**A**

**Cours du 15 novembre**

**INTRODUCTION .Net**

Professeur : Benoit CHAUVET

[contact@benoitchauvet.com](mailto:contact@benoitchauvet.com)

.Net :

C# (.cs)  
VB.net (.vb)

CIL : Common Intermediate Language (.dll, ou si main : .exe)

CLR : Common Language Runtime 🡪 équivalent de la JVM

Visual Studio : équivalent de Eclipse, c’est notre IDE

Actuellement version 6 de C#.

**Pour le projet, travailler sur une unique version de VISUAL STUDIO.**

Workspace en java = solution en visual studio, qui ne s’ouvre pas toute seule

La solution contient plusieurs projets.

Quand on crée notre premier projet, il est créé dans une solution comportant le même nom ; il est de bon ton de la renommer avec sln, pour les différencier.

Dossier dans le même nom que la solution, qui a le même nom que cette dernière mais .sln

Idem dans monProjet : un fichier monProjet.csproj

Comporte aussi un fichier App.cs

On peut aussi faire tout le code compilé dans un fichier, nommé **assembly**.(équivalent d’un .jar)

Au niveau du code :  
Comme en java les constructeurs ont même nom que la classe

Les méthodes commencent par **une majuscule !**

Public void DireBonjour()

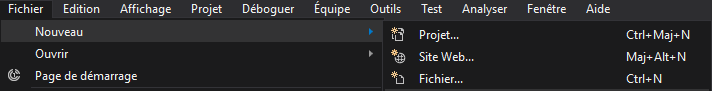
{

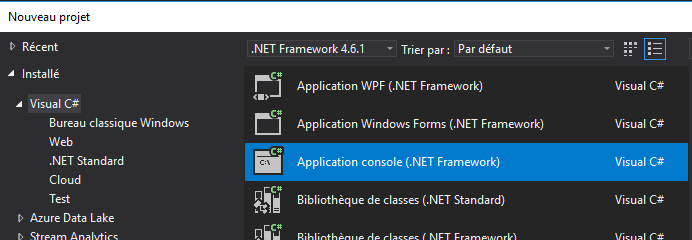
Return « bonjour » ;

}

Quant au chemin, pendant l’installation, il faut choiisr **son propre chemin !** Sinon compliqué à gérer avec l’interrogation

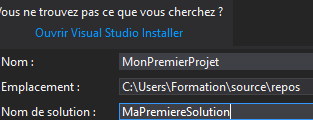
CTRL + MAJ + N pour créer nouveau fichier



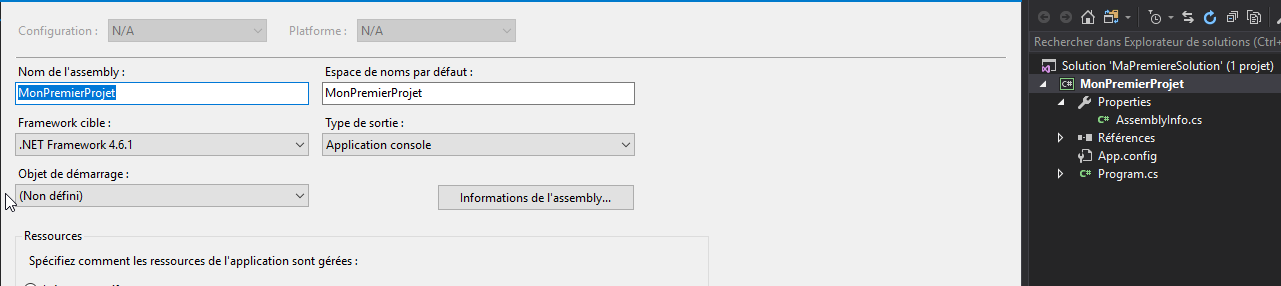


Ensuite on rename le projet (en dessous même fenêtre)

On rename ensuite la solution !



Clic droit sur le projet à droite (dans l’explorateur), pour pouvoir rename le package :



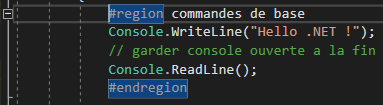
CTRL + ou ALTR + ENTER pour l’autocompletion

F5 pour lancer le projet.  
Option pour arrêter avant la fin (équivalent d’un scanner)  
  
Console.ReadLine(); équivalent du Scanner.NextLine();

« Syntax sugar » : n’est pas du tout une instruction, mais simplifie la vie du codeur,

On peut par exemple poser une « région » avec **#region** Commandes de base, terminé par **#endregion**

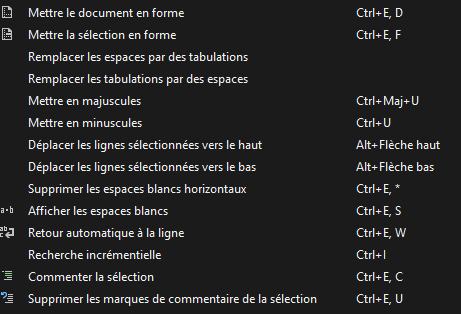
Permet d’accolader :



Devient :

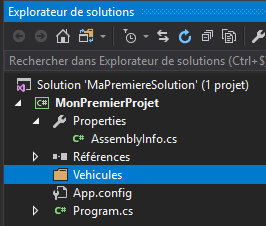


Raccourcis obtenus via Edition -> options avancées :



Nb : les TODO fonctionnent.  
Accessibles via la TASK LIST ensuite.

Sur le projet dans l’explorateur de solution, on ajoute un nouveau dossier pour créer un nameSpace



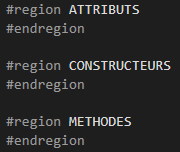
Nb : si on déplace une classe, il faut renommer le nameSpace : VS ne considère pas ça comme une erreur (comme JAVA).

Nb : si on ne met que « class », la visibilité est par défaut au minimum.  
Ainsi si on veut la classe publique, il faut mot clef « public ».

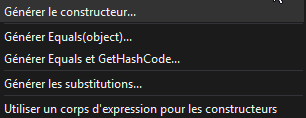
Visibilité **internal** est uniquement au sein de l’assembly (du projet).



On peut commencer par direct faire nos régions :

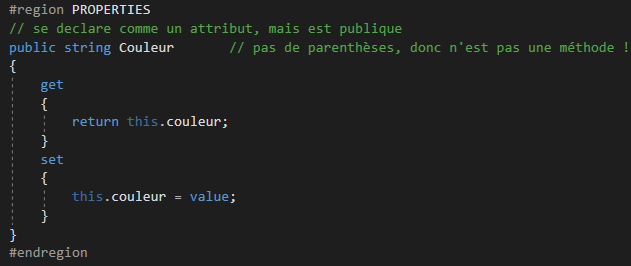


Générer les constructeurs : CTRL + ;

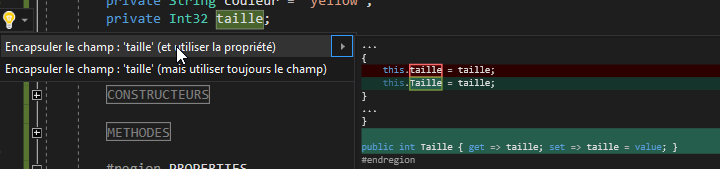


Pour les geters et seters, on les déclare en properties !  
Symbole : clé à molette.

Prop + TAB (x2) pour raccourcis (snippet)



Générer les propriétés :

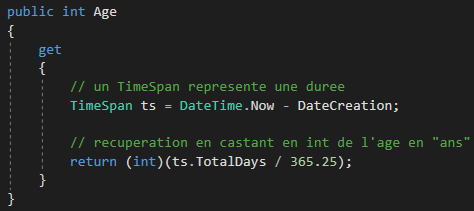


Cela donne cette écriture :



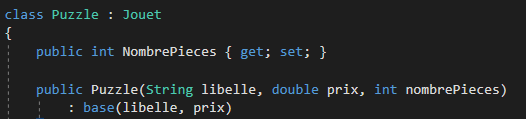
Raccourcis : **prop + tab (x2)** génère une propriété

En fait, on génère les properties (avec une majuscule), et si besoin de l’attribut on le crée.



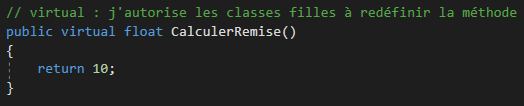
**CTRL + MAJ + B** ou **F6** : compile les erreurs et les fait apparaitre

En C#, pour l’héritage on utilise **base()** et pas **super()**

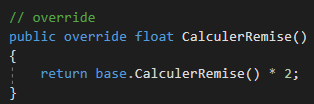
****

En C#, on ne redéfinit pas les méthodes d’une classe mère, c’est mal.  
On utilise le mot clé  **virtual**

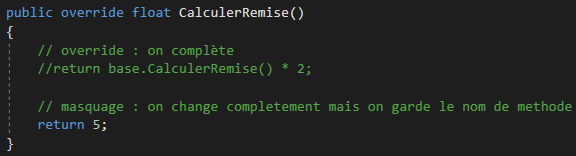
Dans la classe mère :



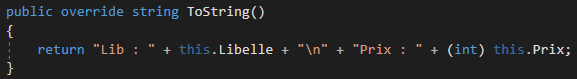
Dans la classe fille :



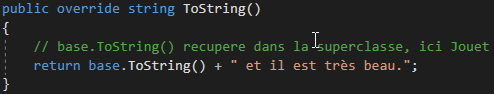
Masquage :



Exemple avec Jouet qui est base (superclasse) de Puzzle ; dans la classe Jouets :

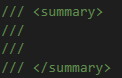


Et dans la classe puzzle, héritant de Jouets :



Allez regarder System.Speech !!

Tripple slash pour la documentation du code

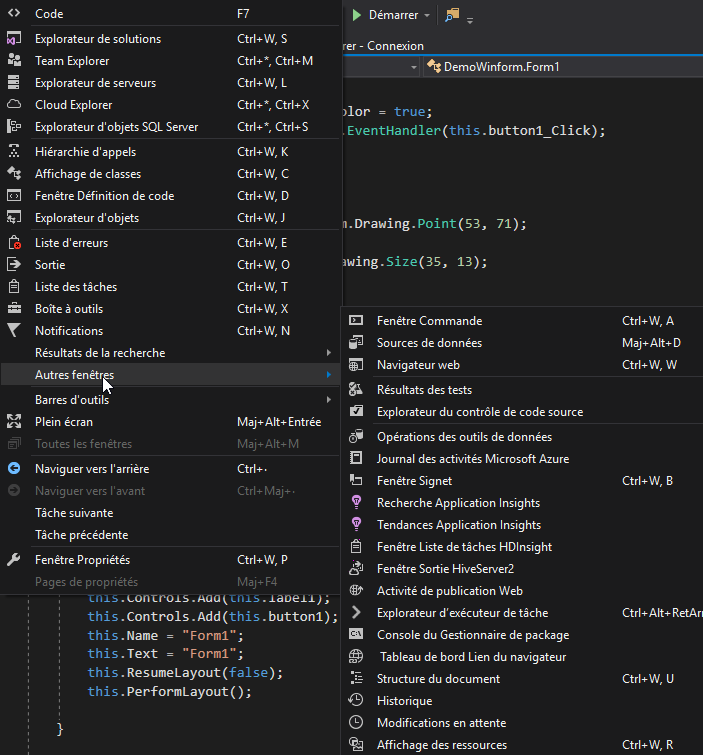


Java



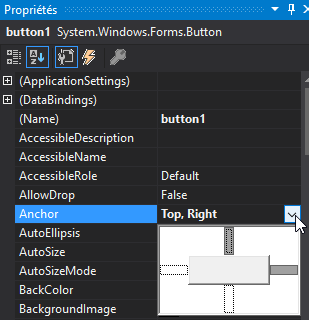
Jour 2 – 16 novembre 2017

Raccourcis :

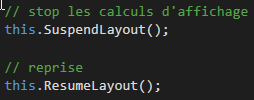


Le DockPane : équivalent du BorderPane (si j’ai bien compris), se met à droite, gauche, haut ou bas toute. Genre, collé au bord.

Anchor : pour permettre l’étirage des boutons !



Quant aux classes partielles **partial class NomClasse()** : [clic](https://docs.microsoft.com/fr-fr/dotnet/csharp/programming-guide/classes-and-structs/partial-classes-and-methods)



Dans la fenêtre de propriétés, ce qui est en gras est les valeurs changées depuis la dernière sauvegarde, donc on peut identifier si on fait de la merde ce qu’il faut corriger.



Dans la classe Form.Designer.cs (généré automatiquement) :

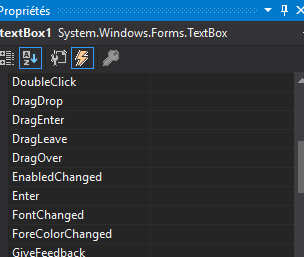


new System.EventHandler(this.button1\_Click);

Signifie en fait qu’il faut aller chercher dans la méthode, c’est un **pointeur** sur la méthode (ça ne l’appelle pas directement).

**Astuce :** **F6** est la simple compilation, **F5** ajoute aussi le lancement (post compilation).

Dans la fenêtre des properties, l’éclair représente tous les événements :



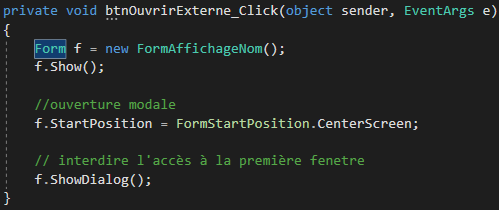
Applications MDI : Multiple Document Interface, l’application est lancée et dedans on peut avoir plusieurs classeurs (ou ouvertures de documents), mais regroupés dans un même cadre

* Ex : Excel (sur PC)

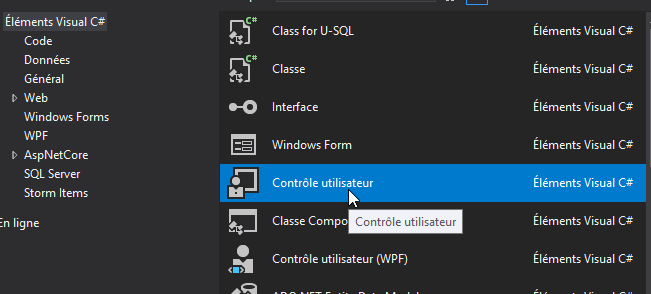
Application non MDI :

* Ex : Word

Au niveau des formulaires :



Créer un contrôle personnalisé : contrôle **utilisateur**



Jour 3 – 17 novembre 2017

On a vu :  
Winform  
Evenements  
UserControl

MDI

On va voir :

Bdd 🡪 config Oracle

ADO.NET

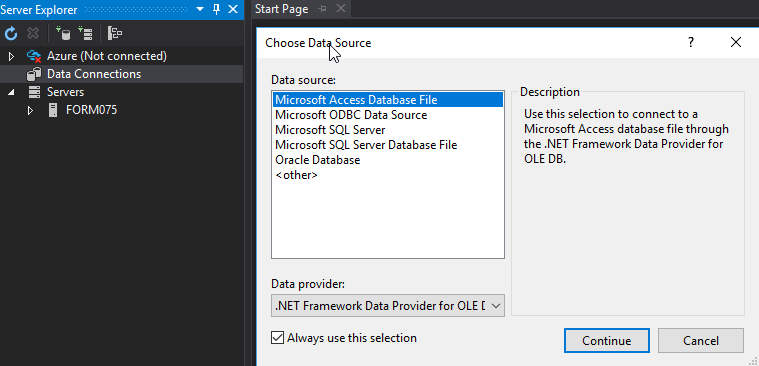
Pour se connecter à un serveur Oracle, il faut :  
Un protocole  
Un service  
Un port  
Un hôte (ip)

Ceci est dans le TNSName.Ora

Ai2@afcepf 🡪 le @afcepf est le serveur

Localhost : 127.0.0.1

Connection à la base :



ADO.NET permet de transiter entre la base de donnée et l’application qui l’utilise : « composants du framework pour la manipulation de données ».

Comme pour les imprimantes, système de pilots à installer pour communication entre les services (entre Java et MySQL, par exemple).

Nous, nous communiquons de Winform à ADO.NET, qui blablate à OLE DB ;  
Et là OLE DB correspond avec MySQL.

OLE DB est un **provider,**  comme entre autre Oracle, MySql, PostGreSQL, SQLServer :il doit fournir entre autre les classes Connection, Command, DataReader et DataAdapter.  
Toutes ces classes commencent par OleDb (avec OLE DB)

Nb : sqlConnection et pas sqlServerConnection !

Nb2 : On utilise des using

Il faut commencer par créer une Connection(avec les paramètres)  
Créer une commande 🡪 SQL

Ouvrir la connection

Exectuer la commande

Traiter le résultat

Fermer la connection

**C**reate INSERT

**R**ead SELECT

**U**pdate UPDATE

**D**elete DELETE

ArrayList en java = List en .net

Quelques méthodes :

GetAll()

GetByID()

Create()  
Update()  
Delete()

Il faut ajouter les usings :

Using System.Data.common ;  
Using System.Data.OleDb ;

Ensuite par exemple pour Delete :

Public void Delete(int id)

{

// etape 1 : creation connexion

DbConnection cnx = new DbConnection() ;

cnx.ConnectionString = « ici, mettre la chaine de connexion » ;

// etape 2 : creation commande

DbConnection cmd = new OleDbCommand() ;

cmd.CommandText = « DELETE FROM film WHERE idFilm = » + id ;

cmd.Connection = cnx ;

// etape 3 : ouverture connection

cnx.Open() ;

// etape 4 : Exectuer la commande

Cmd.executeNonQuery

// etape 5 : traiter résultat

// etape 6 :fermeture connection

cnx.Close() ;

}

Si erreur ORA : la requête est mal écrite, mais arrive bien au serveur !

Comme il faut tout de même fermer la requête, il faut impérativement un bloc try catch et surtout FINALLY sur le close !!

Exemple dans le catch :

Catch(Exception e)

{

Throw new Exception(« Erreur de suppression ») ;

}

Et le finally :

Finally

{

cnx.Close() ;

}

Voir : attaques par requêtes SQL

Astuce importante :   
Les paramètres peuvent être déclarés par des « ? », et on les redéfini ensuite via  
cmd.Parameters.Add(new Oleodb etc)

Il existe une classe ConfigurationManager, dans System.Configuration :

Using System.Configuration

(nb : dll = assembly en .net)

Le **GAC** est le **Global Assembly Cache**

Le GAC contient des dossiers

Méthod pour du read (select) :

executeReader pour renvoyer plusieurs lignes.

Les points d’arrêts marchent comme en VBA,  
Juste F8 devient F10 (pas à pas détaillé), et F11 si on veut rentrer dans une méthode, pendant son appel.

**Jour 4 – 20 novembre 2017**

On parle de « mode déconnecté » car ce n’est pas nous qui ouvrons / fermons la connexion.

DataSet est le mode déconnecté.

1 DataAdapter = 1 DataTable  
1 DataSet = \* DataTable

Etapes :

1. créer une connexion
2. créer une commande 🡪 cnx
3. créer un DataAdapter 🡪 cmd
4. créer un DataSet
5. Remplir le DataSet 🡪 da.Fill(ds, « Films ») (\*)

(\*) Il faut que le DataAdapter remplisse le DataSet, d’ou le DataAdapter.Fill

La méthode update de dataAdapter, génère les insert, génère les lignes modifiées, et effectue les delete des fichiers passés en supprimés.

Car avant, rien n’est définitif, les lignes supprimées sont en fait passées en statut « invisibles » (en gros, c’est mal dit).

En gros, le DataSet récupère les données de la base, on travaille dessus, puis on synchronise avec la base en effectuant la méthode update de DataSet !

Nb : DataTable se parcours ligne par ligne, donc en fait DataColumn spécifie un type pour la colonne mais ne récupère par une colonne… (car useless en soit)

Le mode **DataSet typé** (en Ikea) inclu le **mode déconnecté (DataSet)** qui inclu le **mode connecté**

**TP CINEMA**

[Exemple : C# DataGridView Tutorial](http://csharp.net-informations.com/datagridview/csharp-datagridview-tutorial.htm)

On crée un projet TPCinema, dedans on crée un nouveau DataSet (dans Eléments Visual C# 🡪 Données 🡪 DataSet)

On fait ensuite glisser de nos tables (explorateur de serveur) au DataSet,

On obtient le modèle physique de données (en gros).

On crée ensuite avec la boite à outil un DataGridVue (obnglet Données) ;

Voir captures d’écran.

On paramètre les Anchor à partout : histoire qu’il y ait adaptation entre la fenêtre et l’affichage.

En fait, en créant un DataSet (DataSetCinema) dans le projet, on a créé une classe du même nom : DataSetCinema. Cette classe hérite de DataSet.

De plus, des classes ont été créées : FILMDataTable par exemple (qui hérite de DataTable, en gros).

La création du DataSet se fait « à la sourie », en Ikea. Le code est généré automatiquement, il faut comprendre comment il marche.

Ici on a alimenté un DataGridVue (qui est un Winform) : dgv1.

On remarque que DataSetCinema, FILMBindingSource et FILMTableAdapter sont apparus sous la fenêtre Form1, en dessous du dgv1.

On les renomme pour plus de lisibilité :

DataSetCinema = ds

FILMTableAdapter = filmTA (de type FILMTableAdapter)

FILMBindingSource est un gadget (objet de type BindingSource) qu’on ne va pas forcément utiliser, mais très pratique par ailleurs.

Nb : attention, les tableAdapter sont dans une sous classe générée automatiquement.

Méthode InitializeComponent() :  
Il crée tous les outils.

Création du bouton MAJ :

filmTA.Update(ds.FILM) ;

**Chargement de la combobox :**  
Elle est initialisée dans le InitializeComponents()

Contrôle lié à des données, donc lié à une DataSource (qui peut être une DataTable).

Pour récupérer une DataTable de genres, par exemple, on utilise getData ;

Il nous faut donc un genreTableAdapter.

// chargement de la combobox

GENERETableAdapter genreTA = new GENERETableAdapter() ;

cbxGenre.DisplayMember = « LIBELLEGENRE » ;

cbxGenre.ValueMember = « IDGENRE » ;

cbxGenre.DataSource = genreTA.GetData() ;

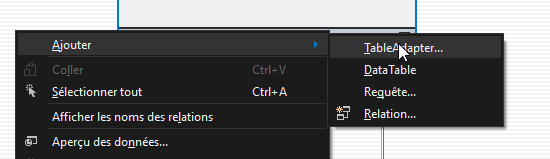
Si soucis lorsqu’on utilise un fichier d’un collègue !

*Erreur : invalid username/password ; logon denied*  
Dossier **app.config** et dans **Settings.Settings** il faut mettre à jour le mot de passe, si on récupère un fichier d’un collègue.

On commence par le app.config,  
On sauvegarde,

Normalement mise à jour automatique de Settings.settings sinon maj aussi ici.

Créer une TableAdapter : dans le « MPD », on clic droit, next next, boum.



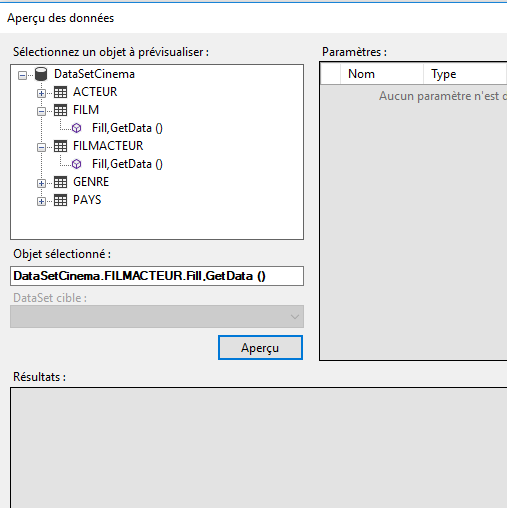
On jour sur les propriétés du GridView pour améliorer l’interface :  
**SelectionMode : FullRowSelect**  
(pour sélectionner la ligne en entier lors d’un clic sur une cellule)

**MultiSelect : false** pour bloquer la multiselection

Voir [ici](https://openclassrooms.com/courses/apprenez-a-programmer-en-vb-net/introduction-sur-les-bases-de-donnees) cours OCR concernant SQL et les DataSet.

Inner : interne.

Penser à utiliser « aperçu » dans les menus Visual Studio.



Si on a des choses qui apparaissent en double, triple etc, regarder au niveau de notre **requête SQL,** c’est surement là que ça pèche.

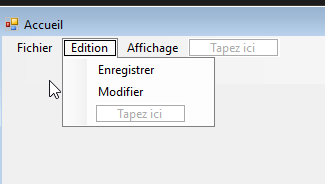
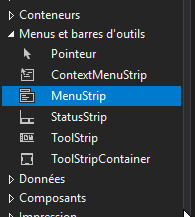
Un hack à gérer pour avoir par défaut pas un index préselectionné : on utilise une union.

**Retour aux MDI :**

Différence entre un userControl et un winform, c’est essentiellement la classe dont ils héritent.

Menu Strip :

On obtient ci-dessous :



ToolStrip.

Panel : propriété dock à FILL pour qu’il prenne toute la place.

On reprend, on veut alimenter un pannel avec des informations d’un film, qui est lui-même identifié par un IDFilm (décimal).

On peut utiliser le FlowLayoutPanel qui est un équivalent de la VBox ou HBox (à vérifier).

ASTUCE :

**SelectedRows[0] 🡪 récupération de la ligne actuelle !**En effet, on ne peut sélectionner qu’une ligne (paramètre choisi précédemment),  
Lorsqu’elle est sélectionnée, son index est paramétré à 0 (premier indice) par C#

Ligne de code qui permet de récupérer la valeur de la cellule de la colonne nommée IDFILM :

Decimal idFilm = (decimal) dvgFilm.SelectedRows[0].column[“IDFILM”].value

ASTUCE : changer la nature de l’événement ;  
Naturellement il est sur le CONTENU de la cellule. On veut qu’il porte sur un clic sur la cellule (et pas restreinte à son contenu),  
On change l’événement (avec bon orthographe).

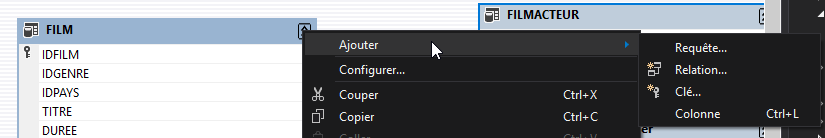
On aura une erreur, mais en pointant dessus on arrive direct sur l’erreur et on peut rename aussi.

Lors d’événements, les entête de colonnes sont indicés à Rows[-1].

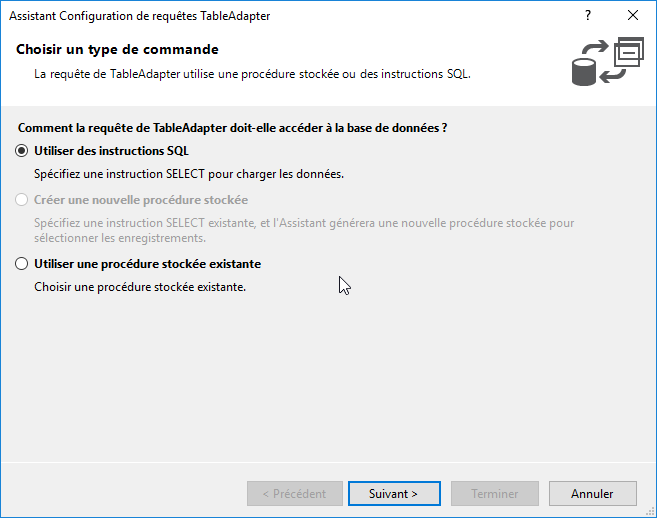
e.RowIndex récupère la ligne actuelle lors du clic.

On crée une requête SQL :

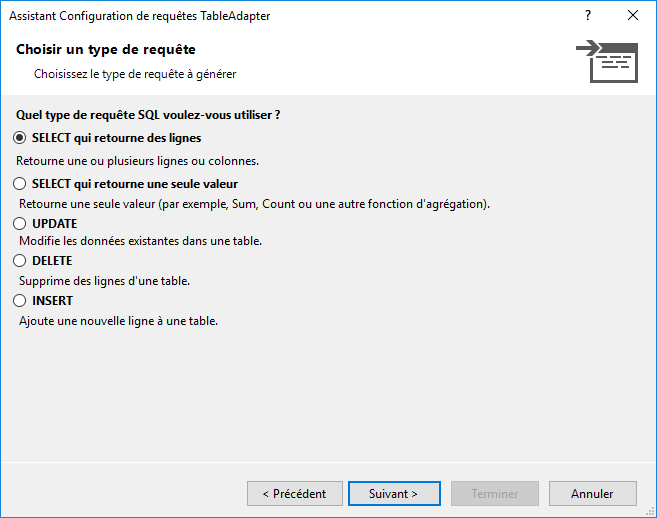
Clic droit sur Film dans le « MPD » et :



Puis



Puis



On veut rajouter dès l’appel du constructeur des films, le genre.  
Il faut créer un GENRETableAdapter.  
En effet, « un TableAdapter = un DAO ».

GENRETableAdapter gta = new GENRETableAdapter() ;

cbxGenre.DisplayMember = « LIBELLEGENRE » ; // pour choix d’affichage, en string

cbxGenre.valueMember = « IDGENRE » ;

cbxGenre.DataSource = gta.GetData() ;

Exemple d’utilisateur ternaire : renvoyer 1 si couleur, sinon 0 :

radTypeCoul.Checked ? 1 :0 ;

**CREATION D’UN EVENEMENT**

Se rappeler l’abonnement à un événement ( avec += ).

On doit être capable de crée nos propres événements, par exemple maj de la base **mais aussi de l’affichage** à gauche, dès qu’on clique sur ENREGISTRER :

ucEdit.Enregistrement += new EventHandler(ucEdit\_Enregistrement);

void ucEdit\_Enregistrement ( Object sender, EventArgs e)

{

// coder le rechargement de la liste

}

Les événements sont particuliers : on les déclare en events !

public event EventHandler Pp

Il faut ensuite le déclencher

this.Enregistrer(this, new EventArgs());

Et avant cette ligne, on met if(this.Enregistrer !=nul)

En effet, si personne n’est abonné l’événement est initialisé à « null » et cela pose problème lors de la compilation.

En gros, on s’abonne à un événement, et on fait des actions particulières si l’événement occure.

En général, créer l’abonnement au moment ou l’on crée le userControl.

On crée aussi dans la classe une méthode de traitement de l’événement.

Petit hack :

Pour un combobox avec « tous » (exemple pour obtenir la liste des sports et aussi « tous »), on doit ajouter virtuellement ce Tous (avec id = -1 par exemple) ;

En gros :  
SELECT -1 AS ‘ID’, ‘TOUS’ AS LIBELLE **FROM DUAL**

UNION  
SELECT ID, LIBELLE FROM SPORT

Requête : SELECT truc WHERE idSport = (…) OR idSport = -1

Deuxième hack qui y fait suite :  
Si on choisit justement un « tous » dans les choicebox, on récupère un -1 qui n’existe pas dans la base de donnée,  
On doit donc écrire :

SELECT …  
FROM …  
WHERE

(idsport = ? OR ? = - 1)

AND

(idville ? OR ? = -1)

Ensuite comme il y a quatre points d’interrogation ?, il faut faire ta.getData(idsport, idsport, idville, idville).

Astuce : si un UC doit communiquer avec un Form (son parent), il utilise un événément ; le parent y répond en utilisant une propriété ou une méthode.

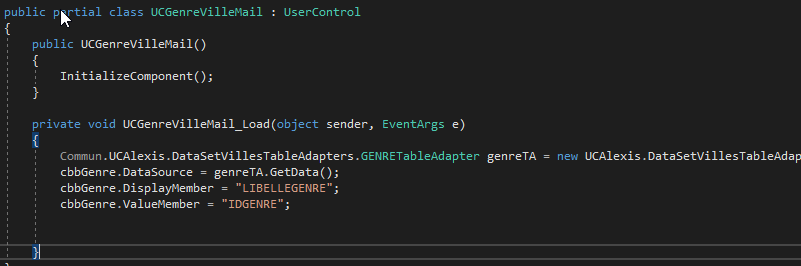
Si remplacement d’un UC lors d’un événément déclenché sur cet UC, le parent est averti par événément, puis il clear le UC, et fait ce qu’il doit faire.

EventArgs permet de récupérer les événements, et trier selon un cas.

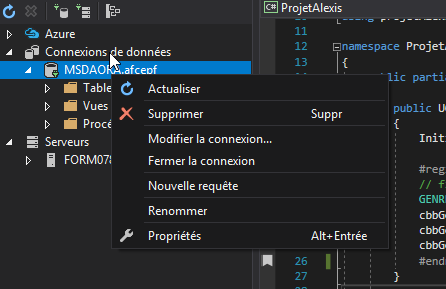
[Cours OPENCLASSROMM top WINFORM](https://openclassrooms.com/courses/apprenez-a-programmer-en-c-sur-net/les-winforms-ou-windows-forms)

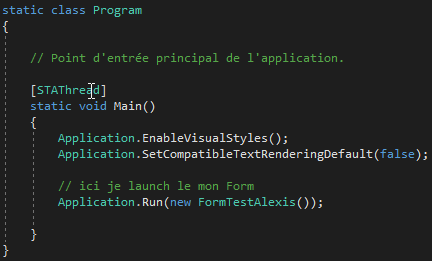
[Cours OPENCLASSROOM sur les JOINTURES](https://openclassrooms.com/courses/introduction-aux-jointures-sql)

Projet : Création d’un data set puis requête SQL,  
Et code de la récupération par combobox :

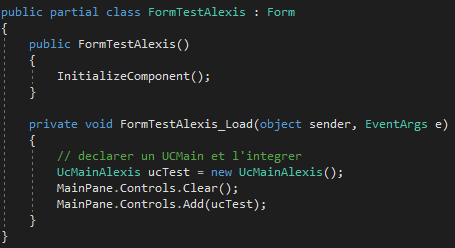


De plus il faut auparavant me connecter au serveur :

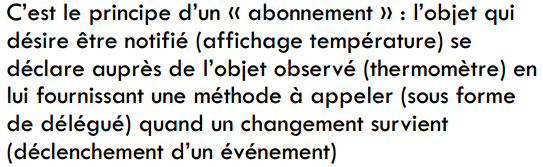


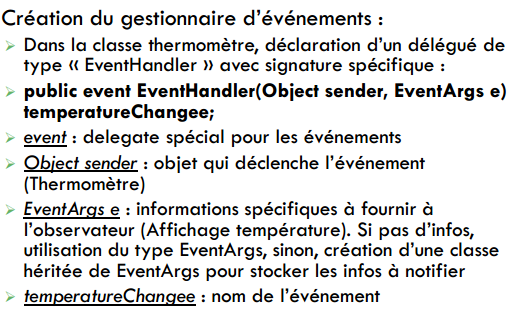
Projet :  
Dans le Main, dans Program :  


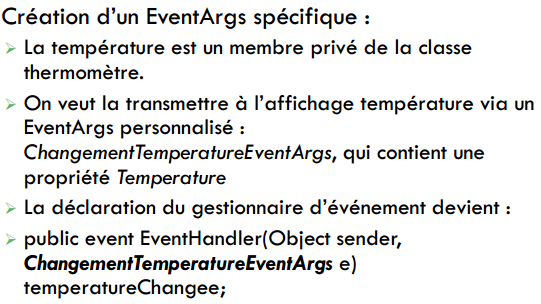
Nb : pour la capture ci après, il vaut mieux mettre plutôt dans le FormTestAlexis après le InitializeComponent()



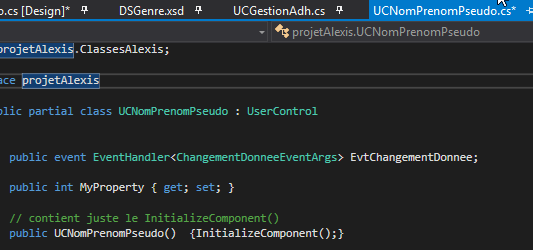
Evenement (cours pdf de Benoit):



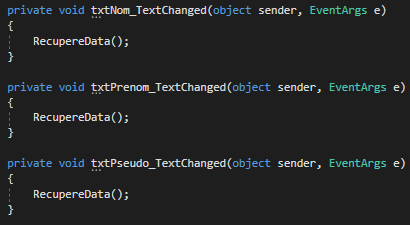




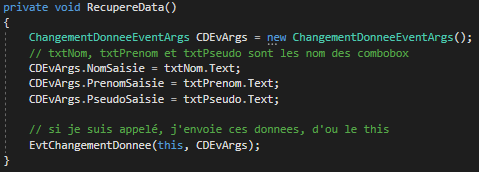
On commence par créer la méthode <> dans le UserControl concerné (ici UCNomPrenomPseudo) :



Ensuite juste après, on écrit ce code qui permet de recuperer les datas:

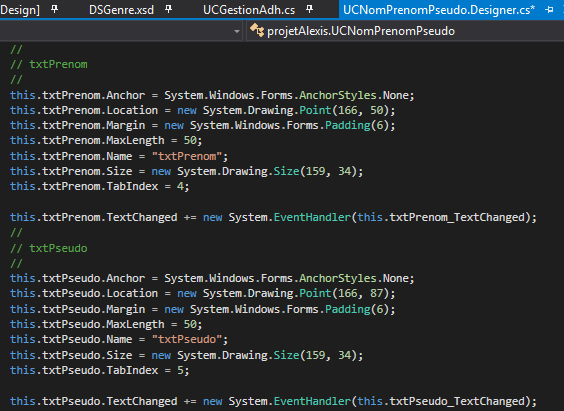


En effet, on code aussi RecupereData() :



Au niveau de mon UCNomPrenomPseudo :

Ceci est généré automatiquement, c’est l’abonnement ?



Travail sur le projet :

