Cégep de Saint-Hyacinthe
Département d'informatique

Programmation orientée objet

420-2DP-HY

(3-3-3)

WPF - Notions de multiformulaires (modal)

(Version 1.1)

3 à 6 heures

Préparé par

Martin Lalancette

Comprendre les éléments suivants :

- Relation parent vs enfant
- Ouverture d'un formulaire en mode modal

Table des matières

Introduction

Cette séquence a pour but de vous initier aux notions de base nécessaires à la programmation à base d'objets et d'événements. Nous commencerons par énoncer les éléments théoriques appuyés d'exemples simples et faciles à reproduire. Afin d'axer l'attention sur la compréhension de ces notions, il y aura des exercices à faire tout au long de cette séquence. **La séquence portera sur une initiation aux notions de multiformulaires de type modal**.

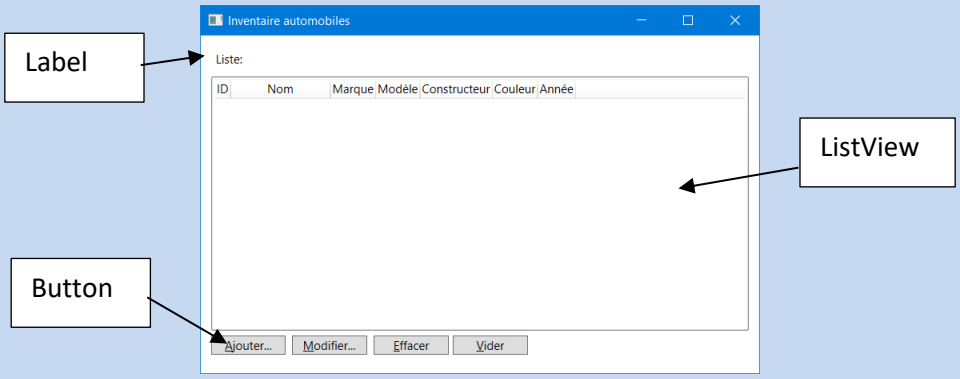
Préparation de base

Pour bien suivre les instructions qui vont être mentionnées tout au long des séquences d'apprentissage, une préparation de base s'impose. Il est important de créer un répertoire de travail (sur votre P : ou clé USB). Voici une suggestion d'arborescence :

Exercice 1. : S'assurer d'avoir créé l'arborescence ici haut mentionnée sur votre C ou votre clé USB. Copier ce document dans le répertoire en rouge ici haut mentionné.

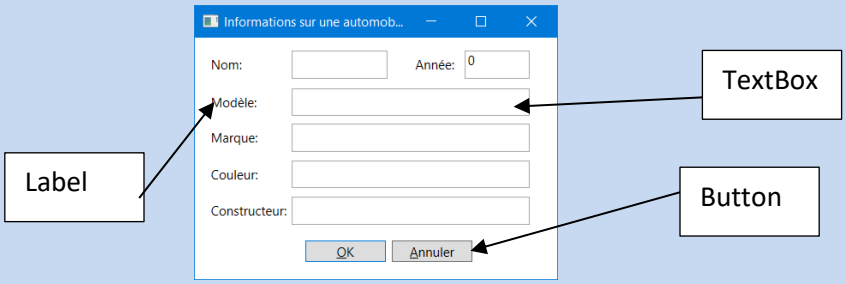
Dans la conception d'une application de type formulaire, il est courant et même nécessaire d'avoir recours à plusieurs formulaires afin de recueillir ou présenter les informations à l'utilisateur.

Exercice 2. Extraire les fichiers associés au fichier .ZIP de cette séquence. Créer une solution nommée selon le titre de ce document. Ajouter un nouveau projet WPF nommé **« InventaireAutomobiles »**. Structurer ce projet en patron MVVM. Créer un dossier **Images** et y ajouter les images extraites. Faire le design suivant dans MainWindow.xaml :



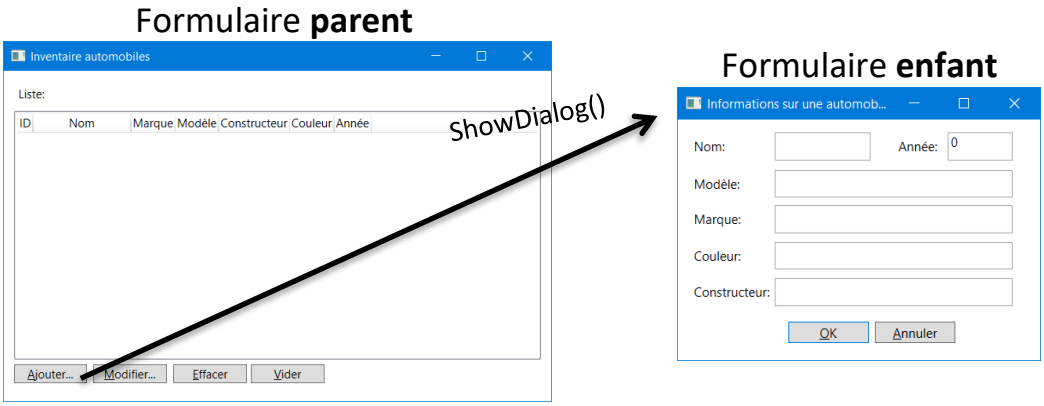
Suivre les instructions du professeur.

Exercice 3. Ajouter une nouvelle fenêtre nommée **frmAutomobile** au projet et modéliser comme suit :



Relation parent vs enfant

Avant de procéder à l'affichage d'un formulaire à partir d'un autre formulaire, il faut comprendre la relation entre les deux. Celui qui appelle l'ouverture d'un formulaire via la méthode ShowDialog() se trouve à être le **parent**, à moins d'en spécifier un autre en paramètre. Le formulaire qui est ouvert par la suite à cet appel est appelé **enfant**.



En voici un résumé :

Méthode	Description
ShowDialog()	Permet d'afficher le formulaire. L'utilisateur doit absolument fermer ce formulaire pour revenir au formulaire parent.

Ouverture d'un formulaire en mode modal

Une boîte de dialogue ou un formulaire **modal** doit être fermé avant que vous puissiez continuer à travailler dans le reste de l'application. La méthode qui permet d'afficher un formulaire en mode modal est **ShowDialog**(). Cette méthode retourne une valeur booléenne dite *nullable* (trois valeurs possibles : *true*, *false*, *null*). Pourquoi *nullable*? **La possibilité d'assigner null à des types numériques et booléens est particulièrement utile lorsque vous utilisez des bases de données et d'autres types de données contenant des éléments auxquels vous ne pouvez pas assigner de valeur**.

Retour	Description
true	Cette valeur est retournée lorsque l'utilisateur **accepte** (cliquer sur le **bouton *OK*** désigné) les informations remplies sur le formulaire.
false	Cette valeur est retournée lorsque l'utilisateur **annule** (cliquer sur le **bouton *Cancel*** désigné) les informations remplies sur le formulaire.
null	Cette valeur est retournée lorsque l'utilisateur décide de ne pas accepter ni d'annuler. Autre état → Autres boutons.

Voici deux exemples (solutions) qui utilisent le retour de la fonction Show-Dialog :

Exemple #1 : En utilisant la variable booléenne.

```
frmAutomobile frmAutoInfo = new frmAutomobile();
bool? bRetour = frmAutoInfo.ShowDialog();

if (bRetour == null) // Autres actions
{
    /* actions à faire */
}
else if (bRetour == true) // Si accepté
{
    /* actions à faire */
}
else if (bRetour == false) // Si annulé
{
    /* actions à faire */
}
```

Exemple #2 : En utilisant la variable booléenne et les propriétés HasValue et Value.

```
frmAutomobile frmAutoInfo = new frmAutomobile();
bool? bRetour = frmAutoInfo.ShowDialog();

if (!bRetour.HasValue) // Autres actions
{
    /* actions à faire */
}
else if (bRetour.Value) // Si accepté
{
    /* actions à faire */
}
else if (!bRetour.Value) // Si annulé
{
    /* actions à faire */
}
```

Exercice 4. Dans le formulaire MainWindow, ajouter l'instanciation du formulaire **frmAutomobile** dans l'événement Click du bouton **Ajouter**.

Démarrer le projet, cliquer sur bouton Ajouter…. Qu'arrive-t-il si vous cliquez sur les boutons?

Cependant, pour bien utiliser ces méthodes, il faut préparer correctement les formulaires.

Les objets de type bouton sur le formulaire

Il faut associer les codes de retour (*true*, *false* ou *null*) avec les boutons du formulaire **enfant** afin d'informer le **parent** des actions à entreprendre. Le fait d'affecter une valeur (*true* ou *false*) dans la propriété **DialogResult** d'un bouton fera fermer le formulaire lorsque l'utilisateur clique sur ce bouton. Pas besoin de faire appel à Close(). Voici les propriétés à connaître :

Propriété	Description
IsCancel	Définir un bouton avec la propriété **IsCancel** à *true* fera en sorte que lorsque l'utilisateur clique sur ce bouton (ou ESC), le formulaire se fermera et la valeur de retour DialogResult sera *false*. L'événement Click de ce bouton sera déclenché. Une boite de dialogue est annulée également lorsque (Click pas déclenché): • Appuie sur ALT+F4. • Clique sur le bouton Fermer. • Sélectionne Fermer dans le menu Système.
IsDefault	Définir un bouton avec la propriété **IsDefault** à *true* fera en sorte que lorsque l'utilisateur clique sur ce bouton (ou ENTER), l'événement Click sera déclenché. Il suffit d'y ajouter **DialogResult = true;** pour que le formulaire se ferme et retour *true*.

Exercice 5. Dans le projet précédemment ouvert, configurer adéquatement les boutons OK et Annuler du formulaire frmAutomobile en définissant les bonnes propriétés et en ajoutant le code approprié. Insérer un point d'arrêt (BreakPoint) après l'appel de la méthode **ShowDialog** pour valider les codes de retour.

Obtenir les informations du formulaire enfant

Le but premier d'ouvrir un formulaire est de demander à l'usager de saisir les informations et de pouvoir les utiliser une fois le formulaire fermé.

Création d'une classe pour recueillir les informations

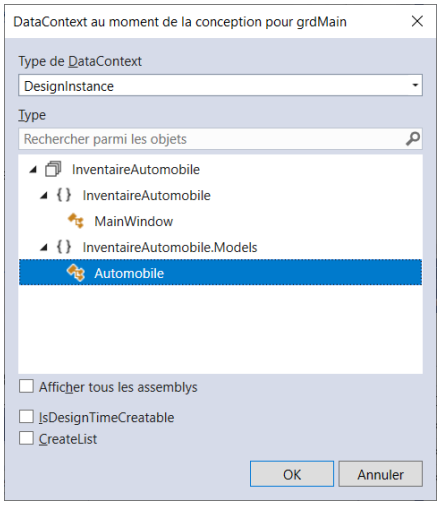
Pour obtenir ces informations, une bonne pratique est **de créer une classe avec des champs et propriétés que l'on associe aux composantes visuelles se trouvant sur le formulaire**.

Exercice 6. Dans le projet en cours. Ajouter une classe nommée **Automobile** et y définir les propriétés : **ID**, **Nom**, **Année, Modèle, Marque, Couleur** et **Constructeur**. **Recompiler le projet**.

Association des propriétés de la classe aux composantes visuelles (DataBinding)

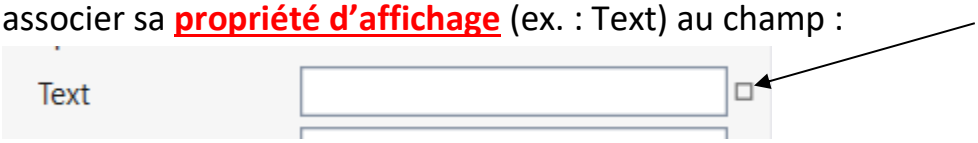
Cette classe se trouve détachée des objets graphiques. En WPF, nous pouvons associer un objet servant à recueillir les informations aux composantes visuelles grâce au *Binding* et au *DataContext*. Dans un premier temps, il faut associer le formulaire à la classe grâce à la propriété DataContex. Procédure :

1. En mode **Design**, cliquer sur la composante parent immédiat (celle qui contient l'ensemble des composantes visuelles à associer, ex. Grid). Dans le menu **Format**, choisir **Définir DataContext au moment de la conception…**
2. Dans **Type de DataContext**, choisir **DesignInstance** et la classe appropriée. Exemple :



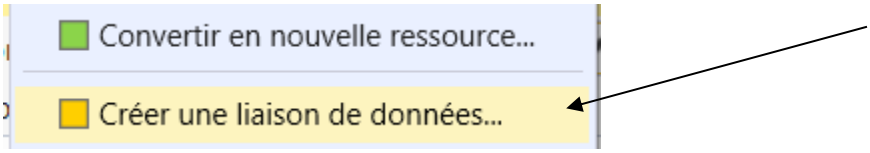
Appuyer sur OK.

3. Par la suite, sélectionner la composante visuelle (ex. TextBox) pour associer sa **propriété d'affichage** (ex. : Text) au champ :

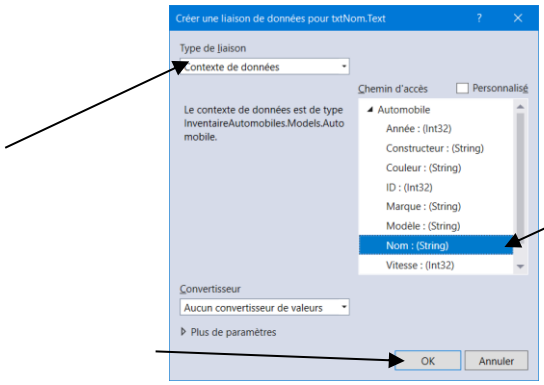


Cliquer sur le petit carré.

4. Choisir « Créer une liaison de données… ». Exemple :



5. Choisir « Contexte de données » à droite et le champ à gauche puis appuyer sur OK. Exemple :



6. Il faut répéter les étapes 4 à 6 pour les autres champs du formulaire

Exercice 7. Appliquer la procédure précédente au formulaire **frmAutomobile** .

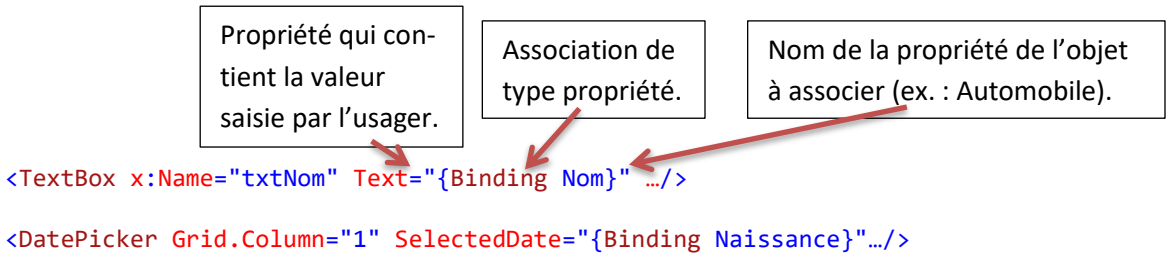
Vous pouvez également procéder par programmation pour l'association au *DataContext*. Exemple :

```
public partial class frmAutombile : Window
{
    #region Champs
    private Automobile _AutoOriginale = new Automobile();
    #endregion

    public frmAutomobile()
    {
        InitializeComponent();

        this.DataContext = _autoOriginale;
    }
}
```

1) Créer une instance de l'objet à remplir.

2) Associer l'objet au formulaire.

Vous pouvez également aller directement dans le XAML pour associer chacune des propriétés de l'objet à sa composante visuelle correspondante grâce au mot-clé *Binding*. Exemples :



Exercice 8.	Associer les propriétés de l'objet **Automobile** à chaque composante visuelle.

Au retour de l'appel du formulaire, vous pouvez obtenir les informations via la propriété DataContext. Vous pouvez également déclarer vos propres propriétés si vous le désirez.

Exercice 9.	Préparer la **ListView**. Associer les colonnes de la ListView avec les propriétés de l'objet **Automobile**. À savoir : 1) Associer le **DataContext** à Automobile 2) Dans Columns, parcourir chaque entête pour associer chaque propriétés de la classe Automobile via la propriété **DisplayMemberBinding**

Exercice 10.	Programmer le bouton **Ajouter**. Vous devez utiliser le **DataContext** et définir la propriété **ItemsSource**.

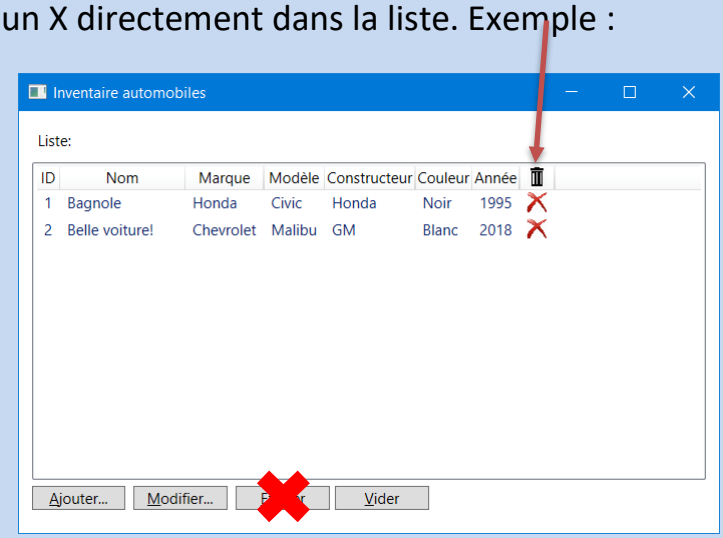
Exercice 11.	Programmer le bouton **Modifier**. Qu'arrive-t-il si vous cliquez sur le bouton Annuler?

Exercice 12. Trouver une solution pour faire fonctionner adéquatement le bouton **Annuler**. Indice : Ajouter un objet Automobile de travail qui sera responsable de contenir les informations saisies et de transférer les informations à l'objet original.

Exercice 13. Programmer le bouton Effacer.

Exercice 14. Programmer le bouton Vider.

Exercice 15. (FACULTATIF) Retirer le bouton Effacer pour le remplacer par un X directement dans la liste. Exemple :



Suivre les instructions du professeur. Indices : `GridViewColumn.HeaderTemplate`, `GridViewColumn.CellTemplate`, `DataTemplate`.

Exercice 16. : (FACULTATIF → Voir **Initiation à JSON** avant) Ajouter un bouton **Enregistrer**. Dans Click, ajouter le code nécessaire pour enregistrer la liste des automobiles dans un fichier nommé **automobiles.json**. Attention! Ici les automobiles sont contenues dans : **lstAutos.Items**. Utiliser la bibliothèque **Newtonsoft.Json.**

Exercice 17. : (FACULTATIF → Voir **Initiation à JSON** avant) Ajouter un bouton **Charger**. Dans Click, ajouter le code nécessaire pour lire le fichier **automobiles.json** et charger les données dans la liste des automobiles dans un fichier nommé **automobiles.json**. Utiliser la bibliothèque **Newtonsoft.Json.**

Bibliographie

Aucune source spécifiée dans le document actif.