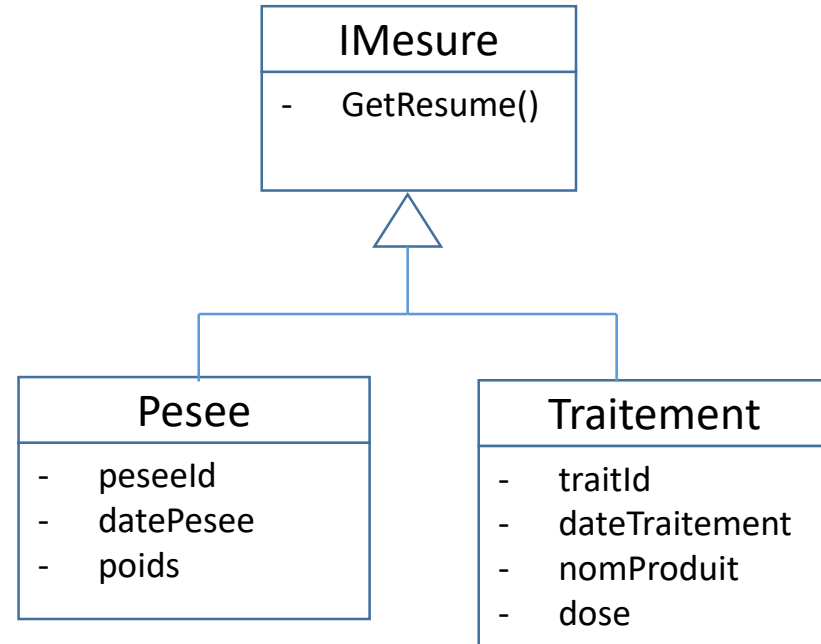


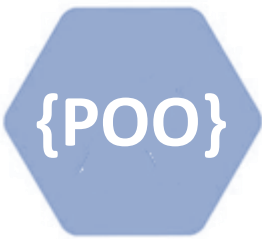
# Programmation orientée objet avec un projet en Xamarin

TP : Création d'une interface



- Créer l'interface IMesure
- Créer la classe Traitement
- Implémenter IMesure pour les classes Pesee et Traitement





# Programmation orientée objet avec un projet en Xamarin

Accès aux données



Liste d'élevages ▼

Numéro animal OK

Poids animal

Enregistrer

Afficher historique

## Fonctions dont on a besoin :

- Lister tous les élevages de la base de données
- Lister tous les animaux d'un élevage
- Enregistrer la pesée
- Afficher l'historique pour un animal
- => 4 méthodes à écrire :
  - `List<Elevage> GetElevages()`
  - `List<Animal> GetAnimauxByElevage(Elevage e1)`
  - `List<String> GetHistoriqueByAnimal(Animal an)`
  - `int Save(Pesee pe)`
- Comme ces 3 méthodes concernent 3 objets différents, il est recommandé de **créer 3 classes dans le dossier DAO** :
  - `ElevageDAO`, `AnimalDAO` et `PeseeDAO`



# Programmation orientée objet avec un projet en Xamarin

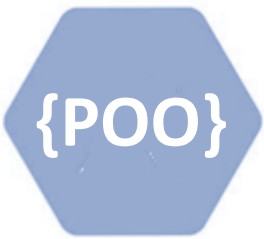
TP : Création des 3 classes DAO et leurs méthodes

- Créer le dossier DAO
- Créer la classe `FictifElevageDAO`  
et sa méthode `List<Elevage> GetElevages()`  
(utilisation du constructeur de la classe Elevage)
- Créer la classe `FictifAnimalDAO`  
et sa méthode `List<Animal> GetAnimauxByElevage(Elevage e1)`  
(utilisation du constructeur des classes Bovin et Poisson et aussi ???)
- Créer la classe `FictifPeseeDAO`  
et sa méthode `int Save(Pesee pe)`  
(retourne 1 si l'animal n'est pas null, -1 sinon)



Pour ce TP, on ne va créer que des méthodes fictives d'accès aux données.  
Dans la vraie vie, ces classes récupèrent les données en se connectant à la BD ou via des WS





# Programmation orientée objet avec un projet en Xamarin

TP : Création des 3 classes DAO et leurs méthodes

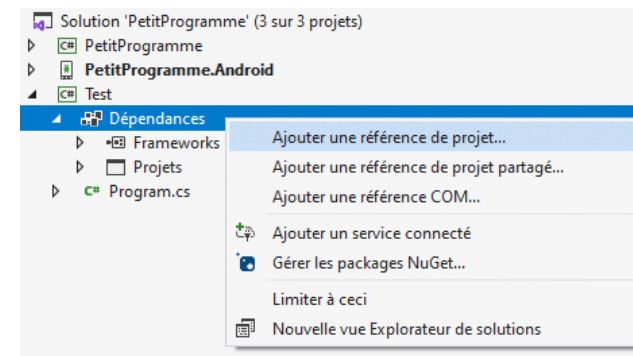
- Dans la classe `FictifAnimalDAO`, créer la méthode `List<String> GetHistoriqueByAnimal(Animal an)`

Pour cette méthode, on va considérer que la BD nous retourne une liste de mesures : `List<IMesure>` avec un mélange de Pesées et de Traitements



Pour tester :

1. Ajouter un nouveau projet à la solution de type « Application Console »
2. Ajouter à ce projet une référence au projet principal
3. Ecrire le code test dans la méthode `Main()`
4. Définir ce projet comme « projet de démarrage »





# Programmation orientée objet avec un projet en Xamarin

Construction de l'IHM



Liste d'élevages	▼
Numéro animal	OK
Résultat validation numéro	
Poids animal	
Enregistrer	
Afficher historique	
Affichage de l'historique	

## Composition de l'écran :

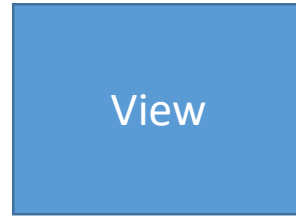
- Une grille (2 colonnes, 6 lignes) : `<Grid>`
- Une liste déroulante : `<Picker>`
- Deux champs de saisie : `<Entry>`
- Trois boutons : `<Button>`
- Deux zones de texte d'affichage : `<Label>`

Donc 1 vue à écrire dans le dossier View : `PeseeAnimalView`



# Programmation orientée objet avec un projet en Xamarin

Construction de l'IHM



Liste d'élevages	▼
Numéro animal	OK
Résultat validation numéro	
Poids animal	
Enregistrer	
Afficher historique	
Affichage de l'historique	

**La grille :**

```
<Grid ColumnDefinitions="75*,25*"
RowDefinitions="auto,auto,auto,auto,auto,auto,auto">
```

**La liste déroulante :**

```
<Picker Grid.Row="0" Grid.ColumnSpan="2" Title="Liste des
élevages"/>
```

**Les 2 champs de saisie :**

```
<Entry Grid.Row="1" Grid.Column="0" Placeholder="Numéro
animal" Keyboard="Numeric"/>
```

```
<Entry Grid.Row="3" Grid.Column="0" Placeholder="Poids"
Keyboard="Numeric"/>
```



# Programmation orientée objet avec un projet en Xamarin

Construction de l'IHM



Liste d'élevages	▼
Numéro animal	OK
Résultat validation numéro	
Poids animal	
Enregistrer	
Afficher historique	
Affichage de l'historique	

## Les 3 boutons :

```
<Button Grid.Row="1" Grid.Column="1" Text="OK"/>
```

```
<Button Grid.Row="4" Grid.ColumnSpan="2" Text="Enregistrer"/>
```

```
<Button Grid.Row="5" Grid.ColumnSpan="2" Text="Afficher" />
```

## Les 2 zones de texte :

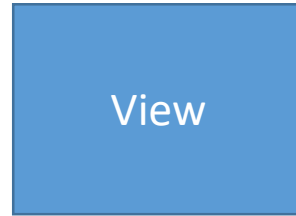
```
<Label Grid.Row="2" Grid.ColumnSpan="2" />
```

```
<Label Grid.Row="6" Grid.ColumnSpan="2" />
```



# Programmation orientée objet avec un projet en Xamarin

Construction de l'IHM



**Pour que la vue soit appelée au démarrage de l'application :**

Dans le constructeur de la classe App, ajouter l'appel de la vue :

```
public App()  
{  
    InitializeComponent();  
  
    MainPage = new PeseeAnimalView();  
}
```



# Programmation orientée objet avec un projet en Xamarin

Interagir avec la vue

ViewModel

**Afin de mettre en place le binding dans le projet :**

Créer la classe `BaseViewModel` dans le dossier ViewModel

```
public class BaseViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Cette classe sera héritée par toutes les autres classes ViewModel afin de bénéficier de la méthode `OnPropertyChanged`



# Programmation orientée objet avec un projet en Xamarin

Interagir avec la vue

ViewModel

Afin d'interagir avec la vue `PeseeAnimalView` :

- Créer la classe `PeseeAnimalViewModel` dans le dossier ViewModel

```
public class PeseeAnimalViewModel : BaseViewModel
```

**Règle : 1 ViewModel pour 1 View**

- Instancier cette classe dans le constructeur de la vue :

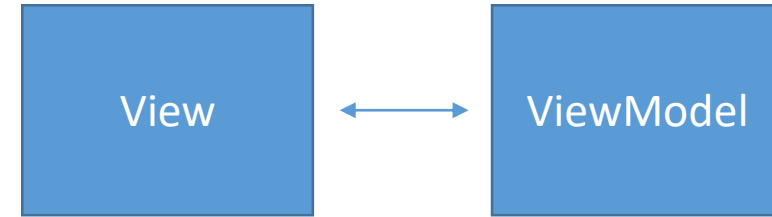
(dans le code Behind)

```
public PeseeAnimalView()  
{  
    InitializeComponent();  
    this.BindingContext = new PeseeAnimalViewModel();  
}
```



# Programmation orientée objet avec un projet en Xamarin

Alimentation de la liste d'élevages



Liste d'élevages	▼
Numéro animal	OK
Résultat validation numéro	
Poids animal	
Enregistrer	
Afficher historique	
Affichage de l'historique	

Dans la View, sur le Picker ajouter la propriété `ItemsSource`  
`<Picker Grid.Row="0" Grid.ColumnSpan="2" Title="Liste des élevages" ItemsSource="{Binding Elevages}"/>`

Dans le ViewModel, création d'un attribut `elevages` et sa propriété `Elevages`

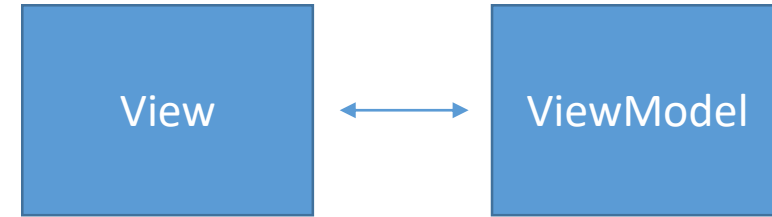
```
public class PeseeAnimalViewModel : BaseViewModel
{
    private List<Elevage> elevages;
}
```





# Programmation orientée objet avec un projet en Xamarin

Alimentation de la liste d'élevages



Dans le ViewModel, création d'un attribut elevages et sa propriété Elevages

```
public List<Elevage> Elevages
{
    get { return elevages; }
    set
    {
        if (elevages != value)
        {
            elevages = value;
            OnPropertyChanged();
        }
    }
}
```

Liste d'élevages

Numéro animal OK

Résultat validation numéro

Poids animal

Enregistrer

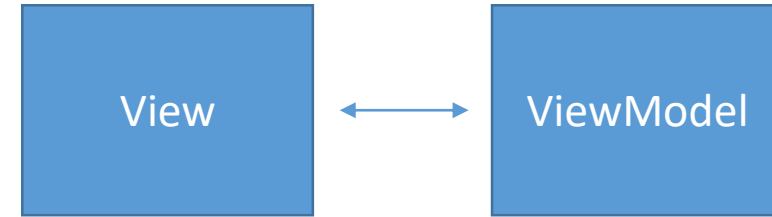
Afficher historique

Affichage de l'historique



# Programmation orientée objet avec un projet en Xamarin

Alimentation de la liste d'élevages



**Dans le constructeur du ViewModel, alimentation de la liste Elevages**

```
FictifElevageDAO e1DAO = new FictifElevageDAO();  
Elevages = e1DAO.GetElevages();
```



A cette étape, normalement, la liste des élevages est bien alimentée.

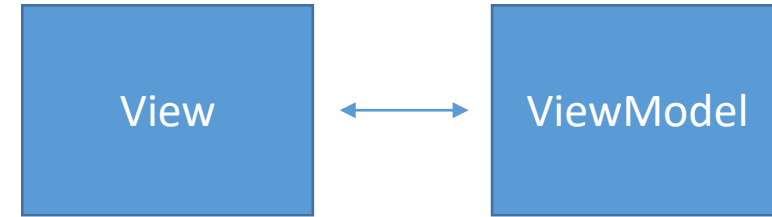
Par contre, que constatez-vous ?

Liste d'élevages	▼
Numéro animal	OK
Résultat validation numéro	
Poids animal	
Enregistrer	
Afficher historique	
Affichage de l'historique	



# Programmation orientée objet avec un projet en Xamarin

Alimentation de la liste d'élevages



Liste d'élevages	▼
Numéro animal	OK
Résultat validation numéro	
Poids animal	
Enregistrer	
Afficher historique	
Affichage de l'historique	

**Afficher le nom de l'élevage dans la liste déroulante.**

**2 solutions**

- **Surcharger la méthode ToString() de la classe Elevage**

```
public override string ToString()  
{  
    return this.ElevageNom;  
}
```

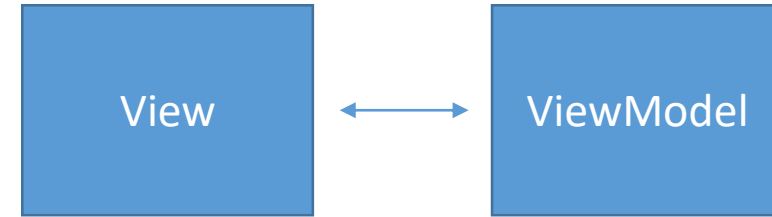
- **Préciser le display du Picker**

```
<Picker Grid.Row="0" Grid.ColumnSpan="2" Title="Liste des  
élevages" ItemsSource="{Binding Elevages}"  
ItemDisplayBinding="{Binding ElevageNom}"/>
```



# Programmation orientée objet avec un projet en Xamarin

Récupération de l'élevage sélectionné



Pour la suite du programme, il est nécessaire de récupérer l'élevage sélectionné par l'utilisateur.

- Renseigner le `SelectedItem` du `Picker Elevage`

```
<Picker Grid.Row="0" Grid.ColumnSpan="2" Title="Liste des élevages"
ItemsSource="{Binding Elevages}" ItemDisplayBinding="{Binding ElevageNom}"
SelectedItem="{Binding ElevageSelectionne}"/>
```

- Dans le `ViewModel`, déclarer cette variable

```
private Elevage elevageSelectionne;
```

```
public Elevage ElevageSelectionne
```

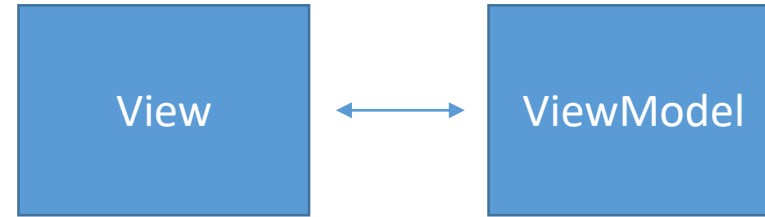
```
{
    get { return elevageSelectionne; }
    set
    {
        ...
    }
}
```

```
...
if (elevageSelectionne != value)
{
    elevageSelectionne = value;
    OnPropertyChanged();
} ...
```



# Programmation orientée objet avec un projet en Xamarin

TP : Lier les composants visuels de la View à des variables du ViewModel



- Lier les propriétés **Text** des 2 champs de saisie
- Lier les propriétés **Text** des 2 labels

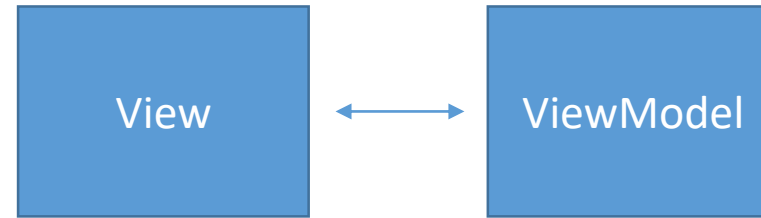


Liste d'élevages	▼
Numéro animal	OK
Résultat validation numéro	
Poids animal	
Enregistrer	
Afficher historique	
Affichage de l'historique	



# Programmation orientée objet avec un projet en Xamarin

TP : Solution



## Les 2 champs de saisie :

```
<Entry Grid.Row="1" Grid.Column="0" Placeholder="Numéro animal" Text="{Binding AnimalNumero}" Keyboard="Numeric"/>
```

- Dans le VM : **AnimalNumero** est un **string** qui sera alimenté par la saisie de l'utilisateur

```
<Entry Grid.Row="3" Grid.Column="0" Placeholder="Poids" Text="{Binding AnimalPoids}" Keyboard="Numeric"/>
```

- Dans le VM : **AnimalPoids** est un **int** qui sera alimenté par la saisie de l'utilisateur

## Les 2 labels :

```
<Label Grid.Row="2" Grid.ColumnSpan="2" Text="{Binding RetourValidation}"/>
```

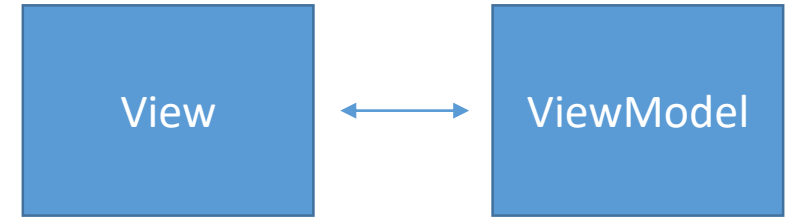
```
<Label Grid.Row="6" Grid.ColumnSpan="2" Text="{Binding AffichageHistorique}"/>
```

- Dans le VM : **RetourValidation** est un **string** qui sera alimenté par la validation du bouton « OK »  
**AffichageHistorique** est un **string** qui sera alimenté par la validation du bouton « Afficher... »



# Programmation orientée objet avec un projet en Xamarin

Lier les boutons de la View au ViewModel



Liste d'élevages	▼
Numéro animal	OK
Résultat validation numéro	
Poids animal	
Enregistrer	
Afficher historique	
Affichage de l'historique	

## Les 3 boutons :

```
<Button Grid.Row="1" Grid.Column="1" Text="OK"  
Command="{Binding ValiderNumeroAnimalCommand}"/>
```

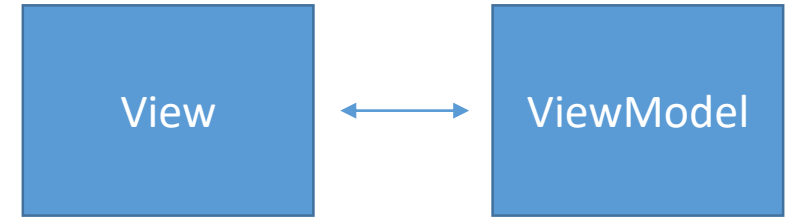
```
<Button Grid.Row="4" Grid.ColumnSpan="2" Text="Enregistrer"  
Command="{Binding EnregistrerCommand}"/>
```

```
<Button Grid.Row="5" Grid.ColumnSpan="2" Text="Afficher"  
Command="{Binding AfficherCommand}"/>
```



# Programmation orientée objet avec un projet en Xamarin

Lier les boutons de la View au ViewModel



## Faire fonctionner le bouton « OK » :

- `Command="{Binding ValiderNumeroAnimalCommand}"`

`ValiderNumeroAnimalCommand` est un objet de type `ICommand` qui permet de lier le bouton à une action et à une condition d'exécution

- Dans le ViewModel

- Déclaration :

```
public ICommand ValiderNumeroAnimalCommand { get; private set; }
```

- Instanciation dans le constructeur du ViewModel :

```
ValiderNumeroAnimalCommand = new Command(ValiderNumeroAnimal, CanValiderNumeroAnimal);
```

- 2 méthodes à écrire :

```
private void ValiderNumeroAnimal()
```

Permet de vérifier (et afficher) si le numéro saisi correspond bien à un animal de l'élevage sélectionné

```
private bool CanValiderNumeroAnimal()
```

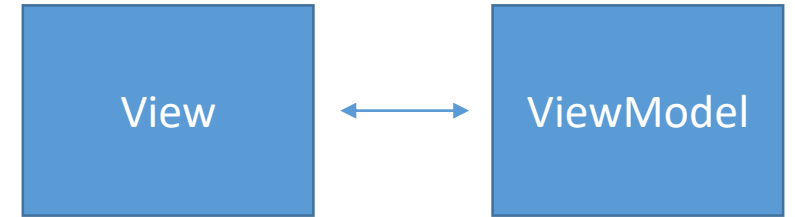
Retourne vrai si le numéro animal est saisi





# Programmation orientée objet avec un projet en Xamarin

Lier les boutons de la View au ViewModel



## Faire fonctionner le bouton « Enregistrer » :

- `Command="{Binding EnregistrerCommand}"`

`EnregistrerCommand` est un objet de type `ICommand` qui permet de lier le bouton à une action et à une condition d'exécution

- Dans le ViewModel

- Déclaration :

```
public ICommand EnregistrerCommand { get; private set; }
```

- Instanciation dans le constructeur du ViewModel :

```
EnregistrerCommand = new Command(EnregistrerPesee, CanValiderNumeroAnimal);
```

- 2 méthodes à écrire :

```
private void EnregistrerPesee()
```

Permet d'enregistrer la pesée par l'appel de la méthode `save()` de `PeseeDAO`

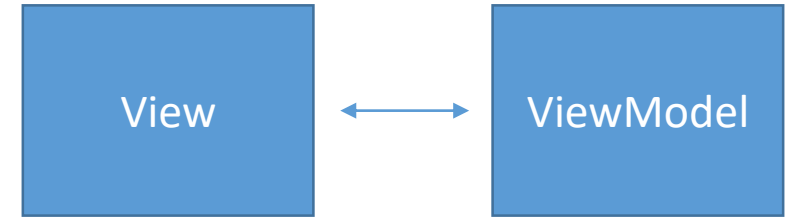
```
private bool CanValiderNumeroAnimal()
```

Retourne vrai si le numéro animal est saisi



## Programmation orientée objet avec un projet en Xamarin

Lier les boutons de la View au ViewModel



### Et, enfin, pour que tout fonctionne bien :

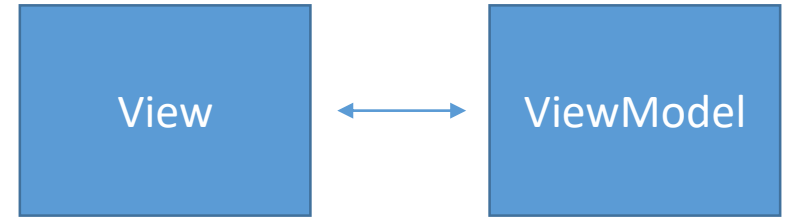
Comme la condition sur les commandes dépend de la saisie ou non du numéro animal, lancer la méthode `ChangeCanExecute()` dans le set de `AnimalNumero` :

```
((Command)ValiderNumeroAnimalCommand).ChangeCanExecute();  
((Command)EnregistrerCommand).ChangeCanExecute();
```



# Programmation orientée objet avec un projet en Xamarin

TP : faire fonctionner le bouton « Afficher... »



- Déclarer l'objet **AfficherCommand**
- Instancier cet objet
- Ecrire les méthodes d'action et de condition
- Ne pas oublier d'appeler `ChangeCanExecute()` sur le set de `AnimalNumero`



# Programmation orientée objet avec un projet en Xamarin

Pour aller plus loin...

## Changer la couleur d'un label dynamiquement (1/2) :

- Créer la classe `StringToColorConverter` (dans le dossier Converter)

```
public class StringToColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        string valueAsString = value as String;

        if (valueAsString == null) return Color.Default;

        if (valueAsString.StartsWith("#")) return Color.FromHex(valueAsString);

        if (!valueAsString.Equals("")) return System.Drawing.Color.FromName(valueAsString);

        return Color.Default;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return null;
    }
}
```



# Programmation orientée objet avec un projet en Xamarin

Pour aller plus loin...

## Changer la couleur d'un label dynamiquement (2/2) :

- Dans la vue, mettre en place le binding sur **TextColor**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    ...
    xmlns:labelTextColorSample="clr-namespace:PetitProgramme.Converter"
    x:Name="peseAnimalView">

    <ContentPage.Resources>
        <ResourceDictionary>
            ...
            <labelTextColorSample:StringToColorConverter x:Key="StringToColorConverter"/>
            ...
        </ResourceDictionary>
    </ContentPage.Resources>

    ...
<Label Grid.Row="3" Grid.ColumnSpan="2" Text="{Binding RetourValidation}"
        TextColor="{Binding CouleurMessage, Converter={StaticResource StringToColorConverter}}"/>
    ...
```

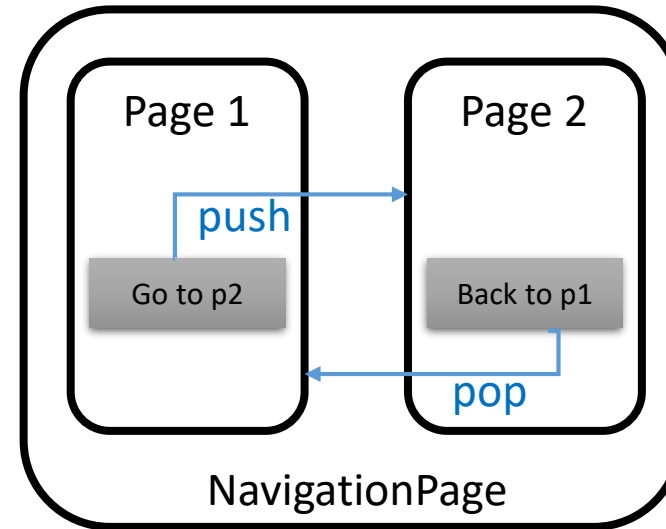
- Dans le ViewModel, ajouter la propriété **CouleurMessage** et coder ses mises à jour



# Programmation orientée objet avec un projet en Xamarin

Pour aller plus loin...

Ouvrir une page à partir d'une autre page :



- Dans le constructeur de la classe App, appeler la 1<sup>ère</sup> page en paramètre de `NavigationPage()` :

```
MainPage = new NavigationPage( new Page1());
```

- Dans le ViewModel coder le changement de page :

```
App.Current.MainPage.Navigation.PushAsync(new Page2());
```

```
App.Current.MainPage.Navigation.PopAsync();
```



# Programmation orientée objet avec un projet en Xamarin

Pour aller plus loin...

## Créer un style pour un contrôle :

```
<ContentPage.Resources>
  <ResourceDictionary>
    ...
    <x:Int16 x:Key="border">3</x:Int16>
    <Style x:Name="myButton" x:Key="myButton" TargetType="Button">
      <Setter Property="BorderWidth" Value="{DynamicResource border}"/>
      <Setter Property="CornerRadius" Value="10"/>
      <Setter Property="BackgroundColor" Value="#00A3A6"/>
      <Setter Property="BorderColor" Value="#0077A6"/>
      <Setter Property="TextColor" Value="White"/>
      <Setter Property="Margin" Value="5,0,5,0"/>
    </Style>
    ...
  </ResourceDictionary>
</ContentPage.Resources>

...

<Button Grid.Row="2" Grid.Column="1" Text="OK" Command="{Binding ValiderNumeroAnimalCommand}"
  Style="{StaticResource myButton}"/>
```