BLOC2 POO	CHAPITRE 7 A la découverte des applications Windows

Objectifs :

- Savoir créer une application windows
- Maîtriser la classe d'un formulaire
- Découvrir les composants graphiques de Dotnet (Label, TextBox, ListBox, Combobox, Button)

SOMMAIRE

I. Création d'une application de type Windows Forms....2

1) Création d'un projet....2

2) L'environnement de travail....3

3) Sauvegarde et exécution du projet....4

II. Les composants graphiques d'un formulaire....5

1) Convention de nommage des composants graphiques....5

2) La propriété Name d'un composant graphique....6

3) La propriété Text d'un composant graphique....6

4) La fenêtre Propriétés....6

5) Quelle est la différence entre une combobox et une listbox ?....7

6) Affecter des valeurs dans une liste déroulante....8

7) Définir l'ordre de déplacement avec la propriété TabIndex....9

8) Assigner un rôle particulier à un bouton....10

III. En coulisse : le code généré pour décrire le formulaire....11

IV. Gerér des événements....14

1) La programmation événementielle....14

2) Présentation des classes Animal et Menagerie....14

3) Réalisation de méthodes événementielles....17

Gérer l'événement clic sur le bouton "Créer la ménagerie"....17

Afficher une boîte de dialogue avec Message.Show....18

Gérer l'événement clic sur le bouton "Ajouter l'animal dans la ménagerie"....19

V. En coulisse : le code généré pour gerer un évènement....23

VI. Contrôler les saisies sur un formulaire....24

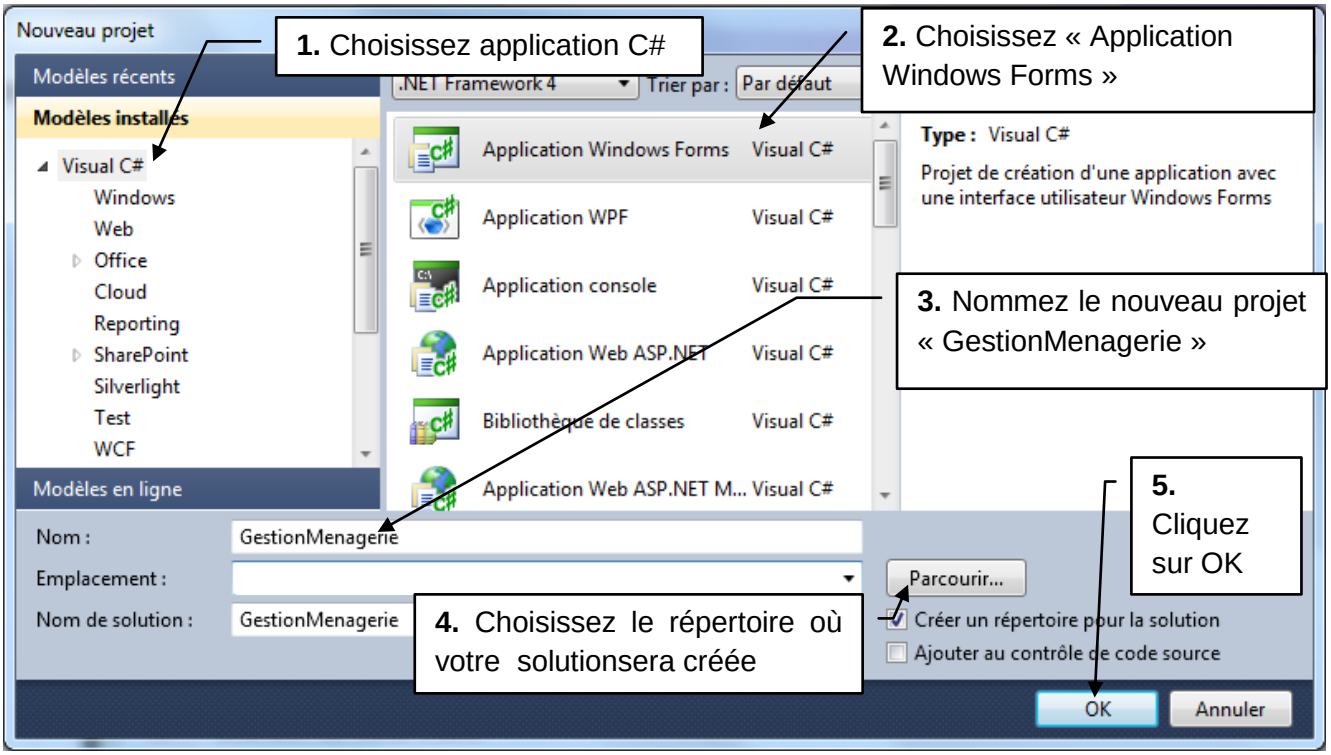
VII. Afficher des données....28

VIII. Et si vous deveniez autonome ??....29

I. CREATION D'UNE APPLICATION DE TYPE WINDOWS FORMS

1) Création d'un projet

Créez un nouveau projet de type Applixcation Windows Forms (C.F. ci-dessous)
Cette solution s'appellera « GestionMenagerie »



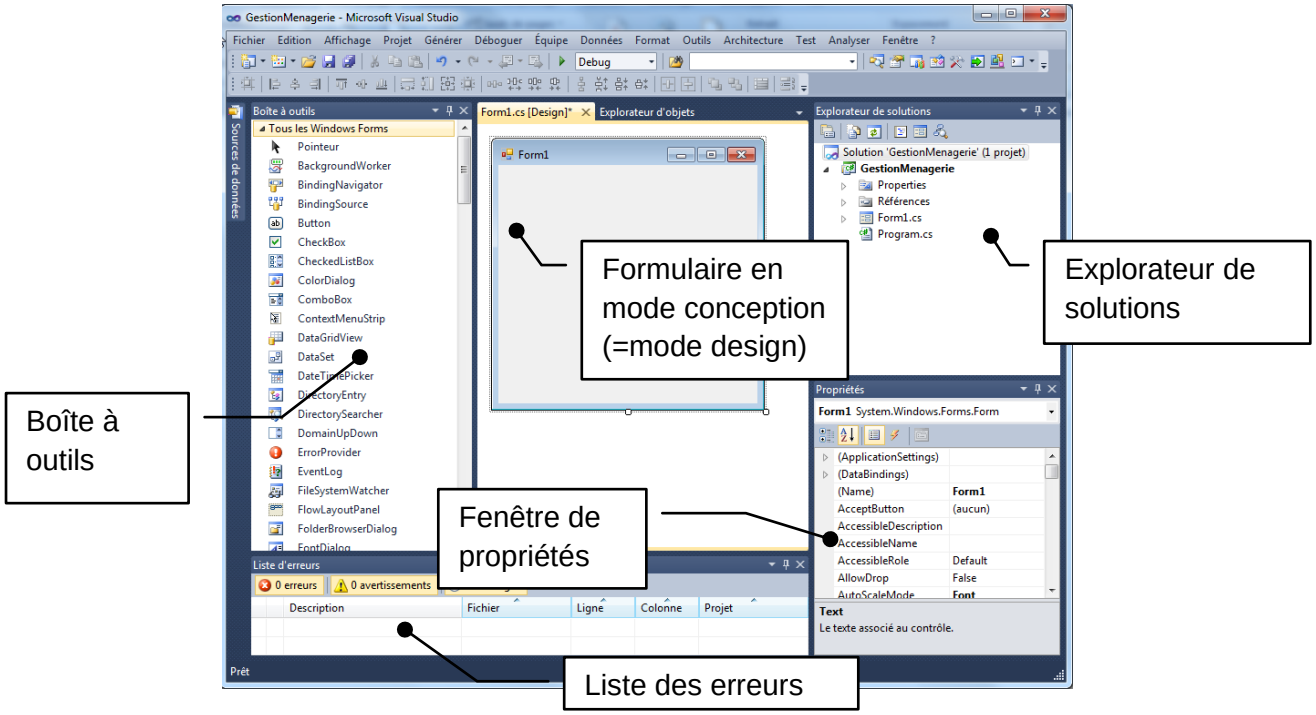
Rappel sur les solutions :

- La solution correspond à l'espace de travail du développeur,
- Une solution peut contenir plusieurs projets car on peut, par exemple, souhaiter travailler en même temps sur un projet contenant une application windows et sur une bibliothèque de classes utilisée par l'application lorsque l'on met au point ces deux projets. On peut ainsi faire évoluer l'un ou l'autre des projets sans être obligé de changer de solution.

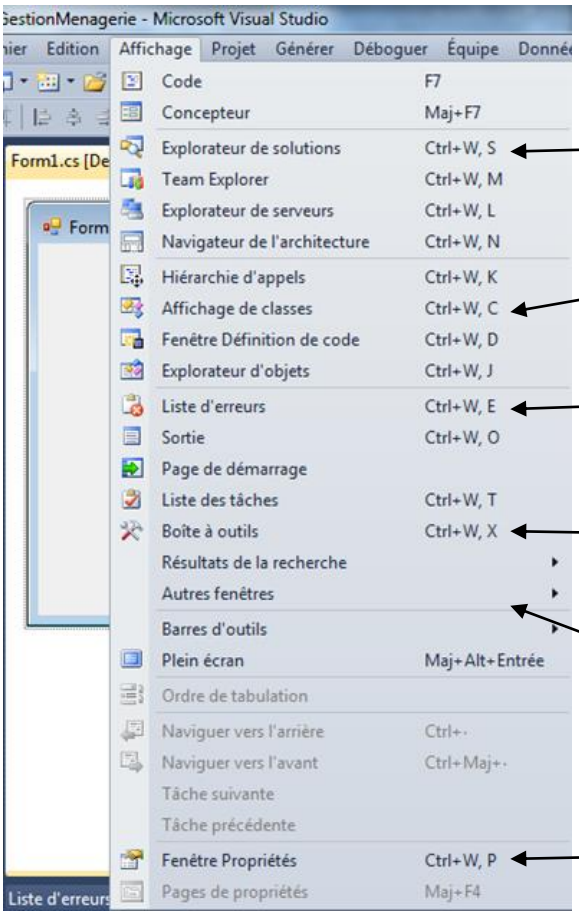
  Dans un premier temps votre solution ne contiendra que le projet application Windows Forms puis vous ajouterez un projet de type bibliothèque de classes.

2) L'environnement de travail

Voici ce que vous obtenez après avoir créé votre projet :



Rappel : **Si une des fenêtres n'apparaît pas**, vous pouvez l'obtenir en affichant le menu Affichage et en sélectionnant la fenêtre qu'il vous faut (vous pouvez également utiliser la combinaison de touches indiquée) :



Cliquez ici pour obtenir les composants de votre solution (projets, fichiers .cs, autres fichiers …)

Cliquez ici pour obtenir la liste des classes de votre projet

Cliquez ici pour obtenir la liste des erreurs

Cliquez ici pour obtenir la boîte à outils

Cliquez ici pour <u>obtenir les autres fenêtres</u>

Cliquez ici pour obtenir la fenêtre permettant d'obtenir les propriétés du contrôle sur lequel on vient de cliquer

3) Sauvegarde et exécution du projet

Procédez aux ajustements suivants :

⌨ **Renommez le fichier "Form1.cs" en "FormSaisieMenagerie.cs" :**
Attention : dans la boîte de dialogue qui s'affiche ensuite, acceptez le renommage dans tout le projet.

⌨ **Exécutez votre « projet » en cliquant sur** 

- Lors de l'exécution de votre programme, vous obtenez un formulaire vide : c'est normal car vous n'avez aucun contrôle sur votre formulaire !

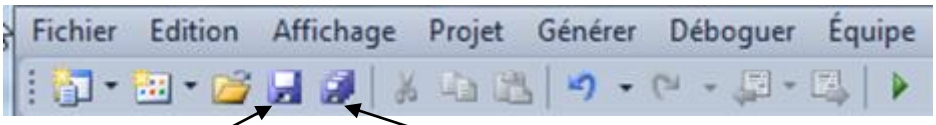
- Nous allons procéder à la création des composants graphiques qui vont nous permettre de pouvoir saisir des animaux dans notre ménagerie.

Petits rappels :

- lorsque vous demandez l'exécution de votre projet en cliquant sur Déboguer+Démarrer le débogage ou Déboguer+Démarrer le débogage ou encore plus simplement en cliquant sur  ou sur la touche F5 ou ctrl F5, que se passe-t-il ? :
  - une **sauvegarde** de la totalité des fichiers de votre projet est réalisée
  - **puis** la **compilation** est réalisée (un assembly est créé = .exe ou .dll)
  - **puis** le programme commence **à s'exécuter**

- lorsque vous demandez la génération (=compilation) de votre projet en cliquant sur Générer + Générer la solution ou plus simplement en appuyant sur la touche F6 :
  - une **sauvegarde** de la totalité des fichiers de votre projet est réalisée
  - **puis** la **compilation** est réalisée

- vous pouvez à tout moment demander de sauvegarder les fichiers de votre projet (il est recommandé de faire cette opération régulièrement pour ne pas perdre votre travail en cas de coupure de courant, de problème sur le réseau, de plantage de visual studio…) :

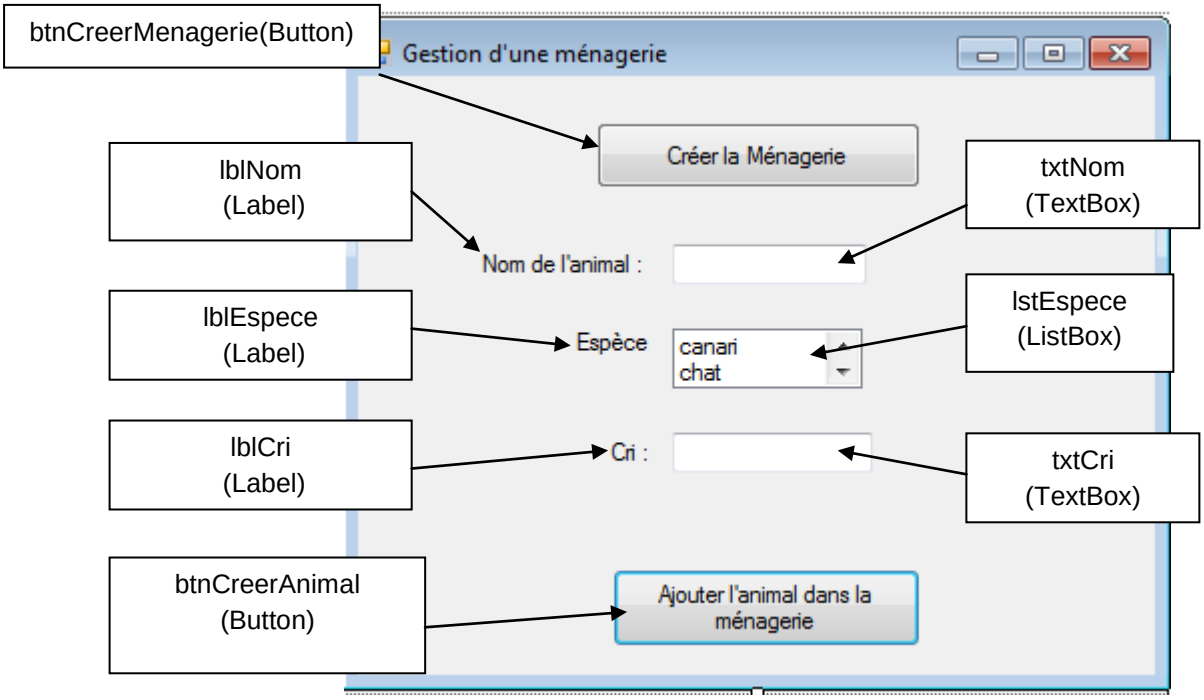


Cliquez ici pour enregistrer **le fichier sur lequel vous êtes en train de travailler**. Vous pouvez également utiliser la combinaison de touches Ctrl+S

Cliquez ici pour enregistrer **tous** les fichiers de votre projet. Vous pouvez également utiliser la combinaison de touches Ctrl+Maj+S

II. LES COMPOSANTS GRAPHIQUES D'UN FORMULAIRE

Voici les composants graphiques que vous devez placer sur le formulaire :



1) Convention de nommage des composants graphiques

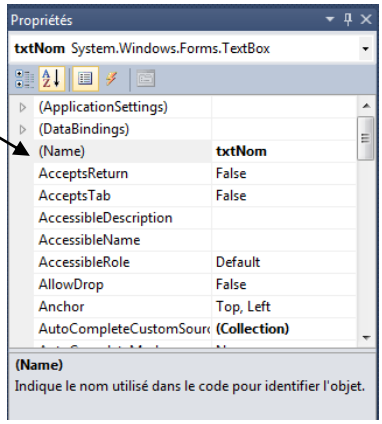
Il n'y a pas de règle unique pour le nom des composants graphiques **(appelés également contrôles graphiques)**. En général on associe un préfixe , représentant le type du composant, à un nom explicite représentant l'objet du composant.

Voici les préfixes de quelques composants que vous allez utiliser :

Préfixe	Nom du composant graphique dans Visual Studio	Nom en français
txt	TextBox	Zone de texte (champ de saisie libre)
lbl	Label	Etiquette (intitulé sur un formulaire)
lst	ListBox	Zone de liste
cbx	ComboBox	Zone de liste modifiable
btn	Button	Bouton de commande
chk	CheckBox	Case à cocher
opt	RadioButton	Bouton à option ou radio bouton
dtg	DatagridView	Grille de données
pnl	Panel	Contrôle contenant un ensemble de contrôles

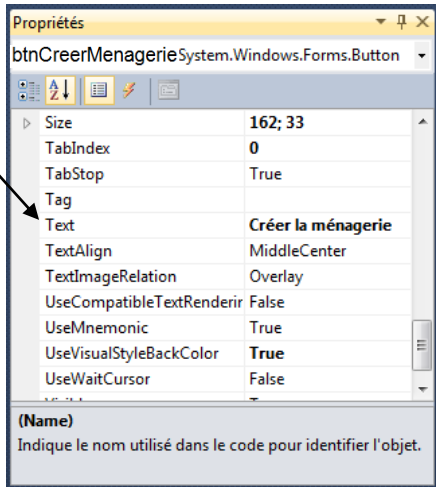
Remarque : le nom d'un formulaire commencera par **Form (on peut aussi utiliser Frm)**

2) La propriété Name d'un composant graphique



- Le nom d'un composanst est contenu dans la propriété « (Name) » en haut de la liste alphabétique
- Le préfixage en fonction du type de composanst est une bonne habitude à prendre. Pour l'instant vous respecterez impérativement la norme que je vous ai proposé ci-dessus : ici « **lst** » pour **L**ist**B**o**x**, « **btn** » pour **B**utto**n**, « **txt** » pour **T**ext**B**o**x**.
- Pour afficher la liste des propriétés d'un contrôle (si celle-ci n'est pas apparente), cliquez sur le contrôle avec le bouton gauche de la souris, et choisir Propriétés dans le menu contextuel (clic droit)

3) La propriété Text d'un composant graphique



La plupart des contrôles ont une propriété « **Text** » qui permet de modifier le texte affiché sur le contrôle.

- la propriété « **Text** » d'un composant graphique de type button contient le texte affiché sur le bouton ; vous pouvez indiquer le texte que vous voulez
- par défaut, la propriété « **Text** » d'un composant graphique de type textbox est vide.
- cas des formulaires ; par défaut, la propriété « **text** » contient le nom du formulaire, modifiez la pour afficher un intitulé plus signification pour le formulaire : Saisie de …, Consultation de … etc.
- Quant à la zone de liste (listbox), il n'y a pas de propriété « **Text** » accessible en mode conception. Ne vous inquiétez pas du fait que le nom du contrôle s'affiche.

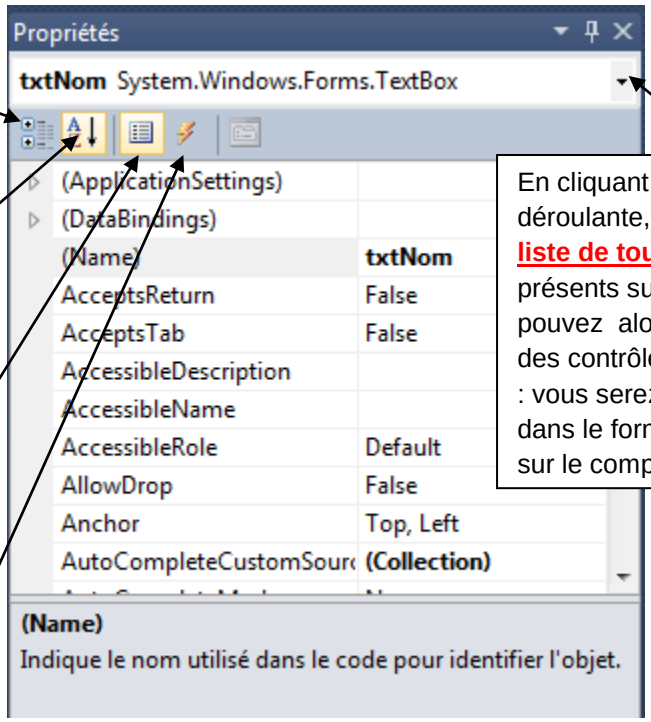
4) La fenêtre Propriétés

Cliquez sur ce bouton pour obtenir un affichage des propriétés **par catégorie**

Cliquez sur ce bouton pour obtenir un affichage des propriétés **par ordre alphabétique**

Cliquez sur ce bouton pour obtenir **la liste des propriétés** associées à un contrôle

obtenir **la liste des évènements** associés à un composant (C.F ci-après)



En cliquant sur cette liste déroulante, vous allez obtenir la **liste de tous les contrôles** présents sur le formulaire. Vous pouvez alors sélectionner un des contrôles en cliquant dessus : vous serez alors positionné dans le formulaire directement sur le composant

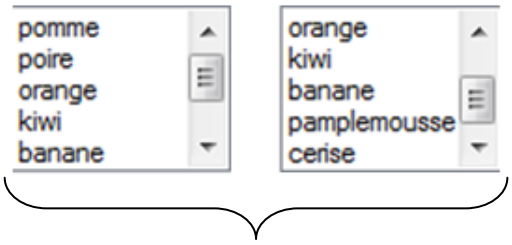
Informations sur la **signification de la propriété**

5) Quelle est la différence entre une combobox et une listbox ?

- Une listbox contient une **liste d'éléments**. La listbox vous permet de choisir un élément dans une liste déterminée.

- Une combox box ou "zone de liste modifiable" est composée **d'un champ de saisie combiné à une liste d'éléments.** La combobox vous permet de choisir dans une liste déterminée mais vous autorise également à saisir une valeur dans le champ de saisie.

- La différence est aussi d'ordre esthétique :
  - la listbox montre d'emblée les éléments de la liste (si ça ne tient pas dans la zone affichée, un ascenseur apparaît : on peut alors obtenir les autres éléments en utilisant l'ascenceur),
  - la combobox vous propose une flèche et un menu déroulant pour voir les éléments de la liste.



Exemple de ListBox



Exemple de ComboBox

- Ces deux listes proposent une aide à la saisie : si vous saisissez le début d'une valeur qui existe dans la liste (par exemple "poi"), vous serez automatiquement positionné sur la première valeur commençant par "poi" par exemple "poisson rouge"

- La propriété DropDownStyle pour une combobox :

  - Par défaut cette propriété contient la valeur **DropDown** : cette valeur autorise un utilisateur à saisir une valeur qui n'existe pas dans la liste, cela peut être utile dans certains cas si on veut que l'utilisateur ajoute de nouvelles valeurs dans la combobox

  - Mais on peut dans d'autres cas utiliser une combobox pour son aspect graphique et refuser la saisie d'une valeur inexistante : dans ce cas la propriété DropDownStyle doit contenir **DropDownList**

➢ **A vous !!!**

Insérer les contrôles adéquats (Label, ListBox, Button, TextBox) sur le formulaire et nommer-les comme indiqué sur la page précédente

➢ Pour ajouter un contrôle, double-cliquez sur le contrôle souhaité dans la boîte à outils, ou faites-un glisser-déposer du composant graphique sélectionné dans la boîte à outils vers votre formulaire.

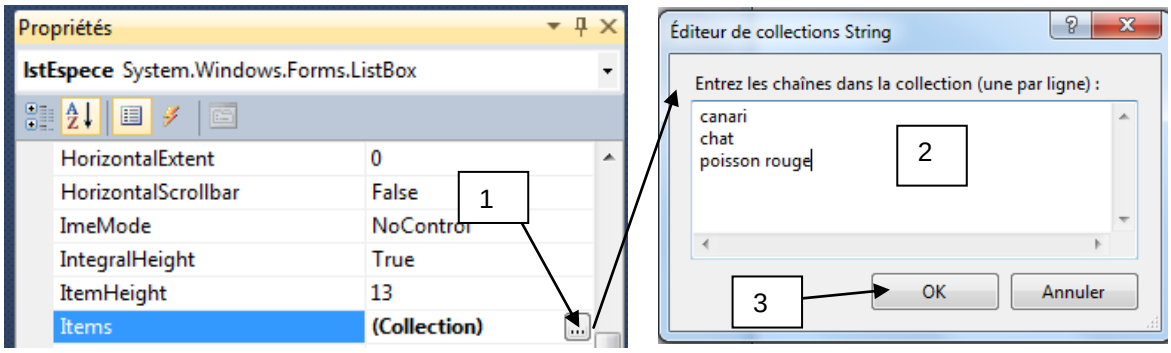
➢ Vous trouverez tous les contrôles dans la boîte à outils, sous la rubrique "Contrôles communs"

6) Affecter des valeurs dans une liste déroulante

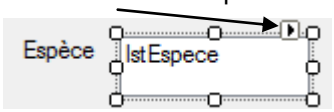
- Pour affecter des valeurs à votre liste (zone de liste ou zone de liste modifiable) vous avez **deux possibilités** (vous devrez donc choisir entre une des deux !!!) :

  - **utilisez le mode Design** du formulaire (ce que vous allez faire) : cliquez sur votre ListBox, puis dans la fenêtre Propriétés cliquez sur le bouton situé en face de la propriété **Items** puis saisissez les valeurs que vous souhaitez voir s'afficher dans votre liste et appuyez sur OK

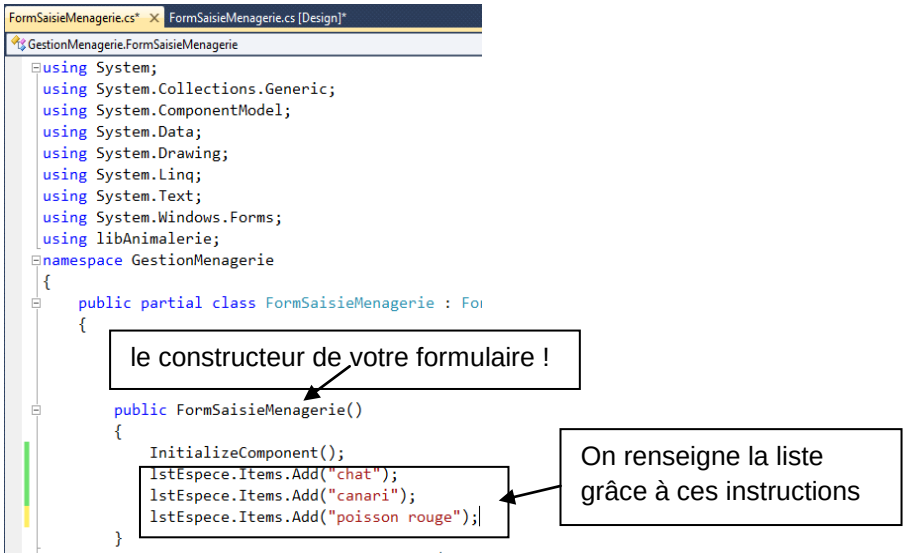
OU



Vous pouvez également accéder directement à l'écran "Editeur de collections String" en cliquant sur la flèche située en haut de la liste quand la liste est active sur le formulaire :

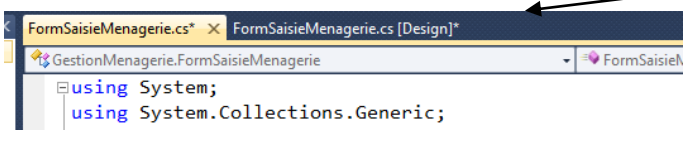


  - **utilisez des instructions** pour ajouter les valeurs dans la liste.
Mais quelles sont ces instructions et où les saisir ? Vous devez obtenir la classe qui correspond au formulaire : pour cela demandez le menu contextuel lorsque vous êtes positionné sur le formulaire (clic droit) puis choisissez "Afficher le code" … ou plus simplement appuyez sur la touche F7 ! Vous obtenez :



Items est une propriété de type collection de la ListBox : c'est elle qui contient les valeurs qui sont affichées dans la liste. Add est la méthode que vous connaissez bien maintenant qui permet d'ajouter un élément dans une collection.

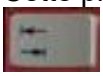
- pour que les valeurs de votre liste **apparaissent triées** modifiez la propriété **"Sorted"** de votre liste pour lui donner la valeur **True**
- Pour revenir au formulaire en mode conception cliquez sur l'onglet



Pour passer du mode conception au code source et vice versa : cliquez alternativement sur ces onglets ou utilisez les touches F7 et Maj F7

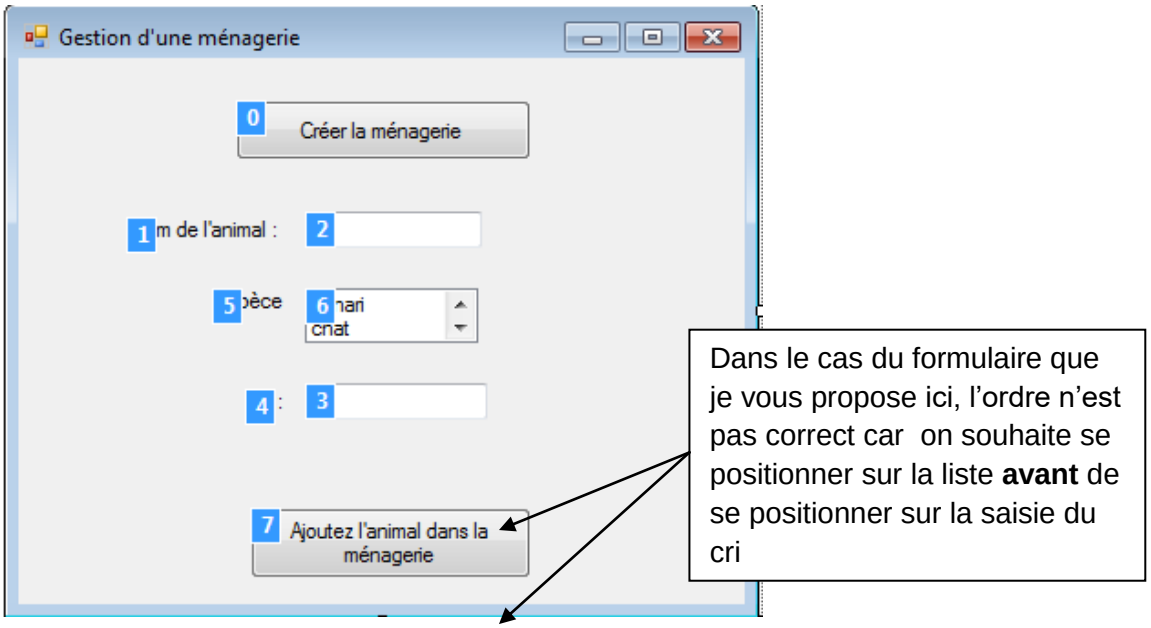
7) Définir l'ordre de déplacement avec la propriété TabIndex

- Cette propriété définit l'ordre de déplacement du lorsque l'utilisateur utilise la touche Tab

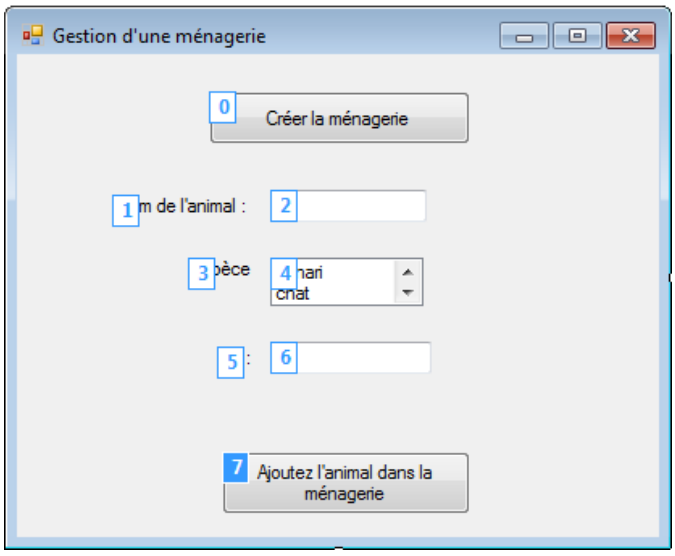


- Par défaut, les valeurs des propriétés TabIndex des composants sont numérotées dans l'ordre selon lequel les composants sont créés.

- Pour modifier cet ordre, vous pouvez modifier directement la propriété TabIndex de chaque composant (c'est un peu fastidieux …) ou utiliser le menu **Affichage – Ordre de tabulation** (c'est plus intelligent !). Les composants sont alors affichés avec, dans leur coin supérieur gauche, la valeur de leur propriété TabIndex.



Pour modifier l'ordre de tabulation, vous devez alors cliquer sur les composants, dans l'ordre selon lequel vous voulez que le curseur se déplace lorsque vous appuyerez sur la touche <Tab>.

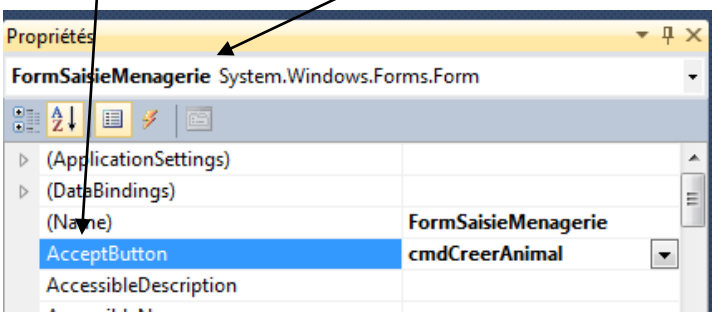


Pour revenir en mode normal, utilisez à nouveau **Affichage – Ordre de tabulation** ou appuyez sur la touche <Echap>

8) Assigner un rôle particulier à un bouton

- Faites en sorte que le bouton "Ajouter l'animal dans la ménagerie" soit le bouton d'acceptation du formulaire, c'est-à-dire **celui qui est « activé »** quand l'utilisateur appuie sur la **touche <Entrée>.**

  Modifiez la propriété **AcceptButton** dans les **propriétés du formulaire** de la manière suivante :



A ce stade, lorsque vous demandez l'exécution de votre application, un formulaire s'affiche mais si vous appuyez sur les boutons, rien ne se passe : aucune ménagerie n'est créée, et aucun animal ne peut y être ajouté … C'est normal car vous devez saisir le code pour chacun de ses évènements !!

Vous allez maintenant ajouter les instructions qui vont permettre ces traitements.

III. EN COULISSE : LE CODE GENERE POUR DECRIRE LE FORMULAIRE

Le développement d'une application « WindowsForms » est presque magique… mais allons dans les coulisses pour comprendre comment le magicien Visual Studio effectue ses tours de magie.

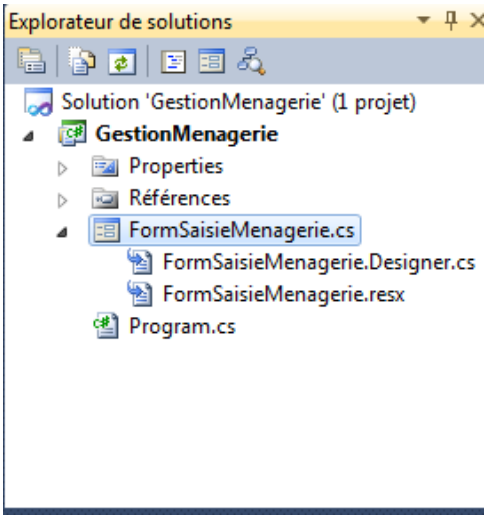
Si on regarde dans l'explorateur de solution, on trouve plusieurs fichiers :

- Le fichier **FormSaisieMenagerie.cs** : c'est dans ce fichier que vous allez ajouter des instructions pour réaliser des traitements spécifiques

- Le fichier **FormSaisieMenagerie.Designer.cs** : comme son nom l'indique il contient tout ce que l'IDE a généré suite à ce que vous avez fait en mode « design ».

- Le fichier **FormSaisieMenagerie.resx** : il décrit les ressources éventuellement associées.

- Le fichier **Program.cs** qui est responsable du lancement de l'application (comme pour les applications console).



Regardons successivement et rapidement le contenu de ces différents fichiers, en commençant par le dernier.

Contenu du fichier Program.cs

Program.cs | Explorateur d'objets | FormSaisieMenagerie.cs | FormSaisieMenagerie.cs [Design] | Animal.cs

GestionMenagerie.Program | Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace GestionMenagerie
{
    static class Program
    {
        /// <summary>
        /// Point d'entrée principal de l'application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FormSaisieMenagerie());
        }
    }
}
```

La méthode Main = point d'entrée de l'application.

La méthode Main crée un objet, instance de la classe FormSaisieMenagerie avec l'instruction *new* et lance l'instruction *Run* ce qui a pour effet d'afficher le formulaire à l'écran.

Contenu du fichier FormSaisieMenagerie.cs

```
using System.Text;
using System.Windows.Forms;
using libAnimalerie;
namespace GestionMenagerie
{
    public partial class FormSaisieMenagerie : Form
    {

        public FormSaisieMenagerie()
        {
            InitializeComponent();
        }

    }
}
```

Le code de la classe **FormSaisieMenagerie** qui décrit le formulaire est réparti **dans 2 fichiers** : le fichier FormSaisieMenagerie.cs et le fichier FormSaisieMenagerie.Designer.cs

partial : indique que ce n'est qu'une partie du code de la classe.

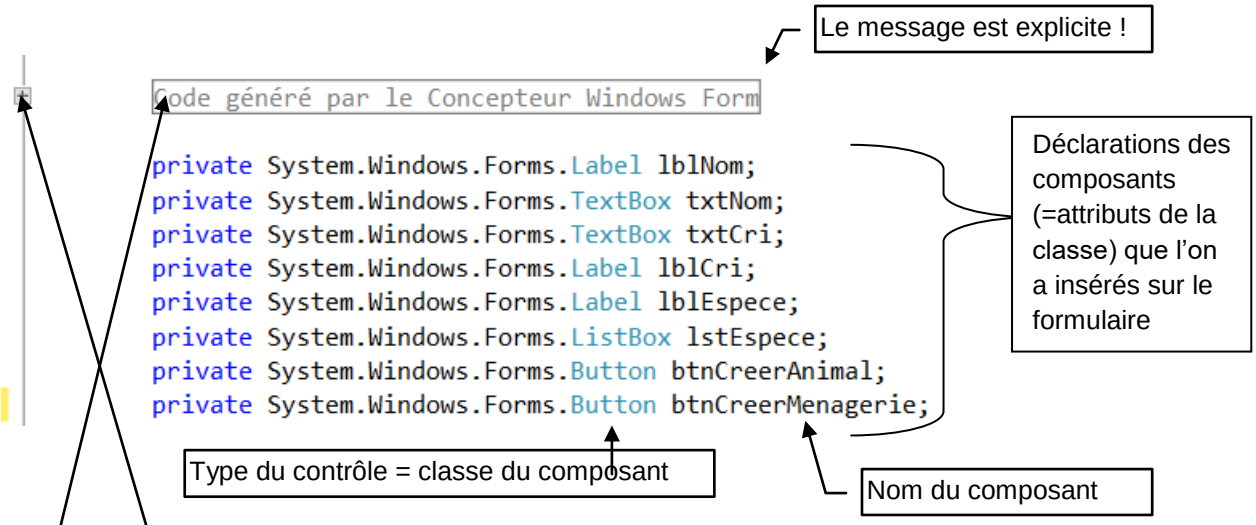
: Form indique que **FormSaisieMenagerie** hérite de la classe Form, ce qui permet au développeur d'utiliser des fonctionnalités propres aux formulaires, sans tout réécrire.

Appel d'une méthode InitializeComponent qui se trouve dans le fichier **FormSaisieMenagerie.Designer.cs**

Quelle est donc cette méthode ???

Contenu du fichier FormSaisieMenagerie.Designer.cs

Je ne vais pas présenter en détail tout le code contenu dans ce fichier. Mais quelques précisions peuvent vous aider :



En fait le code généré est réduit par défaut pour éviter qu'on y mette ses petites mains destructrices. …. mais si on était curieux et qu'on allait y jeter un petit coup d'œil…



ATTENTION… soyez prudent car une mauvaise manipulation pourrait avoir des conséquences fâcheuses !

Cliquer sur le petit ⊞ pour développer le contenu de cette partie de code.

L'aperçu ci-dessous ne montre qu'une partie de ce code :

```
#region Code généré par le Concepteur Windows Form

/// <summary>
/// Méthode requise pour la prise en charge du concepteur - ne modifiez pas
/// le contenu de cette méthode avec l'éditeur de code.
/// </summary>
private void InitializeComponent()
{
    this.lblNom = new System.Windows.Forms.Label();
    this.txtNom = new System.Windows.Forms.TextBox();
    this.txtCri = new System.Windows.Forms.TextBox();
    this.lblCri = new System.Windows.Forms.Label();
    this.lblEspece = new System.Windows.Forms.Label();
    this.lstEspece = new System.Windows.Forms.ListBox();
    this.btnCreerAnimal = new System.Windows.Forms.Button();
    this.btnCreerMenagerie = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // lblNom
    //
    this.lblNom.AutoSize = true;
    this.lblNom.Location = new System.Drawing.Point(88, 134);
    this.lblNom.Margin = new System.Windows.Forms.Padding(4, 0, 4, 0);
    this.lblNom.Name = "lblNom";
    this.lblNom.Size = new System.Drawing.Size(132, 20);
    this.lblNom.TabIndex = 1;
    this.lblNom.Text = "Nom de l\'animal : ";
    //
    // txtNom
    //
    this.txtNom.Location = new System.Drawing.Point(236, 129);
    this.txtNom.Margin = new System.Windows.Forms.Padding(4, 5, 4, 5);
    this.txtNom.Name = "txtNom";
    this.txtNom.Size = new System.Drawing.Size(144, 26);
    this.txtNom.TabIndex = 2;
```

[… suite des composants ….]

```
    //
    // lblEspece
    //
    this.lblEspece.AutoSize = true;
    this.lblEspece.Location = new System.Drawing.Point(159, 194);
    this.lblEspece.Margin = new System.Windows.Forms.Padding(4, 0, 4, 0);
    this.lblEspece.Name = "lblEspece";
    this.lblEspece.Size = new System.Drawing.Size(63, 20);
    this.lblEspece.TabIndex = 3;
    this.lblEspece.Text = "Espèce";
    //
    // lstEspece
    //
    this.lstEspece.FormattingEnabled = true;
    this.lstEspece.ItemHeight = 20;
    this.lstEspece.Items.AddRange(new object[] {
    "canari",
    "chat",
    "poisson rouge"});
    this.lstEspece.Location = new System.Drawing.Point(236, 194);
    this.lstEspece.Margin = new System.Windows.Forms.Padding(4, 5, 4, 5);
    this.lstEspece.Name = "lstEspece";
    this.lstEspece.Size = new System.Drawing.Size(140, 44);
    this.lstEspece.Sorted = true;
    this.lstEspece.TabIndex = 4;
    this.lstEspece.Tag = "choix de l\'espèce";

    //
    // btnCreerAnimal
    //
    this.btnCreerAnimal.Location = new System.Drawing.Point(190, 378);
    this.btnCreerAnimal.Margin = new System.Windows.Forms.Padding(4, 5, 4, 5);
    this.btnCreerAnimal.Name = "btnCreerAnimal";
    this.btnCreerAnimal.Size = new System.Drawing.Size(231, 60);
    this.btnCreerAnimal.TabIndex = 7;
    this.btnCreerAnimal.Text = "&Ajouter l\'animal dans la ménagerie";
    this.btnCreerAnimal.UseVisualStyleBackColor = true;
```

Attention à la mise en garde !

C'est la méthode qui est appelée dans le fichier FormSaisieMenagerie.cs

Création des objets correspondant aux composants (classe de l'objet = type du composant)

Les propriétés du composant lblNom de type etiquette (instance de la classe Label) sont renseignées

Les propriétés du composant txtNom de type champ de saisie (instance de la classe TextBox) sont renseignées

La liste déroulante contenant les espèces

Le bouton permettant de créer un animal

IV. GERÉR DES ÉVÉNEMENTS

1) La programmation événementielle

La programmation événementielle est un type de programmation fondée sur les événements. Elle s'oppose à la programmation séquentielle. Le programme sera principalement défini par ses réactions aux différents événements qui peuvent se produire. *[d'après Wikipedia]*

La programmation événementielle peut être réalisée dans n'importe quel langage de programmation, bien que la tâche soit plus aisée dans les langages de haut niveau (comme Java ou C#). Certains environnements de développement intégré (dont Visual Studio) permettent de générer automatiquement le code des tâches récurrentes dans la gestion des événements.

Il existe différents événements :

- Un clic de souris, le double-clic, le clic gauche,
- Une frappe de touche,
- La fermeture, le déplacement, le redimensionnement d'une fenêtre
- La sélection d'un menu ou d'une option dans une liste, etc…

En programmation événementielle, il est possible de connaître :

- l'événement qui a été déclenché (clic, frappe, sélection),
- Mais aussi l'objet graphique qui a provoqué ou « envoyé » l'événement (bouton, etc …)
- Mais encore des précisions éventuelles concernant l'événement (ex : les coordonnées précises du curseur au moment du clic, etc…)

  Visual Studio va nous aider à écrire le traitement à effectuer lors d'un événement

2) Présentation des classes Animal et Menagerie

Vous allez utiliser la bibliothèque de classes LibMenagerie (dll fournie).

Vous n'avez rien à coder, vous allez juste utiliser les classes que j'ai écrites pour vous.

Cette bibliothèque de classe contient deux classes :

- La classe Animal qui contient les caractéristiques d'un animal et ses comportements
- La classe Menagerie qui contient une liste d'objets de la classe Animal

Voici le contenu de ces deux classes :

- Contenu de la classe Animal :

```
/// <summary>
/// Représente un animal
/// </summary>
public class Animal
{
    // attributs privés
    private string nom;
    private string espece;
    private string cri;

    // Propriétés
    /// <summary>
    /// Obtient ou définit le nom d'un animal
    /// </summary>
    public string Nom
    {
        get { return nom; }
        set { nom = value; }
    }
    /// <summary>
    /// Obtient ou définit l'espèce d'un animal
    /// </summary>
    public string Espece
    {
        get { return espece; }
        set { espece = value; }
    }
    /// <summary>
    /// Obtient ou définit le cri d'un animal
    /// </summary>
    public string Cri
    {
        get { return cri; }
        set { cri = value; }
    }
    // Constructeur
    /// <summary>
    /// Initialise une nouvelle instance de la classe libAnimalerie.Animal
    /// </summary>
    /// <param name="sonNom">Nom de l'animal (Ex : "Garfield")</param>
    /// <param name="sonEspece">Espèce de l'animal (Ex : "Chat")</param>
    /// <param name="sonCri">Cri de l'animal (Ex : "Ouaf")</param>
    public Animal(string sonNom, string sonEspece,string sonCri)
    {
        nom = sonNom;
        espece = sonEspece;
        cri = sonCri;
    }
}
```

- Contenu de la classe Menagerie :

```
/// <summary>
/// Représente une ménagerie, laquelle peut "héberger" un certain nombre d'animaux
/// </summary>
public class Menagerie
{
    // Propriétés
    private List<Animal> listeAnimaux;

    // Constructeur
    /// <summary>
    /// Initialise une nouvelle instance de la classe libAnimalerie.Menagerie
    /// </summary>
    public Menagerie()
    {
        listeAnimaux = new List<Animal>();
    }
    // Méthodes

    /// <summary>
    /// Obtient le nombre d'animaux présents dans la ménagerie
    /// </summary>
    public int NombreAnimaux
    {
        get { return listeAnimaux.Count; }

    }
    /// <summary>
    /// Enregistre l'arrivée d'un nouvel animal dans la ménagerie
    /// </summary>
    /// <param name="unAnimal">L'animal (l'objet de type Animal) à ajouter à la ménagerie</param>
    public void Arriver(Animal unAnimal)
    {
        listeAnimaux.Add(unAnimal);
    }
    /// <summary>
    /// Enregistre le départ d'un animal de la ménagerie
    /// </summary>
    /// <param name="unAnimal">l'animal (l'objet de type Animal) à supprimer</param>
    public void Partir(Animal unAnimal)
    {
        listeAnimaux.Remove(unAnimal);
    }
    /// <summary>
    /// Indique si un animal existe dans la ménagerie : la méthode retourne True si l'animal existe, False sinon
    /// </summary>
    /// <param name="unAnimal">animal dont on souhaite savoir s'il existe ou non</param>
    public bool Exister(Animal unAnimal)
    {
        return listeAnimaux.Contains(unAnimal);
    }
    /// <summary>
    /// Retourne la collection d'animaux
    /// </summary>
    /// <returns>collection de type List</returns>
    public List<Animal> ListeAnimaux()
    {
        return listeAnimaux;
    }
}
```

A vous !!!

- Récupérez le répertoire nommé *dll* qui contient la dll qui vous permettra de travailler avec les classes **Animal** et **Menagerie**
- Ajoutez dans votre projet Application Windows Forms une référence à la bibliothèque LibMenagerie et n'oubliez pas le using

3) Réalisation de méthodes événementielles

Gérer l'événement clic sur le bouton "Créer la ménagerie"

- Pour pouvoir saisir les instructions qui seront éxécutées lors du clic sur le bouton "Créer la ménagerie", effectuez un double clic sur ce bouton

- Une procédure est affichée : cette procédure sera appelée lorsque l'événement « clic sur le bouton » se produira : on dit que cette procédure est une **procédure événementielle.**

Cette procédure événementielle est une **méthode d'instance** du formulaire.

```
namespace GestionMenagerie
{
    public partial class FormSaisieMenagerie : Form
    {
        // déclaration de la ménagerie
        Menagerie maMenagerie;

        public FormSaisieMenagerie()
        {
            InitializeComponent();
        }

        private void btnCreerMenagerie_Click(object sender, EventArgs e)
        {

        }

    }
}
```

btnCreerMenagerie_Click : nom de la procédure événementielle, par défaut le nom est composé du nom du contrôle suivi de l'événement, séparés par un « _ »

sender : objet (contrôle) qui a déclenché l'événement

e : informations complémentaires sur l'événement déclenché

- Vous allez maintenant saisir dans la méthode btnCreerMenagerie les instructions permettant de créer un objet, instance de la classe Menagerie

A vous !!!

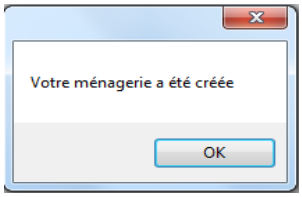
Dans cette méthode, faites le nécessaire pour créer un objet que vous appellerez maMenagerie, cet objet sera une instance de la classe Menagerie

Afficher une boîte de dialogue avec Message.Show

Après l'instruction de création de l'objet maMenagerie vous allez ajouter l'instruction suivante :

```
MessageBox.Show("Votre ménagerie a été créée");
```

Cette instruction permettra à l'utilisateur d'obtenir le message suivant :



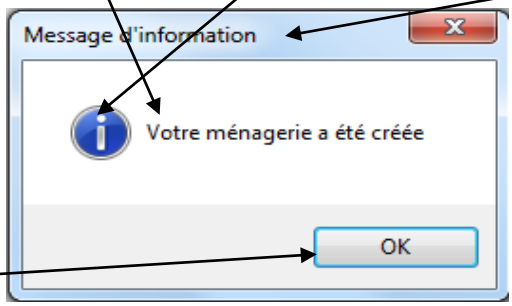
Vous utiliserez souvent cette commande MessageBox.Show(). Il est possible d'afficher une boîte de dialogue plus explicite en passant d'autres paramètres à la méthode Show comme ceci :

```
MessageBox.Show("Votre ménagerie a été créée", "Message d'information",
MessageBoxButtons.OK, MessageBoxIcon.Information);
```

Libellé complet du message.

Icône correspondant à un message d'information

Bouton(s) présents dans la boîte de dialogue : ici un seul, mais il y a d'autres possibilités, par exemple YesNo (Oui/Non), YesNoCancel pour (Oui/Non/Annuler) qui seront utilisés lors d'un message de confirmation...



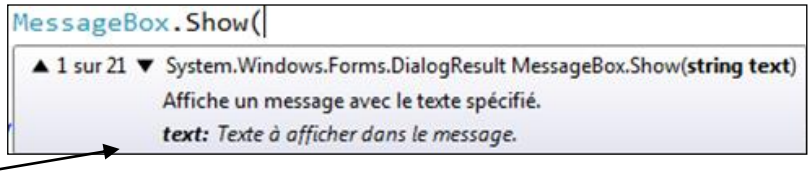
Titre de la boîte de message : c'est un libellé libre. Pour notre application, nous aurons :

- des messages d'information
- des messages d'erreur
- des messages de confirmation

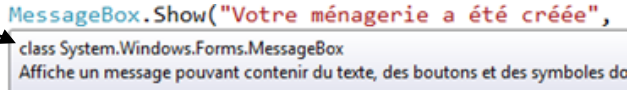
La méthode Show : une méthode un peu particulière

Après avoir saisi **MessageBox.Show(** vous obtenez :

La méthode Show possède **21 surcharges : il y a donc 21 manières d'appeler la méthode !!**



MessageBox est une Classe et non un objet.



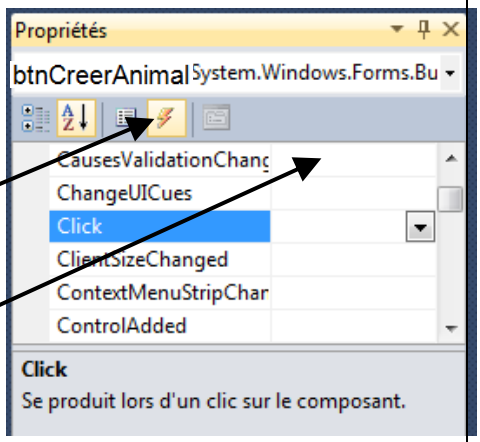
La méthode Show est nommée **méthode de classe** (ou encore **méthode à portée classe**) car pour l'appeler on indique le nom d'une classe devant le nom de la méthode.

Il n'est donc pas nécessaire de créer un objet pour appeler la méthode !

Mais vous connaissez déjà des méthodes de classe…WriteLine() et ReadLine() sont des méthodes à portée classe de la classe Console !

Gérer l'événement clic sur le bouton "Ajouter l'animal dans la ménagerie"

Vous allez ajouter le code sur clic du bouton «Ajouter l'animal … " **en utilisant un autre chemin** :



- Accédez aux propriétés du bouton « btnCreerAnimal »
- Accédez à la liste des événements associés à ce contrôle en cliquant sur l'éclair jaune
- Double-cliquez ensuite à droite de l'événement « Click », dans la case blanche permettant de choisir ou de définir une procédure événementielle.

Visual Studio génère une méthode ayant pour nom : nom du composant_nom de l'évènement : ici ce sera : **btnCreerAnimal_Click**

Vous auriez pu ne pas double-cliquer mais saisir votre propre nom de méthode en face de l'événement et double cliquez sur le nom saisi. Une petite recommandation : même si Visual Studio vous permet de choisir le nom que vous souhaitez, utilisez plutôt les noms proposés par Visual Studio car ils suivent une convention de nommage !!!

Ajouter dans la méthode **btnCreerAnimal_Click** le code suivant qui va vous permettre de creer un animal :

```
private void btnCreerAnimal_Click(object sender, EventArgs e)
{
    Animal unAnimal = new Animal(txtNom.Text,lstEspece.Text, txtCri.Text);
}
```

Si vous voulez déclarer une référence à un objet de la classe Animal et la création de l'objet en 2 instructions, vous pouvez bien entendu le faire sans problème…

Quelques explications sur la manière de récupérer une valeur saisie ou sélectionnée dans un formulaire…

Le constructeur de Animal a besoin d'informations pour renseigner les variables contenues dans l'objet, instance de la classe Animal : son nom, son espèce, son cri

Nous devons lui fournir les valeurs saisies ou sélectionnées dans le formulaire :

- la valeur saisie dans un TextBox est enregistrée dans la propriété Text

ex : txtNom.Text contiendra Nemo si l'utilisateur a saisi Nemo dans le nom

txtNom est un objet de la classe TextBox

Text est une propriété de la classe TextBox

- la valeur **sélectionnée** dans une ListBox est également enregistrée dansText

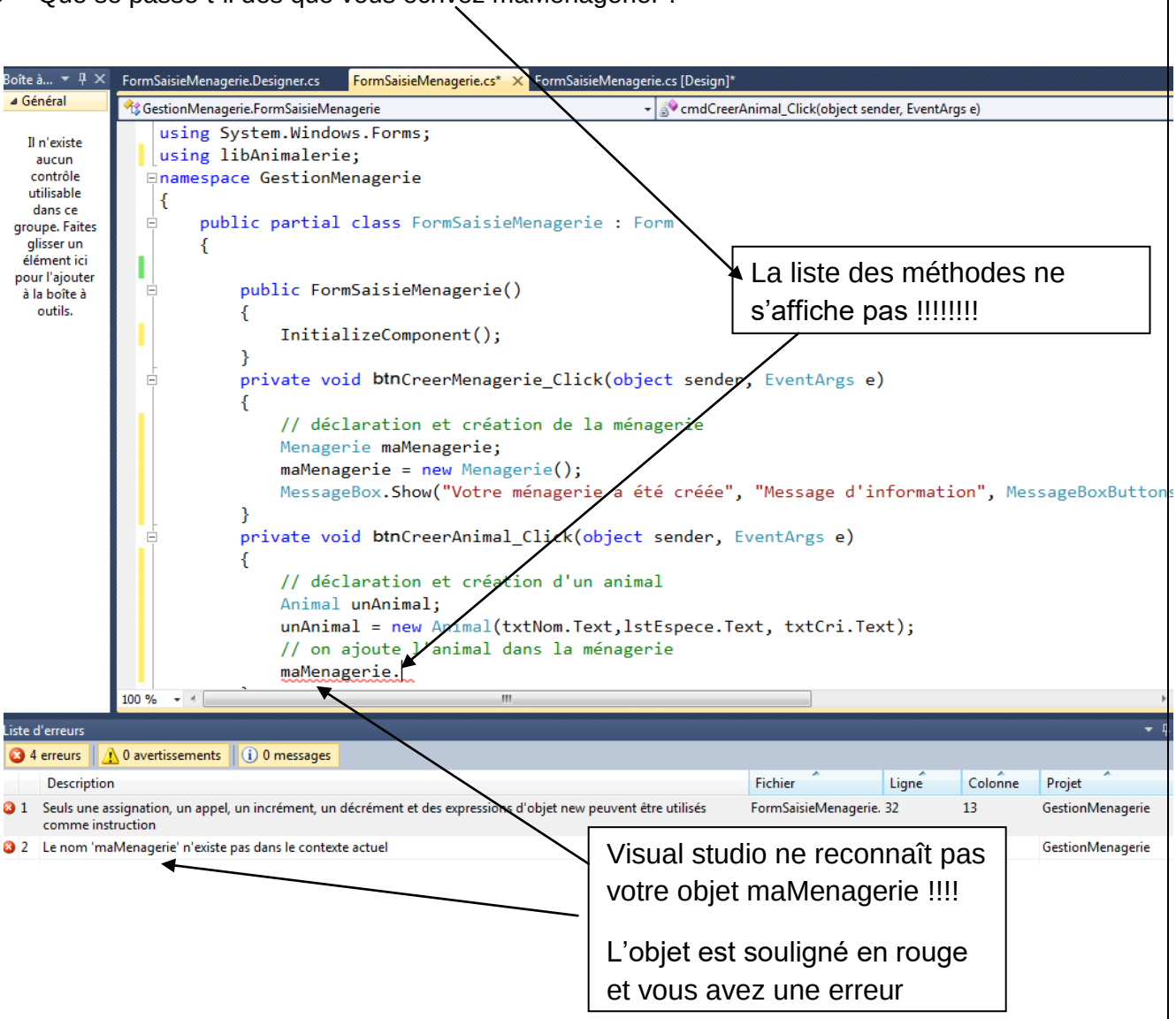
Vous allez compléter la méthode btnCreerAnimal_Click pour ajouter l'animal que vous venez de créer dans votre ménagerie.

Souvenez-vous : pour ajouter un animal dans une ménagerie, il est nécessaire de passer l'animal à ajouter (votre animal est représenté par l'objet unAnimal) à la méthode Arriver(..) de votre ménagerie représentée par l'objet maMenagerie.

Voici l'instruction que vous devez saisir :

```
maMenagerie.Arriver(unAnimal);
```

💣 Que se passe-t-il dès que vous écrivez maMenagerie. ?

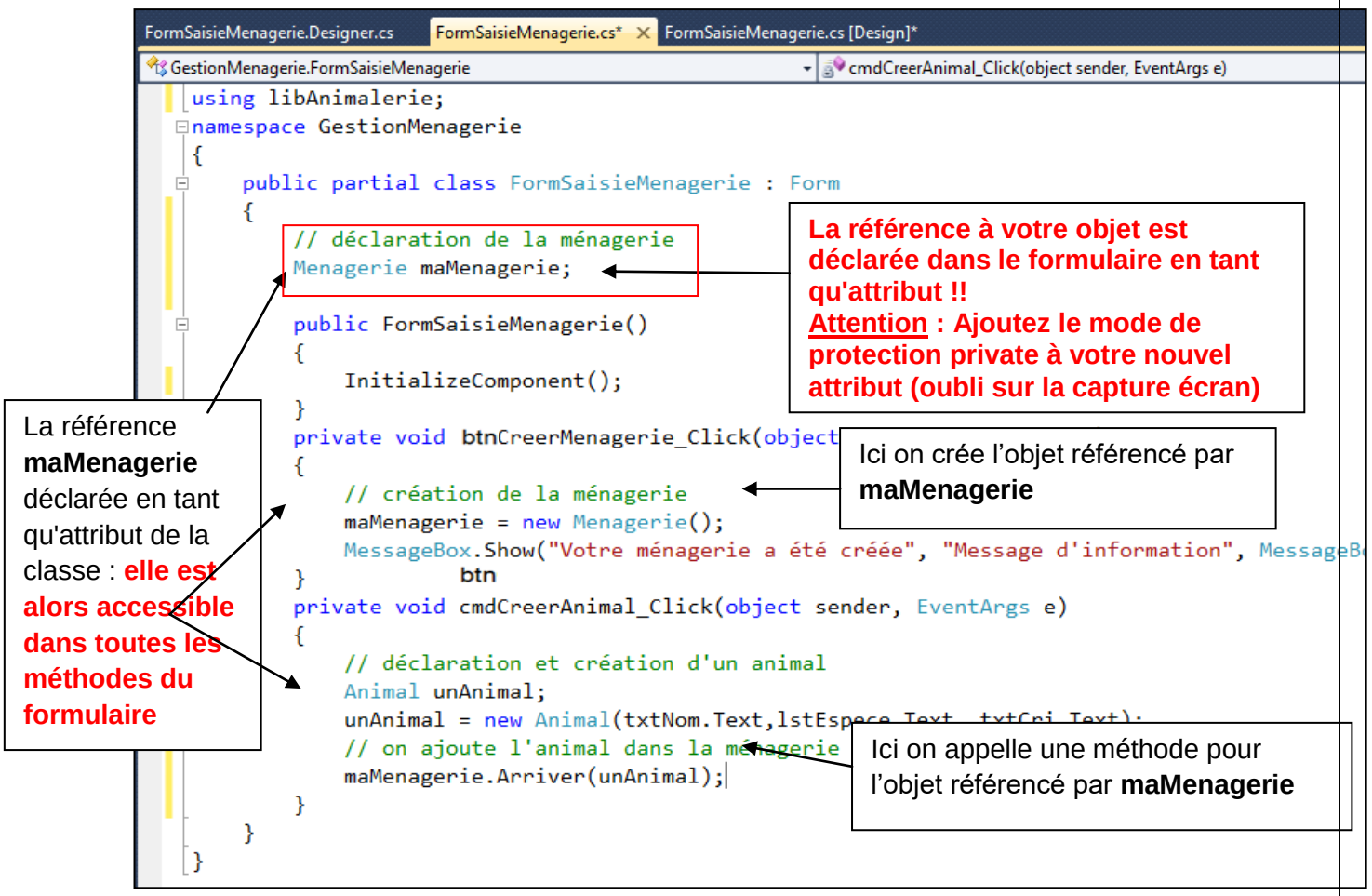


CECI EST TOUT A FAIT NORMAL !!! **Souvenez vous ce que je vous ai déjà dit : une méthode ne peut pas travailler avec des les variables déclarées dans d'autres méthodes !**

maMenagerie est une variable déclarée dans la méthode *btnCreerMenagerie* donc la méthode *btnCreerAnimal* ne peut pas y accéder !

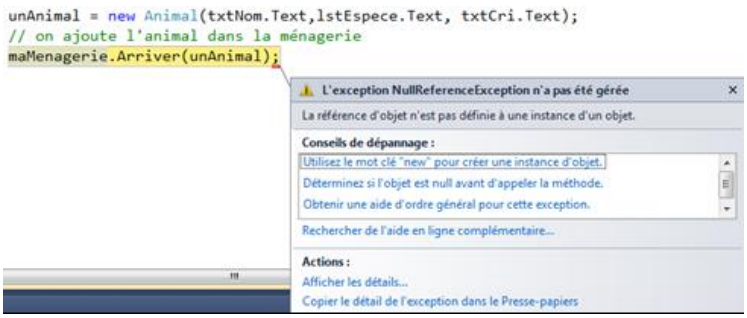
Si vous envisagez de modifier une procédure pour y ajouter un paramètre, oubliez vite cette idée car Visual Studio ne vous le permet pas !! alors que faire ????

- Vous allez déclarer la variable ***maMenagerie*** (votre référence à votre ménagerie) en tant qu'attribut de votre formulaire : de cet fait ***maMenagerie*** sera donc connue et accessible dans toutes les méthodes du formulaire :

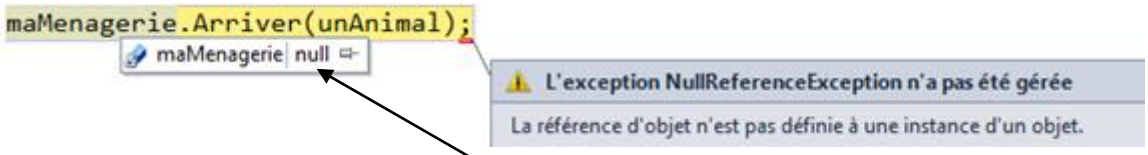


- **A vous !!!**
  - **LANCEZ votre application et faites EXACTEMENT ce qui suit :**
    - **1 - saisissez Nemo dans le nom,**
    - **2 - ne sélectionnez pas de valeur dans la liste**
    - **3 - saisissez Gloup dans le champ cri**
    - **4 - appuyez sur le bouton "Ajouter l'animal dans la ménagerie"**

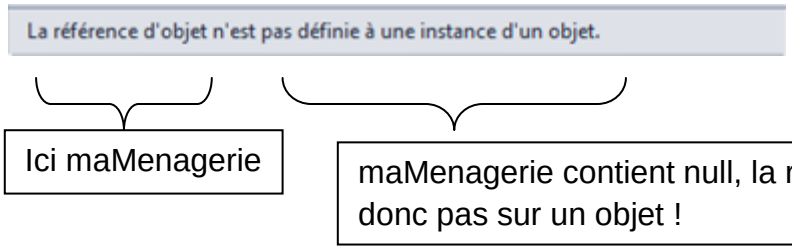
Si vous avez suivi ce que je vous demande, votre application doit génèrer l'exception suivante :



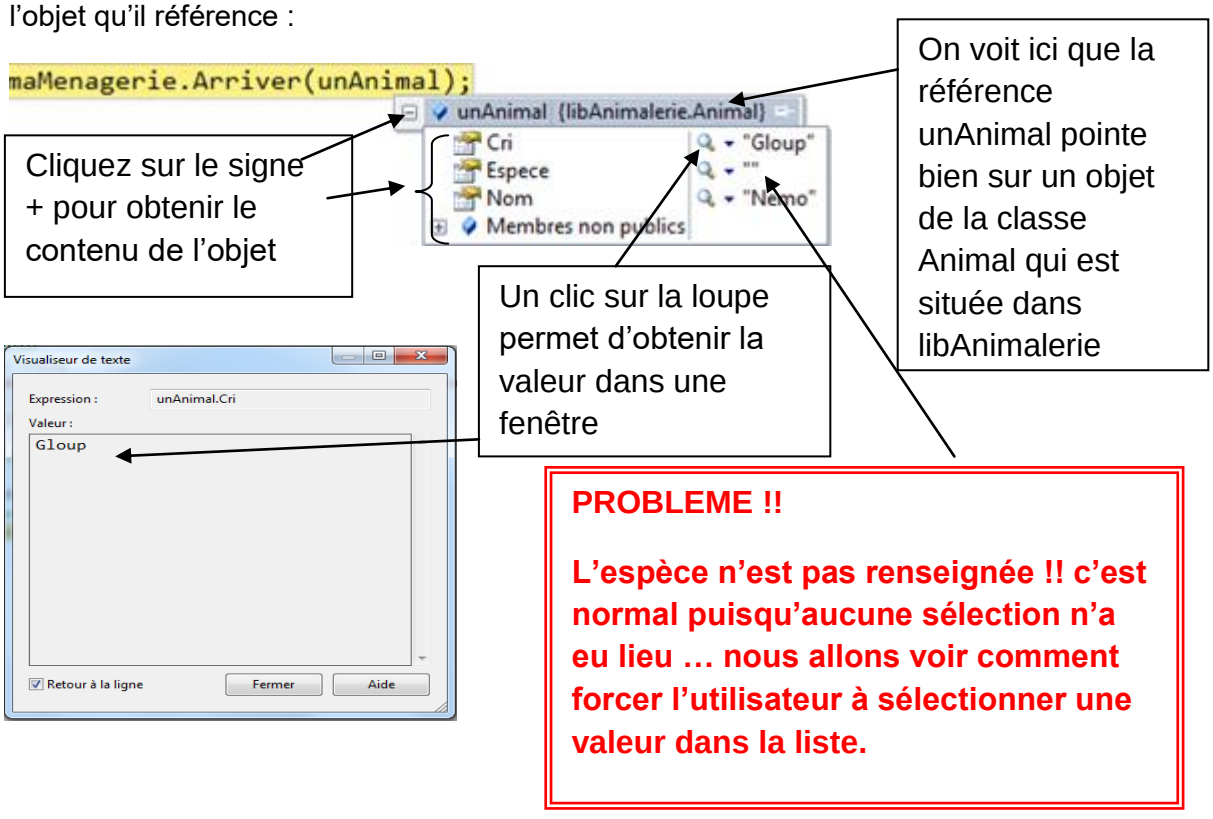
Pour comprendre ce qu'il s'est passé positionnez le curseur sur maMenagerie (amenez le curseur sur le nom de la référence sans cliquer !) afin de consulter le contenu de la référence



Le problème vient du fait que la référence contient la valeur null, ce qui signifie que votre référence maMenagerie ne « pointe » pas sur un objet ! d'ailleurs c'est ce que vous dit également le message de l'exception (prenez le temps de lire les messages d'erreur !!) :



Avant de voir comment résoudre ce problème, nous allons voir que notre application a besoin d'un petit réglage ! Positionnez votre curseur sur unAnimal pour voir le contenu de l'objet qu'il référence :



PROBLEME !!

L'espèce n'est pas renseignée !! c'est normal puisqu'aucune sélection n'a eu lieu … nous allons voir comment forcer l'utilisateur à sélectionner une valeur dans la liste.

V. EN COULISSE : LE CODE GENERE POUR GERER UN EVENEMENT

Les événements

Lors de l'ajout de l'évènement clic sur le bouton btnCreerAnimal, l'instruction suivante a été ajoutée automatiquement dans la méthode **InitializeComponent**:

```
this.btnCreerAnimal.Click += new System.EventHandler(this.btnCreerAnimal_Click);
```

Ajout d'un gestionnaire d'événement

Cette instruction permet d'indiquer que lors de l'évènement de type click sur le bouton btnCreerAnimal alors la méthode btnCreerAnimal_Click sera appelée.

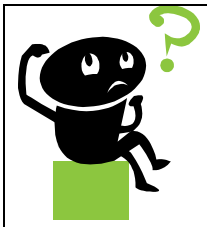
Les manipulations faites en conception pour gérer des événements comme un clic de bouton ou un changement de sélection dans une liste produisent des lignes de code. Visual Studio nous rend un fier service, car la syntaxe n'est pas toujours évidente.

On rappelle que cette procédure événementielle, écrite par vos soins dans le fichier « **FormSaisieMenagerie.cs »** admet deux paramètres, donc le contrôle qui a déclenché l'événement :

sender : Objet qui a déclenché l'événement.

e : Informations sur l'événement

```
private void btnCreerMenagerie_Click(object sender, EventArgs e)
{
    |
}
```



A RETENIR

- Les interfaces graphiques reposent sur les principes de la programmation événementielle.
- Un événement peut être déclenché par l'utilisateur (clic de souris, sélection …), mais aussi par le système (chargement ou fermeture d'une feuille, top d'horloge …).
- Le système capte ces différents événements et les traite dans leur ordre d'arrivée. En programmation événementielle, le programmeur prévoit le traitement à effectuer lorsque l'événement se produit.
- Les instructions à exécuter sont regroupées dans une procédure événementielle (ou gestionnaire d'événement) que l'on associe au composant concerné par l'événement.

VI. CONTROLER LES SAISIES SUR UN FORMULAIRE

Comment vérifier que l'objet maMenagerie existe ?

C'est simple, il suffit de tester que la référence maMenagerie **ne contient pas null avant** d'appeler la methode Arriver !!

Après ce réglage vous n'aurez plus l'erreur précédente !

Comment faire pour vérifier qu'une valeur a bien été sélectionnée dans une liste déroulante ?

- La propriété **SelectedIndex** d'une ListBox contient :
  - **-1** si aucun élément n'a été sélectionné
  - 0 si le 1er élément de la liste a été sélectionné
  - 1 si le 2ème élément de la liste a été sélectionné
  - 2 si le 3ème élément de la liste a été sélectionné

- Pour forcer l'utilisateur à sélectionner une valeur nous allons :
  - vérifier que la propriété **SelectedIndex** de notre liste ne contient pas -1
  - affecter -1 à la propriété **SelectedIndex** de notre liste après l'ajout de l'animal dans la ménagerie : de ce fait pour la saisie du prochain animal aucune ligne ne sera pré-sélectionnée dans la liste !

Voici ce que je vous propose :

```
private void btnCreerAnimal_Click(object sender, EventArgs e)
{
    // déclaration
    Animal unAnimal;
    if (maMenagerie != null)
    {
        if (lstEspece.SelectedIndex >-1)
        {
            // création d'un animal
            unAnimal = new Animal(txtNom.Text, lstEspece.Text, txtCri.Text);
            // on ajoute l'animal dans la ménagerie
            maMenagerie.Arriver(unAnimal);
            // on fait le nécessaire pour qu'aucune valeur
            // ne soit sélectionnée dans la liste
            lstEspece.SelectedIndex = -1;
        }
    }
}
```

On vérifie que la ménagerie a bien été créée

On vérifie qu'une valeur a bien été sélectionnée dans la liste des espèces

Après cette instruction, aucune ligne n'apparaît sélectionnée dans la liste des espèces

Cette procédure fonctionne bien : il n'y a plus d'exception ! … mais l'utilisateur ne sait pas si le traitement s'est bien passé … ni s'il y a des problèmes. Vous allez faire le nécessaire pour afficher des messages à l'utilisateur en utilisant **MessageBox.Show(…)**

A vous !!! Complétez la procédure ci-dessous pour afficher les messages suivants :

- **Affichez le message d'erreur "Veuillez créer votre ménagerie avant d'ajouter un animal ! " si l'objet maMenagerie n'a pas été créé**
- **Affichez le message d'erreur "Veuillez sélectionner une espèce pour l'animal" si l'utilisateur n'a pas sélectionné de valeur dans la liste des espèces**
- **Affichez le message d'information "L'animal a été ajouté dans la ménagerie" si tout s'est bien passé**

- **A vous !!! Testez votre formulaire**
  - **Lancez votre application**
  - **Testez votre application avec des jeux d'essai qui doivent provoquer des erreurs :**

    Jeu n°1 :

    - appuyez sur le bouton "Ajouter l'animal dans la ménagerie"

    Jeu n°2 :

    1 - appuyez sur "Créer la ménagerie"

    2 - saisissez Nemo dans le nom,

    2 - ne sélectionnez pas de valeur dans la liste

    3 - saisissez Gloup dans le champ cri

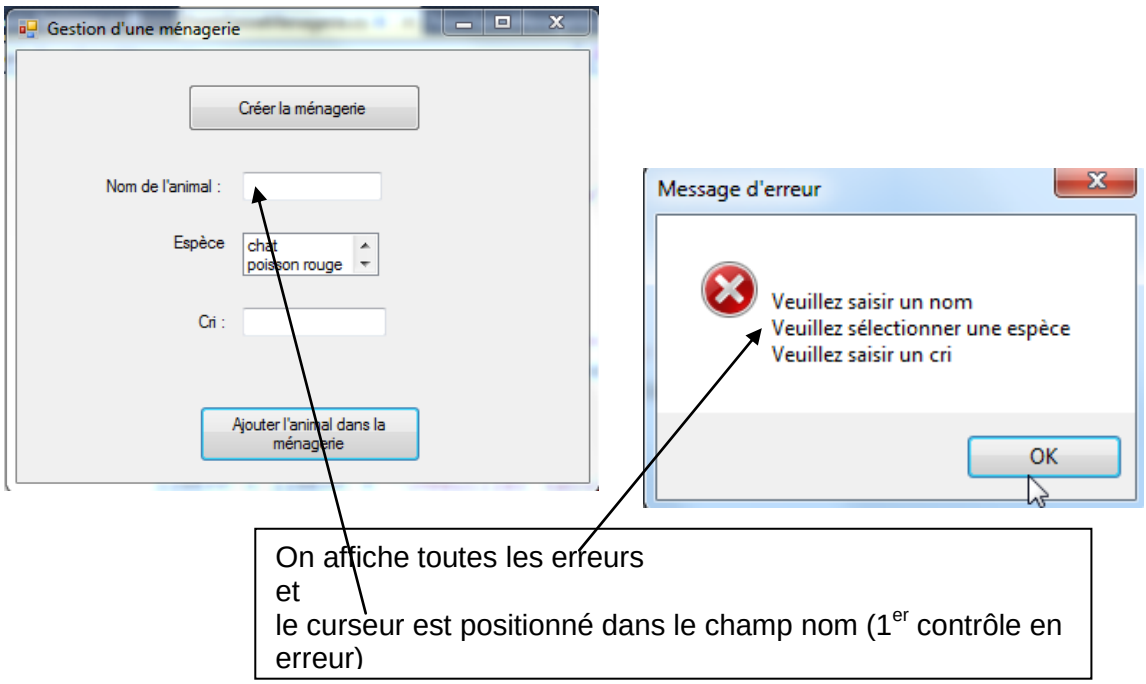
    4 - appuyez sur le bouton "Ajouter l'animal dans la ménagerie"
  - **Exécutez votre application et tester votre application avec des jeux d'essi qui ne provoquent pas d'erreur. Votre ménagerie doit contenir au moins 3 animaux (vous pouvez créer les mêmes animaux que ceux vous avez créés lors du TP précédent)**

Votre application n'est pas encore parfaite … l'utilisateur peut ne pas avoir saisi une valeur pour le nom et/ou pas de valeur pour le cri !! et pour l'instant un animal pourra être créé sans nom, sans cri, avec juste son espèce de renseignée.

Comment faire pour vérifier que tous les contrôles d'un formulaire sont bien renseignés ?

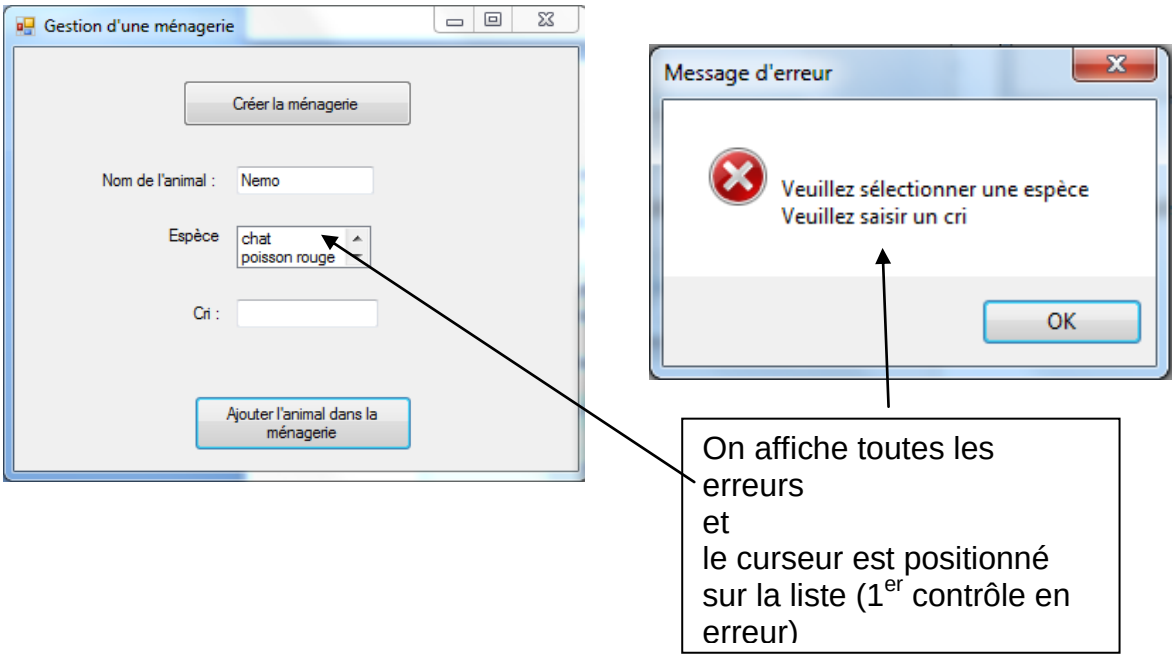
Vous allez voir comment contrôler que toutes les valeurs sont bien saisies ; si ce n'est pas le cas vous afficherez toutes les erreurs et vous positionnerez le curseur sur le premier champ posant problème.

Voici par exemple ce que l'utilisateur obtient s'il n'a rien sélectionné, ni rien saisi :



On affiche toutes les erreurs
et
le curseur est positionné dans le champ nom (1er contrôle en erreur)

Voici ce que l'utilisateur obtient s'il a saisi un nom mais rien sélectionné dans la liste, ni rien saisi dans le cri :



Voici comment faire :

```
if (maMenagerie != null)
{
    string libErr="";
    if (txtNom.Text.Trim() == "")
    {
        libErr = "\nVeuillez saisir un nom";
        txtNom.Focus();
    }
    if (lstEspece.SelectedIndex == -1)
    {
        libErr = libErr + "\nVeuillez sélectionner une espèce";
        if (txtNom.Focused == false) lstEspece.Focus();
    }
    if (txtCri.Text.Trim() == "")
    {
        libErr = libErr + "\nVeuillez saisir un cri";
        if (txtNom.Focused == false && lstEspece.Focused==false) txtCri.Focus();
    }
    if (libErr == "")
    {
        // création d'un animal
        unAnimal = new Animal(txtNom.Text, lstEspece.Text, txtCri.Text);
        // on ajoute l'animal dans la ménagerie
        maMenagerie.Arriver(unAnimal);
        // on fait le nécessaire pour qu'aucune valeur
        // ne soit sélectionnée dans la liste
        lstEspece.SelectedIndex = -1;
        // message d'information
        MessageBox.Show("L'animal a été ajouté dans la ménagerie", "Message d'information", MessageBoxButtons.OK, Mess
    }
    else
    {
        // message d'erreur listant les contrôles du formulaire en erreur
        MessageBox.Show(libErr, "Message d'erreur", MessageBoxButtons.OK, MessageBoxIcon.Stop);
    }
}
```

Déclarez une variable qui va contenir toutes les erreurs. Initialisez-la à ""

On teste le contenu de chaque champ (essayez de comprendre l'intérêt de l'appel de la méthode Trim)En cas d'erreur, on met un libellé d'erreur
On positionne le curseur sur le champ

On ajoute (+) un libellé d'erreur dans la variable qui va recevoir toutes les erreurs

S'il n'y a pas d'erreur, on crée et on ajoute l'animal

S'il y a une erreur, on affiche le libellé contenant toutes les erreurs

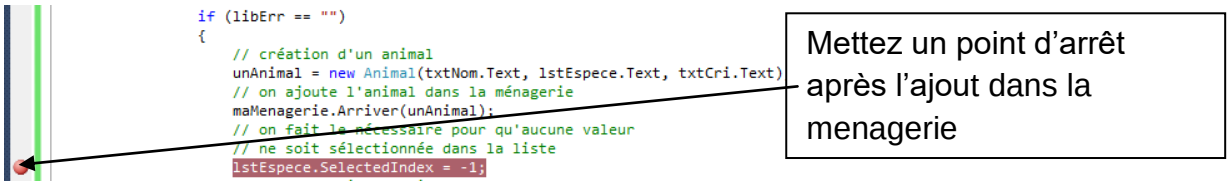
A retenir :

- La méthode Focus() permet de positionner le curseur sur un contrôle
  - Exemple : l'instruction ***txtCri.Focus()*** va positionner le curseur sur le contrôle txtCri

- La méthode Focused() permet de savoir si le curseur est positionné sur un contrôle
  - Exemple : l'instruction ***txtNom.Focused()*** retournera :
    - true si le curseur est positionné sur le contrôle txtNom
    - false sinon

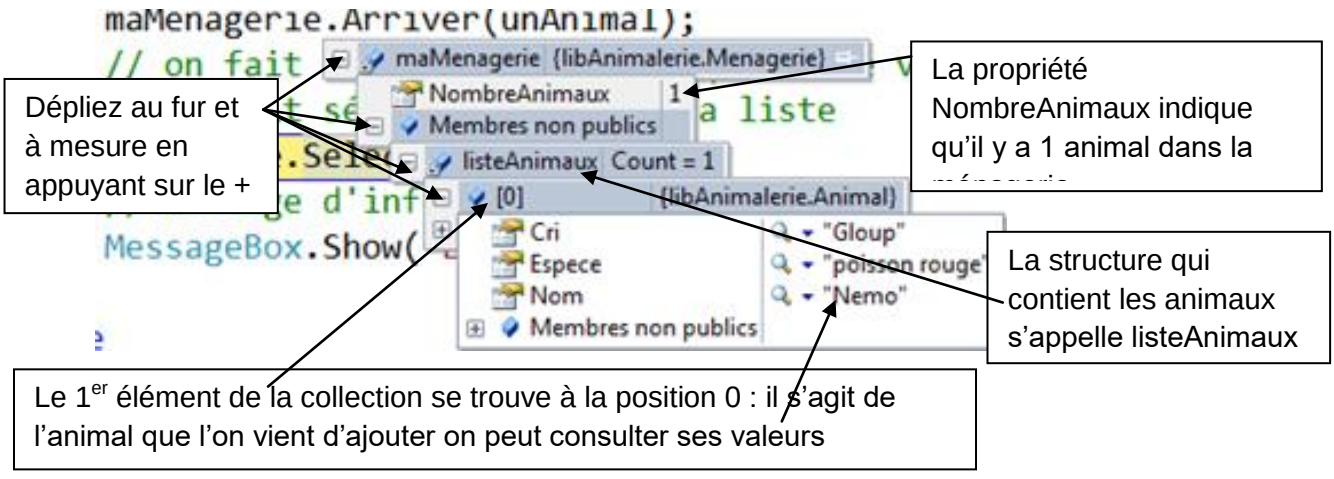
- L'astuce : l'appel de la méthode Focused() dans la procédure ci-dessus permet de rester positionné sur le premier contrôle qui est en erreur !!

Votre application fonctionne parfaitement mais comment voir si les animaux sont bien ajoutés dans votre ménagerie ?? Nous allons vérifier l'ajout en mettant **un point d'arrêt** dans le programme pour regarder le contenu de la ménagerie au fur et à mesure de l'ajout d'animaux :

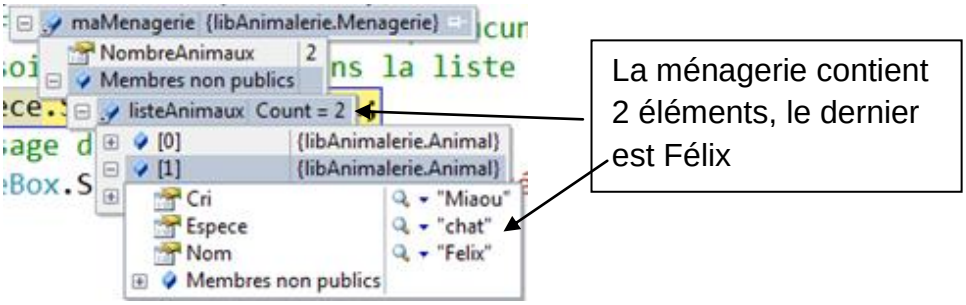


Lancez votre application (en mode debogage) et saisissez les caractéristiques d'un animal (Némo)

Lorsque le programme s'arrête sur l'instruction où vous avez mis le point d'arrêt consultez le contenu de votre ménagerie en positionnant votre curseur (sans cliquer !) sur maMenagerie:

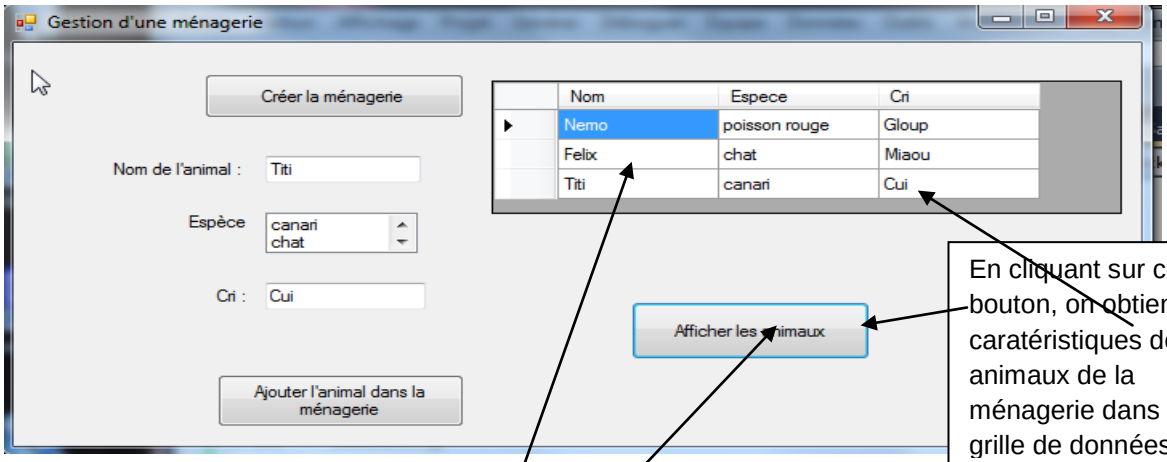


- Appuyez sur la touche F5 pour continuez et saisissez votre deuxième animal (Felix qui est un chat et qui dit Miaou) et consulter de nouveau votre ménagerie !



VII. AFFICHER DES DONNEES

Vous allez faire le nécessaire pour afficher les caractéristiques de tous les animaux de la ménagerie, comme ceci :



- Ajoutez un composant de type datagridView (**dtgAnimaux**) sur votre formulaire : appelez-le
- Ajoutez un bouton (btnAfficherMenagerie) permettant d'afficher les animaux de la ménagerie
- Double-cliquez sur le bouton btnAfficherMenagerie pour coder la procédure évènementielle qui sera exécutée lors du clic sur le bouton

```
private void btnAfficherMenagerie_Click(object sender, EventArgs e)
{
    if (maMenagerie != null)
    {

        if (maMenagerie.NombreAnimaux > 0)
        {
            dtgAnimaux.DataSource = null;
            dtgAnimaux.DataSource = maMenagerie.ListeAnimaux();
            dtgAnimaux.Visible = true;
        }
        else MessageBox.Show("Il n'existe pas d'animal dans la ménagerie !", "Message d'erreur", MessageBoxButtons.OK, MessageBoxIcon.Stop);
    }
    else MessageBox.Show("Consultation impossible : il n'existe pas de ménagerie !", "Message d'erreur", MessageBoxButtons.OK, MessageBoxIcon.Stop);
}
```

L'affichage n'a lieu que s'il existe une ménagerie

L'affichage n'a lieu que s'il existe des animaux dans la ménagerie

On associe les données récupérées à la grille de données et on rend visible ces données

Message d'erreur s'il n'existe pas d'animal dans la ménagerie

Message d'erreur s'il n'existe pas de ménagerie

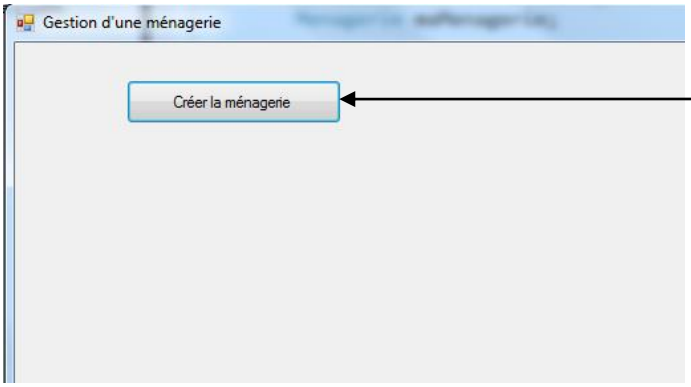
Quelques explications sur les instructions :

- **dtgAnimaux.DataSource = null** permet de ré-initialiser la liste des animaux de la ménagerie de la grille de données
- **maMenagerie.ListeAnimaux()** permet d'appeler une méthode qui retourne la liste des animaux de la ménagerie
- **dtgAnimaux.DataSource = maMenagerie.ListeAnimaux()** permet d'associer la liste des animaux de la ménagerie à la grille de données

VIII. ET SI VOUS DEVENIEZ AUTONOME ??

Vous allez faire le nécessaire pour améliorer votre application :

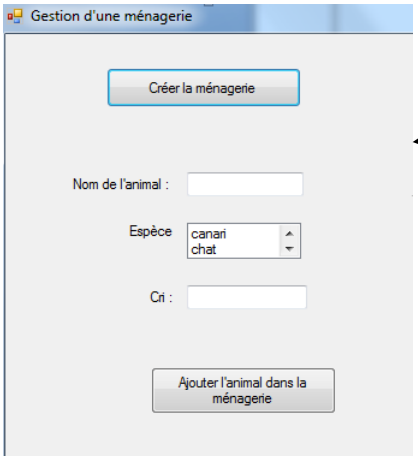
- On souhaite qu'au lancement de votre application seul le bouton "Créer la ménagerie" soit présent sur le formulaire :



Seul le bouton pour créer la ménagerie apparaît.

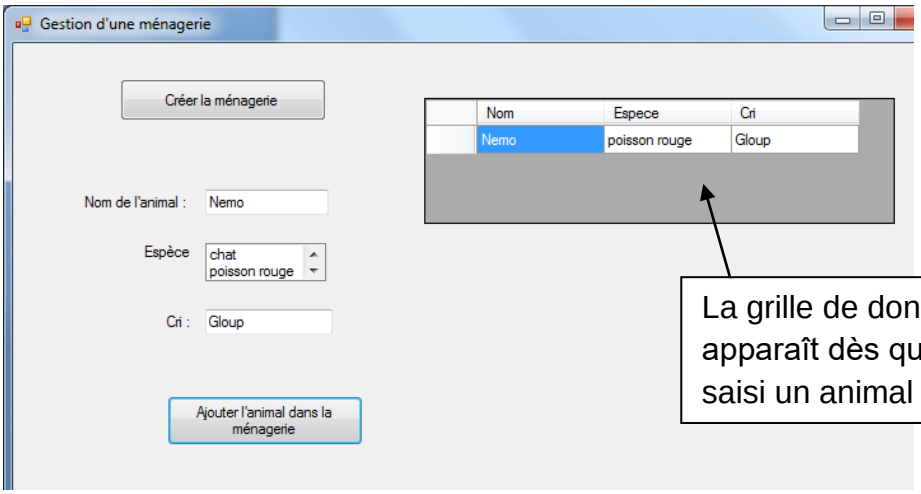
Les composants pour la saisie de l'animal n'apparaîssent pas et la grille de données n'est pas affichée

- Lorsque l'utilisateur clique sur le bouton "Créer la ménagerie", il obient :



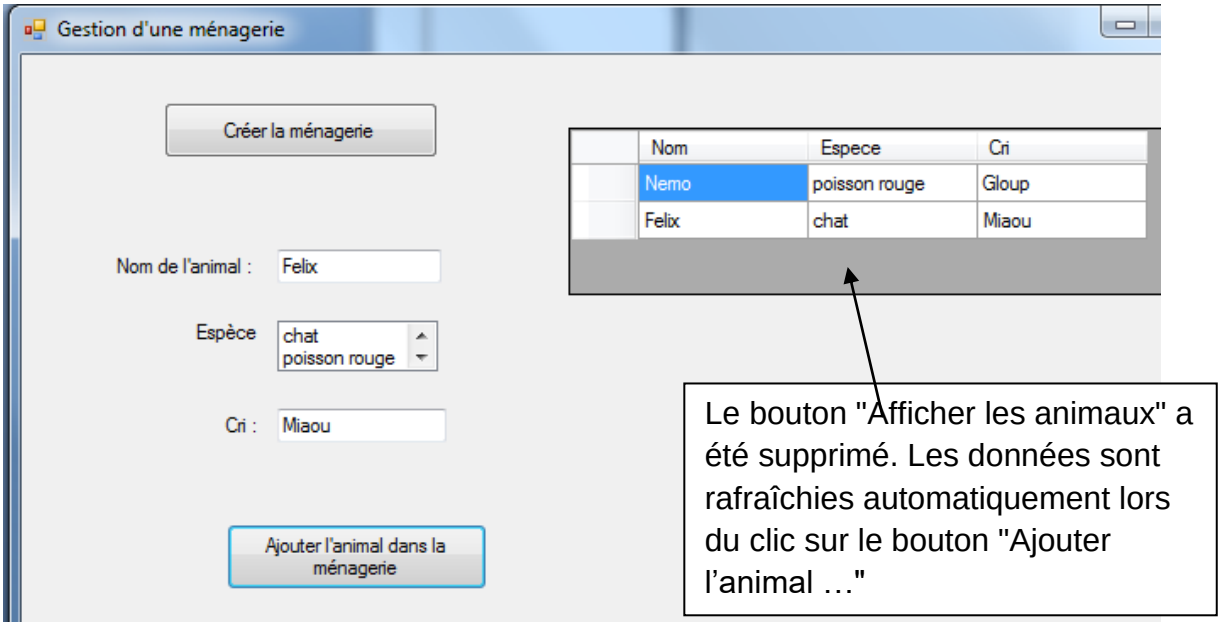
La grille de données n'apparaît pas

- Dès que l'utilisateur que la ménagerie contient au moins un animal, la grille de données s'affiche :



La grille de données apparaît dès que l'on a saisi un animal

- Puis dès que l'utilisateur ajoute un nouvel animal, la grille de données se réaffiche avec la liste des animaux :



:

- La propriété Visible :

  - pour rendre non visible un composant il faut positionner sa propriété Visible à False

  - pour rendre visible un composant il faut positionner sa propriété Visible à True

  - on peut positionner la propriété d'un composant en mode design (en passant par la fenêtre Propriétés d'un composant) ou en utilisant une instruction xxxx.Visible = False (ou True) où xxxx représente le nom du composant

- Les panels :

  - On peut regrouper plusieurs composants dans un conteneur (panel)
  - Un conteneur a des propriétés semblables aux composants (Name,Text …)
  - Si un panel n'est pas visible, ses composants ne le seront pas.