Fiche savoirs : Utilisation des objets graphiques

L'utilisation des objets graphiques dans le code va être présentée à travers un exemple de petite application.

1. Construire l'interface

L'application commence par la construction de l'interface graphique fixe, pour une application de bureau s'exécutant sur un ordinateur.

Lors de l'ajout de chaque objet graphique, n'oubliez pas de préciser son nom (inutile pour les labels informatifs).

Voici un exemple d'interface possédant les objets suivants :

- txtTitre : TextBox (en haut)
- lstTitres : ListBox (au centre)
- lblTitre : Label (en bas)
- btnAjouter : Button
- btnSupprimer : Button
- btnVider : Button

Le but de l'application est de saisir des titres qui vont s'ajouter dans la liste. Il est possible de supprimer un titre de la liste et de vider la liste. Lorsqu'un titre est ajouté ou sélectionné dans la liste, il apparait dans le label de rappel du bas de la fenêtre. De plus, les titres sont ajoutés en majuscule.

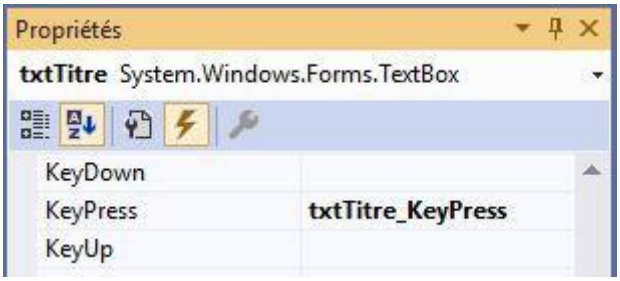
2. Gérer les événements

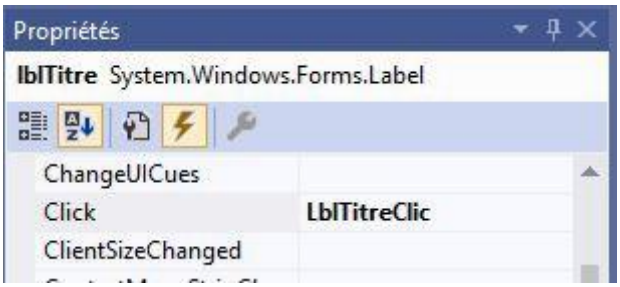
Une fois les objets graphiques placés et dans l'interface et configurés, il faut gérer les événements liés à ces objets.

2A. Créer une procédure événementielle

Voici les différentes méthodes pour créer automatiquement une procédure événementielle liée à un objet graphique.

Solution 1 : Double clic sur un objet graphique : la procédure événementielle de l'événement par défaut est créée dans le code. Exemple : double clic sur le bouton btnAjouter.	`private void btnAjouter_Click(object sender, EventArgs e) {` `}`

Solution 2 :
Sélection d'un objet graphique puis double clic sur l'événement voulu, dans la liste des événements : la procédure événementielle de l'événement sélectionné est créée dans le code.
Exemple : sélection de txtTitre et, dans la liste des événements, double clic sur KeyPress.
	
	`private void txtTitre_KeyPress(object sender, KeyPressEventArgs e) {`
	`}`

Solution 3 :
Sélection d'un objet graphique puis, dans la liste des événements, saisie d'un nom de procédure événementielle et validation : la procédure événementielle de l'événement sélectionné, avec le nom donné, est créée dans le code.
Exemple : sélection de lblTitre et, dans la liste des événements, à côté de l'événement Click, saisie de "LblTitreClic" et validation.
Remarque : cette dernière solution est à éviter, excepté si vous devez respecter un nommage des procédures événementielles imposé par une charte.
	
	`private void LblTitreClic(object sender, EventArgs e)`
	`{`
	`}`

2B. Comprendre le fonctionnement d'une procédure événementielle

Une procédure événementielle est liée à un événement. Lors de sa création, en plus de la procédure créée dans le code, celle-ci est aussi rattachée à un événement et à un objet, dans le code du "Designer".

Exemple :

Lors de la création de la procédure événementielle sur le clic du bouton btnAjouter, voici ce qui est ajouté dans le "Designer" :

```
this.btnAjouter.Click += new System.EventHandler(this.btnAjouter_Click);
```

Cela signifie que sur le clic du bouton btAjouter, une écoute (EventHandler) est mise en place et, lorsque l'événement se produit, la procédure, dont le nom est mis en paramètre, est exécutée.

2C. Supprimer proprement une procédure événementielle

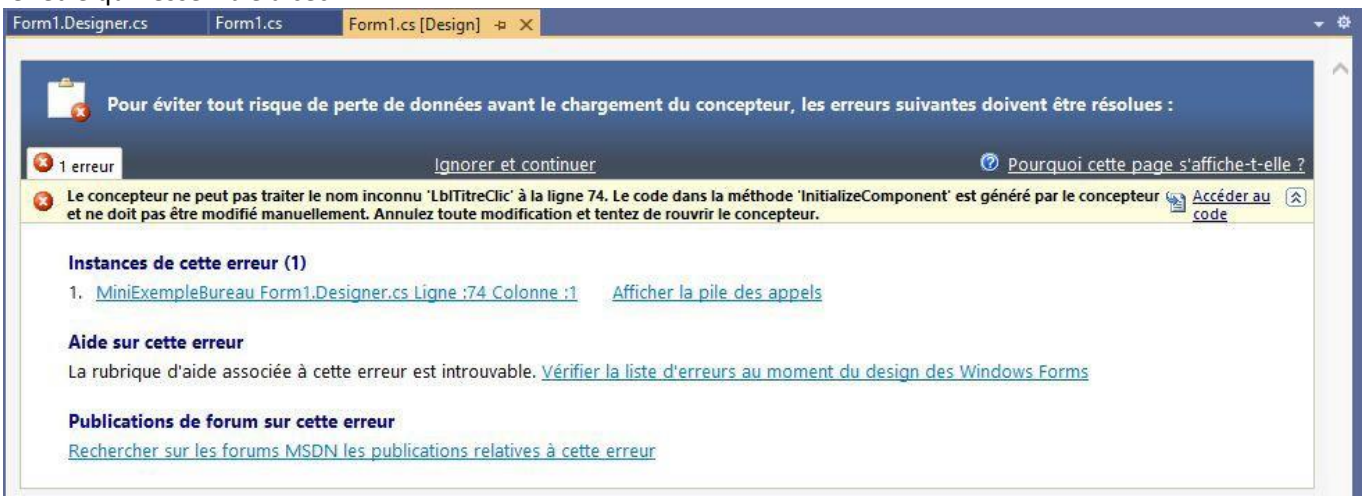
Pour supprimer une procédure événementielle, voici les étapes à suivre, dans cet ordre :

- Sans l'effacer totalement, videz le contenu de la procédure événementielle et enregistrer. - Allez dans la liste des événements et supprimez le nom de la procédure puis enregistrer.

Normalement la procédure a été effacée et elle n'apparait plus dans le code du "Designer".

2D. Gérer les erreurs si l'interface ne s'affiche plus

Il arrive qu'une erreur de manipulation soit faite, souvent justement en tentant de supprimer une procédure événementielle, et la fenêtre "Design" n'est plus accessible. Vous obtenez à la place, une fenêtre qui ressemble à ceci :



Cela signifie qu'il y a incompatibilité entre le contenu du code du "Designer" et le contenu du code de la fenêtre.

Rappelez-vous que ce sont deux "partial Class" donc en réalité c'est une même classe séparée en deux.

Cette erreur se produit par exemple si vous supprimez directement une procédure événementielle dans le code de la fenêtre au lieu de passer par la liste des événements.

Comment réparer l'erreur ?

C'est le seul cas où il faut intervenir dans le code du "Designer" AVEC PRUDENCE, car n'oubliez pas que ce code est généré automatiquement par rapport à l'interface.

Dans le code du "Designer, repérez la ligne qui comporte une erreur : normalement c'est la ligne correspondant à
< l'ajout de l'événement, sur l'objet concerné. Par
exemple :

```
this.lblTitre.Click += new System.EventHandler(this.LblTitreClic);
```

Supprimez cette ligne et enregistrez.

De retour dans la fenêtre, elle devrait à nouveau s'afficher.

3. Manipuler les objets graphiques dans le code

Vous allez découvrir la manipulation des objets graphiques dans le code, à travers la construction de cette petite application.

Commençons par gérer le code de chaque événement :

Chargement (Load) de frmFilms :

Effacer le label qui affiche le titre sélectionné.

Mettre le focus sur la zone de texte pour la saisie d'un titre.

```
private void frmFilms_Load(object sender, EventArgs e)
{
   lblTitre.Text =  "";
   txtTitre.Focus();
}
```

Clic (Click) sur btnAjouter :

Mettre en majuscule le titre saisi dans la zone de texte et le récupérer dans une variable locale.

Si le titre n'est pas vide, l'ajouter à la liste et vider la zone de texte.

Dans tous les cas, remettre le focus sur la zone de texte pour la prochaine saisie.

```
private void btnAjouter_Click(object sender, EventArgs e)
{
   string titre = txtTitre.Text.ToUpper();
   if (!titre.Equals(""))
   {
       lstTitres.Items.Add(titre);
       txtTitre.Text = "";
   }
   txtTitre.Focus();
}
```

Touche appuyée (KeyPress) sur txtTitre :

Si la touche est la touche "entrée" (validation) marquant la fin de saisie du titre, appeler la procédure événementielle btnAjouter.

```
private void txtTitre_KeyPress(object sender,
KeyPressEventArgs e)
{
   if(e.KeyChar == (char)Keys.Enter)
   {
      btnAjouter_Click(null, null);
   }
}
```

Clic (Click) sur btnSupprimer :

Si une ligne est sélectionnée dans la liste, la supprimer et vider le label, sinon afficher un message demandant de sélectionner une ligne. Dans tous les cas, remettre le focus sur la zone de texte pour la prochaine saisie.

```
private void btnSupprimer_Click(object sender, EventArgs
e)
{
   if(lstTitres.SelectedIndex != -1)
   {
      lstTitres.Items.RemoveAt(lstTitres.
SelectedIndex);
      lblTitre.Text = "";
   }
   else
   {
      MessageBox.Show("Sélectionnez une ligne");
   }
   txtTitre.Focus();
}
```

Clic (Click) sur btnVider :

Après demande de confirmation, vider la liste et vider le label.

Dans tous les cas, remettre le focus sur la zone de texte pour la prochaine saisie.

```
private void btnVider_Click(object sender, EventArgs e)
{
   if(MessageBox.Show("Vider la liste ?",
"Confirmation", MessageBoxButtons.OKCancel) ==
DialogResult.OK)
   {
      lstTitres.Items.Clear();
      lblTitre.Text = "";
   }
   txtTitre.Focus();
}
```

Changement de ligne (SelectedIndexChanged) sur lstTitre :

Si une ligne est sélectionnée (normalement oui), copier le titre correspondant dans le label.

Dans tous les cas, remettre le focus sur la zone de texte pour la prochaine saisie.

```
private void lstTitres_SelectedIndexChanged(object
sender, EventArgs e)
{
   if(lstTitres.SelectedIndex != -1)
   {
      lblTitre.Text = lstTitres.SelectedItem.
ToString();
   }
   txtTitre.Focus();
}
```

4. Limiter les accès aux objets graphiques

Il est possible de rendre un objet graphique inaccessible. Cela peut permettre d'éviter certains tests et rend l'interface plus parlante pour l'utilisateur : il voit qu'il ne peut pas accéder à l'objet.

Par exemple, on aimerait que le bouton qui permet de supprimer une ligne dans la liste, ne soit accessible que si une ligne est effectivement sélectionnée. Il faut donc repérer les événements qui aboutissent à l'absence de sélection de lignes dans la liste ou, au contraire, à la sélection d'une ligne.

Au chargement du formulaire : La liste étant vide, le bouton de suppression doit être inactif.	`btnSupprimer.Enabled = false;`
Après la suppression d'une ligne : Il n'y a plus de ligne sélectionnée donc le bouton de suppression doit être inactif.	`btnSupprimer.Enabled = false;`
Après avoir vidé la liste : Il n'y a plus de ligne sélectionnée donc le bouton de suppression doit être inactif.	`btnSupprimer.Enabled = false;`
Lors de la sélection d'une ligne : Une ligne étant sélectionnée, le bouton de suppression doit devenir actif.	`btnSupprimer.Enabled = true;`

5. Optimiser le code avec des modules classiques

Les optimisations possibles avec des procédures et fonctions classiques, sont tout à fait possible en programmation événementielle.

Vous pouvez créer des procédures pour éviter des répétitions de code, des fonctions pour créer des outils.

Ici, la reposition du focus sur la zone de texte se fait plusieurs fois. Créer une procédure événementielle juste pour une ligne de code ne parait pas logique. Pourtant, on peut imaginer d'autres lignes de code qui pourraient s'y ajouter. D'ailleurs ici, il est logique que la zone de texte soit aussi systématiquement vidée avant de repositionner le focus dessus.

Il est possible de créer automatiquement une procédure non événementielle à partir de ligne(s) de code sélectionnée(s).

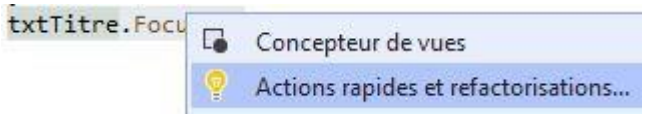
Sélectionner le code à isoler :

À n'importe quel endroit, sélectionnez la ligne de code correspondant à la position du focus sur la zone de texte.

```
txtTitre.Focus();
```

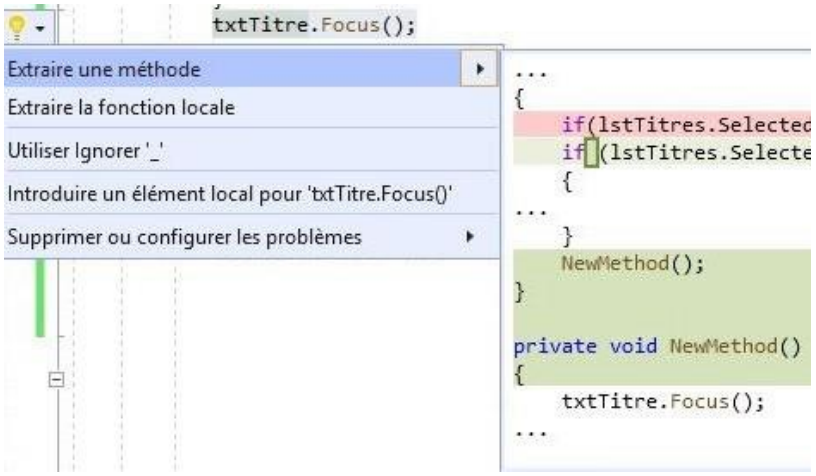
Demander de refactorisation :

Clic droit sur la sélection et, dans le menu qui s'affiche, sélectionnez "Actions rapides et refactorisations…"



Demander de création de la procédure :

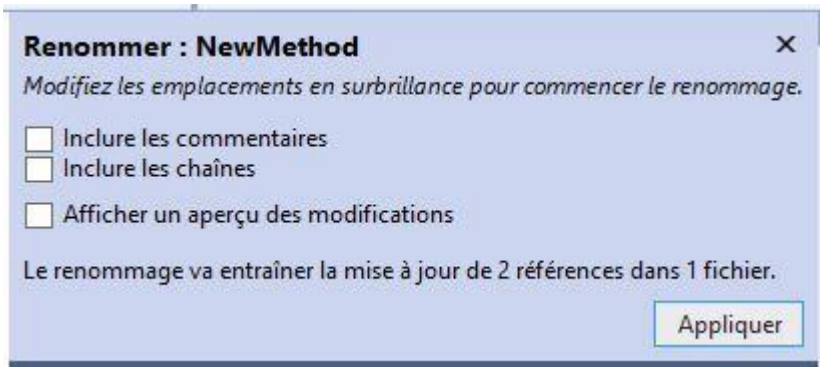
Dans le nouveau menu, sélectionnez "Extraire une méthode". L'aperçu de droite montre ce qui va être fait : Création d'une méthode "NewMethod" contenant le code sélectionné. Remplacement de la sélection par l'appel de la méthode.



Renommer la méthode :

Il vous est directement proposé de renommer la méthode. Directement sur le nom actuel qui est sur fond vert, tapez un nouveau nom (par exemple "GestionZoneDeSaisie") : remarquez que le nom de l'appel se change en même temps. Validez ou cliquez sur "Appliquer".

La méthode est créée ainsi que le premier appel.




```
private void GestionZoneDeSaisie()
{
    txtTitre.Focus();
}
```

Appeler la méthode :

Partout où se trouve la ligne de code qui permet de remettre le focus sur la zone de texte, remplacez la ligne par l'appel de la méthode.

Par exemple, au chargement du formulaire.

```
private void frmFilms_Load(object sender, EventArgs e)
{
    lblTitre.Text = "";
GestionZoneDeSaisie();
}
```

Optimiser la méthode :

Logiquement, lorsque le focus est replacé sur la zone de saisie, celle-ci doit être préalablement vidée.

Ajouter la ligne correspondante.

```
private void GestionZoneDeSaisie()
{     txtTitre.Text =
"";
txtTitre.Focus();
}
```

Nettoyer le code :

Dans le reste du code, enlevez les lignes qui vident la zone de texte (attention, n'enlevez pas les lignes qui vides le label). A priori, ce n'est que sur le clic de btnAjouter.

```
private void btnAjouter_Click(object sender, EventArgs
e)
{     string titre =
txtTitre.Text.ToUpper();     if
(!titre.Equals(""))
    {
lstTitres.Items.Add(titre);
lblTitre.Text = titre;
txtTitre.Text = "";
    }
    GestionZoneDeSaisie();
}
```