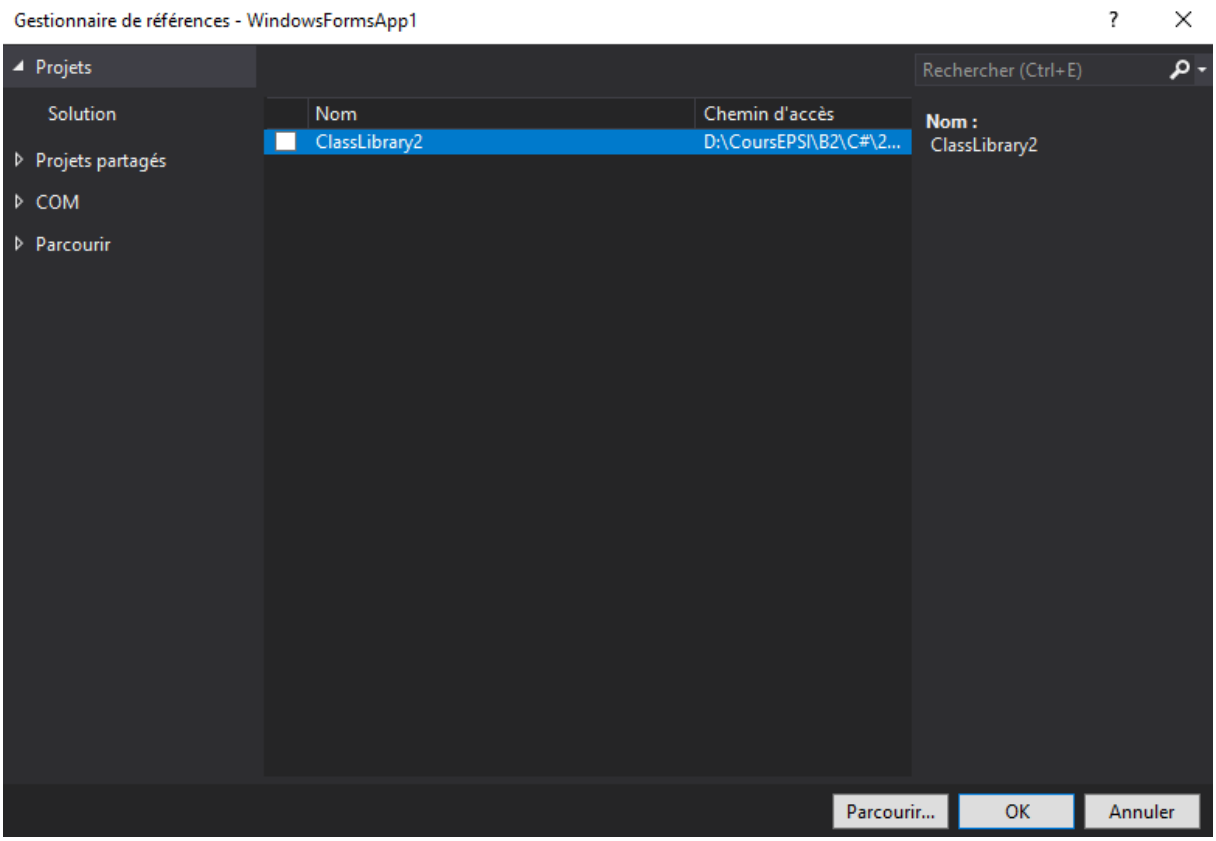
LANGAGE C# - Développement de composants

Les bibliothèques de classes

Une bibliothèque de classes est un projet qui peut créer via le template « Bibliothèque de classes ». Une bibliothèque de classes est un regroupement de classes dans lequel on retrouvera des classes qui traitent d'un même thème. Par exemple, nous pouvons regrouper dans une bibliothèque de classes toutes les classes qui nous permettront de faire des calculs mathématiques. Il y a deux façons d'intégrer une bibliothèque de classes dans une solution :

- On intègre directement le projet de bibliothèques de classes dans notre solution. Il suffira d'ajouter une dépendance à cette bibliothèque de classe dans le projet que nous souhaitons.
- On intègre la bibliothèque de classes via le fichier d'extension .dll qui aura été généré via un autre projet.

Dans les deux cas, la méthode d'intégration est la même :
Clique droit sur la partie Dépendances de votre projet -> Ajouter une référence de projet.



Si votre bibliothèque se trouve dans votre solution, elle apparaitra directement dans l'onglet Projets sur la gauche. Si vous l'intégrez via un fichier .dll, il faudra cliquer sur « Parcourir … » en bas à droite et indiquer l'emplacement du fichier. (Pour des soucis de partage, il est fortement conseillé de créer un dossier Librairies dans votre projet pour stocker les fichiers .dll).

LANGAGE C# - Développement de composants

Les interfaces graphiques (Winforms)

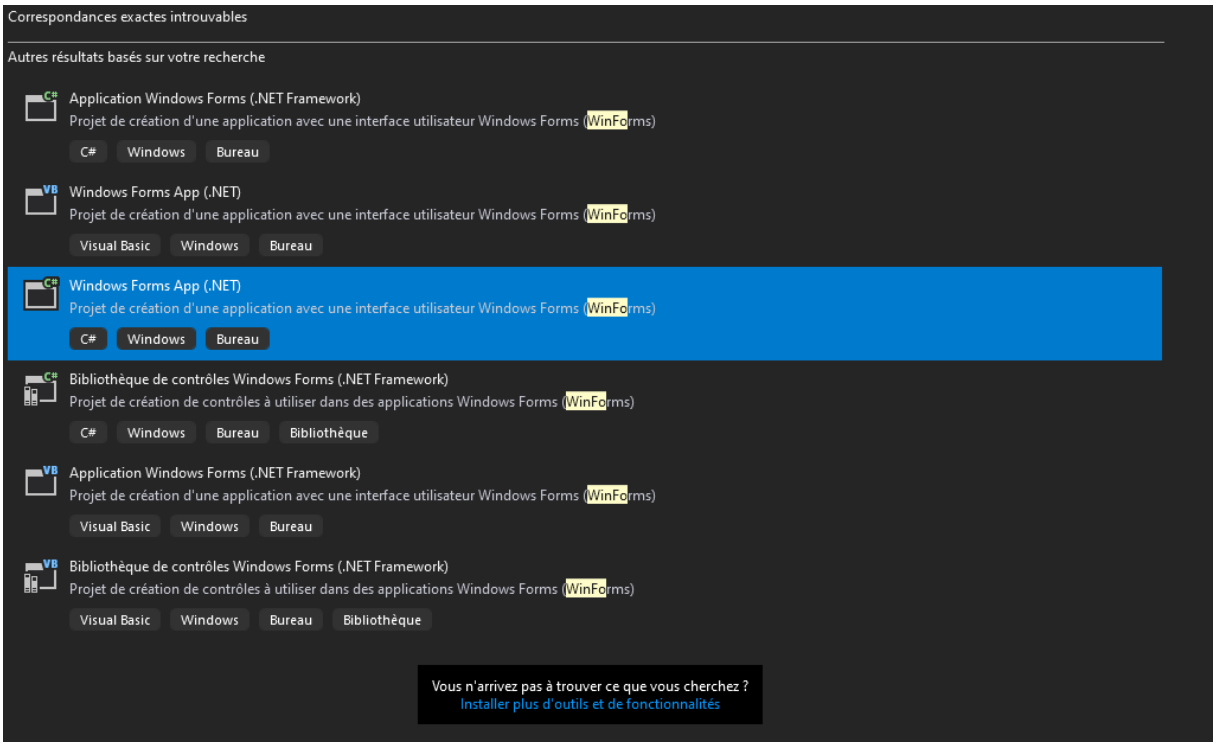
Grâce à la bibliothèque .NET, nous pouvons créer des logiciels avec des interfaces graphiques dans laquelle l'utilisateur aura beaucoup plus d'interaction, comme par exemple des cliques sur des boutons, survoler un endroit etc…

Dans ce module, nous nous attarderons sur Winforms.

Qu'est-ce que Winforms ?

WinForms ou Windows Forms est un ensemble de bibliothèque de classes intégré dans les framework .NET. Il permet d'avoir une plateforme pour créer des applications client lourd pour Ordinateurs. Une application WinForms est une application dite évènementielle, elle attend les actions de l'utilisateur afin de réaliser des opérations. WinForms est disponible uniquement sur les plateformes Windows.

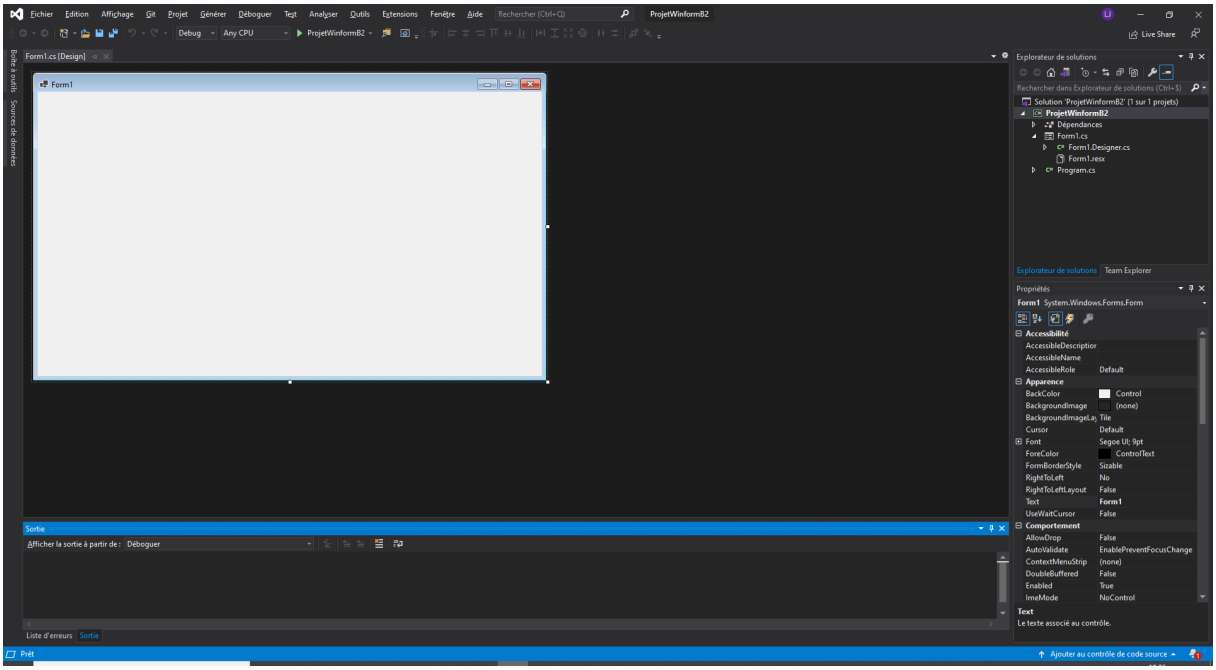
Création d'un projet WinForms



Après avoir choisi WinForms dans la liste des modèles de projet, il faudra indiquer le nom de l'application ainsi que sa localisation.

Un projet WinForms dans Visual Studio

Après la création du projet, Visual Studio lance l'environnement de travail.



Pour une meilleure approche de l'environnement de développement je vous conseille d'avoir cet état-là lors de la création d'un projet.
Les modules à afficher sont :

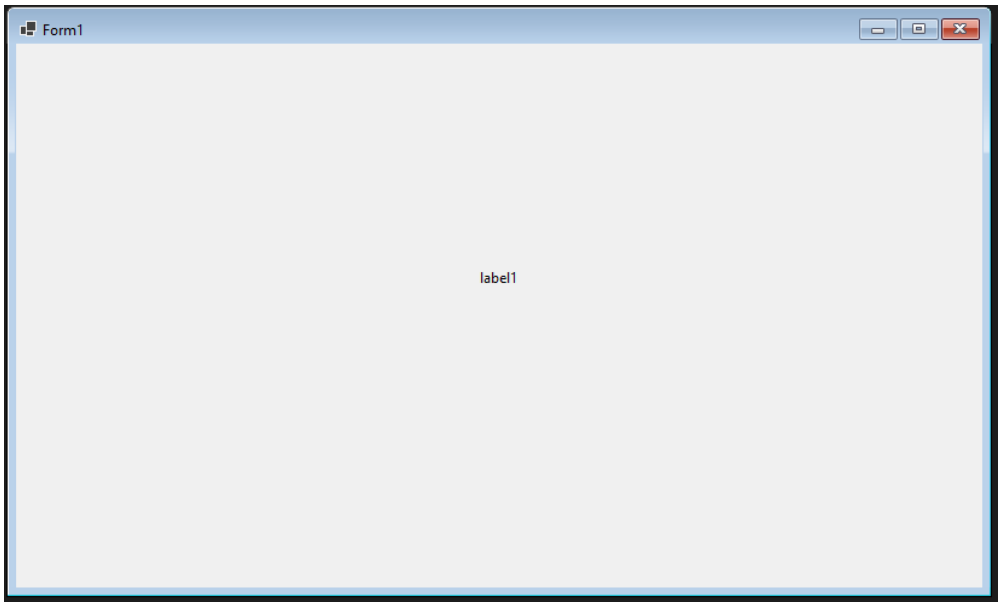
- Boite à outils
- Propriétés
- Explorateur de solutions

Notre projet contient deux fichiers C# : *Program.cs* et *Form1.cs*.
Le fichier *Program.cs* sera le point d'entrée de notre application. Dans nos exercices, nous n'aurons pas à modifier quoi que ce soit dans ce fichier. Le code écrit à l'intérieur permet de lancer la première fenêtre de notre application.
Cette première fenêtre est définie par le fichier *Form1.cs*. Ce fichier est un peu complexe dans sa structure. Tout d'abord, nous allons double cliquer sur le fichier. Une interface graphique apparait. Ici, on pourra glisser-déposer les éléments de la boîte à outils (qui se trouve sur la gauche). On trouvera dans la boîte à outils tous les éléments que vous pouvez disposer dans une application Winforms. Nous pouvons aussi accéder à la partie code de ce fichier en effectuant un clique droit sur le fichier et en choisissant Afficher le code (raccourci F7).
Lorsque que nous déroulons le fichier *Form1.cs*, nous avons deux nouvelles parties. La première *Form1.resx* permet de configurer des ressources constantes dans votre fenêtre. Nous n'utiliserons peu ou pas cette fonctionnalité. La deuxième partie *Form1.Designer.cs* nous permettra de configurer notre fenêtre via le code et non via l'interface graphique.

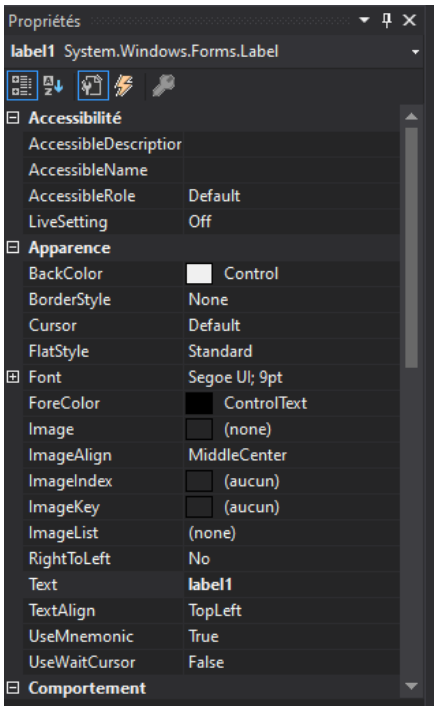
LANGAGE C# - Développement de composants

Ajout d'un composant

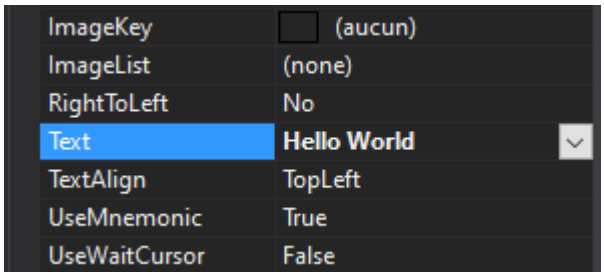
Pour ajouter un composant, il suffit de prendre le composant dans la boite à outils et ensuite de le faire glisser dans la partie centrale.
Dans notre exemple, nous allons prendre le composant Label. Un Label est un composant qui permet d'afficher du texte.



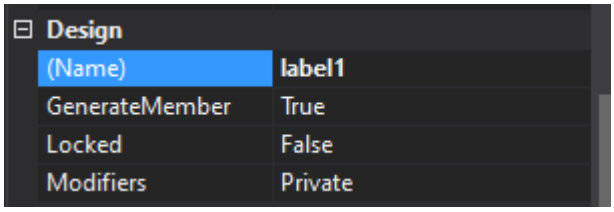
Pour chaque élément, nous pourrons soit changer directement les propriétés dans la boîte Propriétés qui se trouve en bas à droite par défaut.



Les propriétés du composant changent en fonction du composant utilisé. Nous n'allons pas faire ici la liste des différentes propriétés. Nous allons nous concentrer sur celle qui nous intéresse pour un Label c'est-à-dire celle qui nous permet d'afficher du texte.
Pour l'instant, le texte qui est affiché est « label1 ». Si on souhaite changer le texte, nous pouvons aller dans la propriété « Text ». Nous allons mettre « Hello World » à la place.



Changer une propriété via la boîte du même nom permet à l'application de se lancer avec des valeurs par défaut. Cependant, un Label, vu qu'il contient du texte, peut changer la valeur de son texte via le code. Pour cela, nous allons aller dans le code de *Form1.cs.*
Pour récupérer le composant dans notre code, c'est très simple. Il suffit juste d'intéragir avec lui grâce à son nom qui se trouve dans la propriété Name du composant. **Attention : il ne faut pas confondre Text et Name qui sont deux propriétés complètement différentes. L'un sert à donner une valeur de Text à notre Label, tant que l'autre sert à identifier dans le code (et dans le design) le Label.**
Par défaut, la propriété Name de notre Label sera « label1 ». Vous pouvez le modifier à votre guise.



Maintenant nous allons interagir avec lui, et donc changer la valeur de la propriété Text via le code.
Pour changer des valeurs à l'initialisation de notre fenêtre, nous irons ajouter du code dans le constructeur de Form1 après la méthode *InitializeComponent()* . Si vous essayez d'accéder à des composants après cette méthode, l'application crashera car elle n'aura pas instancié les composants que vous avez ajouté dans votre fenêtre.
Ici nous allons ajouter la ligne de code suivante :

```
label1.Text = "Bonjour le monde !";
```

Sauvegardez et lancer votre application. Une fenêtre apparait avec « Bonjour le monde » en plein milieu. Tout fonctionne !

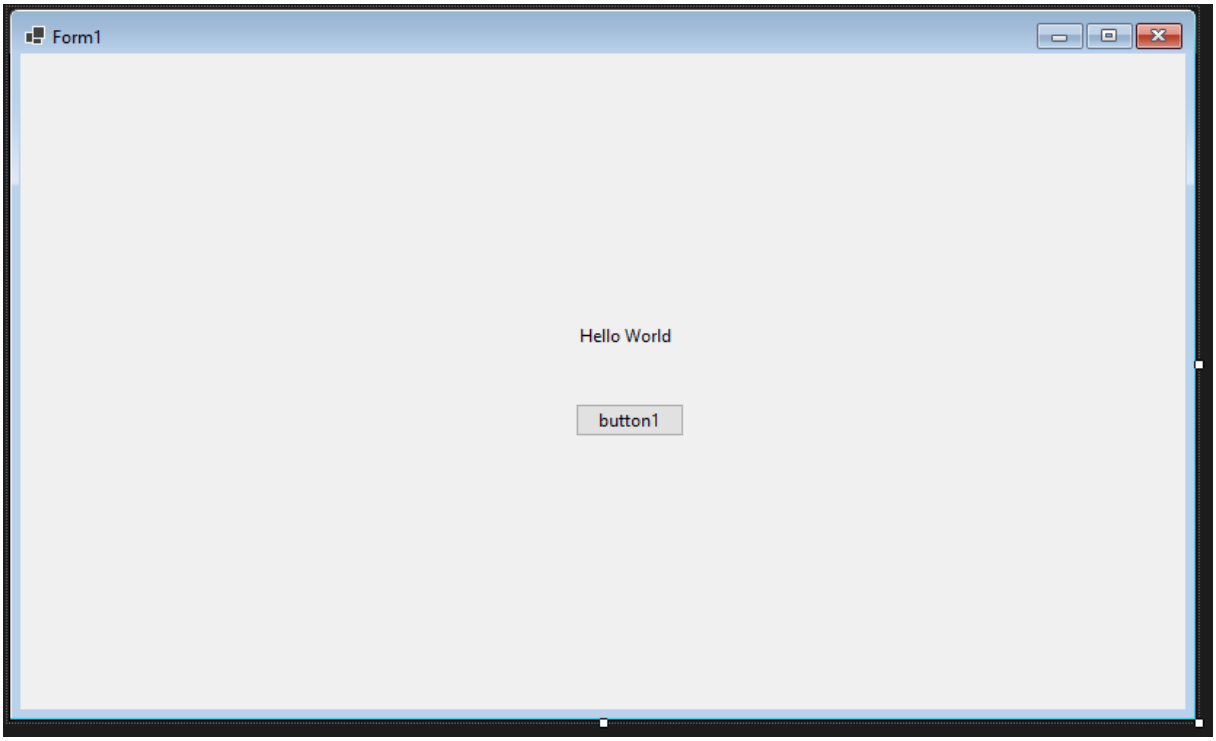
Remarque : Si vous retournez dans le designer, vous verrez que le texte du Label est toujours Hello World. C'est normal, il s'agit de sa valeur par défaut. Nous changeons ensuite sa valeur à l'exécution.

Voici une liste de composants que vous pourriez utiliser :

- **Label** : Permet d'afficher un texte
- **Button :** Permet à l'utilisateur de pouvoir faire un événement de clique avec la souris
- **PictureBox :** Permet d'afficher une image
- **CheckBox :** Permet à l'utilisateur de pouvoir remplir ou vider une case (Vrai ou Faux)
- **ComboBox :** Permet d'afficher des données dans une liste déroulante.
- **TextBox :** Permet à l'utilisateur d'entrer une donnée au clavier.

L'évènement Click

Pour ajouter le comportement naturel sur un élément quelconque d'un formulaire, il suffira de double-cliquer dessus.
Ici, nous allons ajouter un Button dans notre formulaire car il s'agit du composant naturel pour effectuer une interaction via un clique.



Ensuite, nous allons double-cliquer sur notre bouton. Nous allons arriver dans le code de notre formulaire, et du code va être généré.

```
private void button1_Click(object sender, EventArgs e)
{

}
```

Par défaut, cette fonction aura pour nom : Propriété Name du Button + « _Click ».

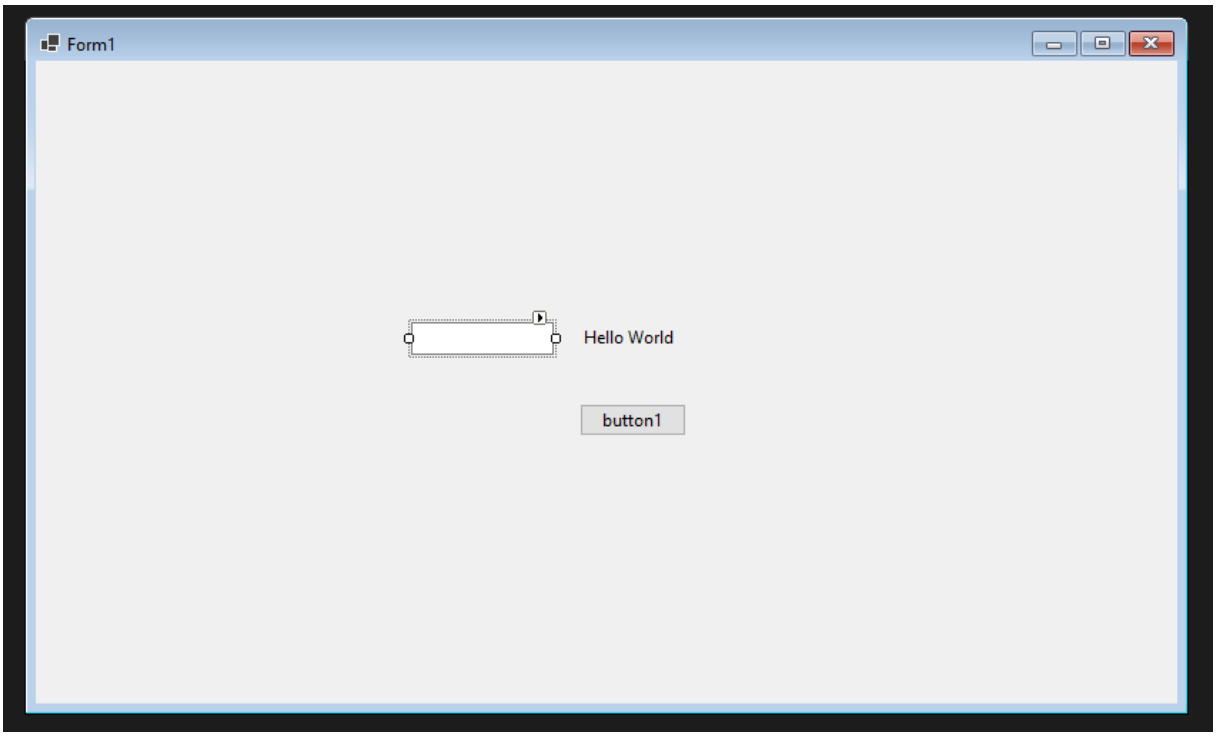
Les deux paramètres ne seront pas utiles dans nos exercices. Néanmoins, ils pourront être utiliser pour effectuer des optimisations.
Dans cette fonction, nous allons ajouter du code : la valeur de texte du Label déjà présent.

```
1 référence
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "On a appuyé sur le bouton.";
}
```

Il suffit de lancer l'application pour tester notre méthode.

Récupérer une donnée au clavier

Pour récupérer une donnée au clavier, nous allons devoir utiliser un composant qui nous permet d'écrire à l'intérieur. Pour cela, nous allons utiliser une TextBox. Ajoutons une TextBox dans notre formulaire.



Via notre bouton, nous allons mettre la valeur insérée dans la TextBox dans la valeur de texte du Label.

```
1 référence
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = textBox1.Text;
}
```

Nous pouvons utiliser aussi le comportement naturel de la TextBox. Il s'agit de savoir quand la valeur du texte change à l'intérieur de la TextBox. Nous allons double-cliquer sur la TextBox. La méthode est TextChanged(). Elle sera appelée dès lors que le texte aura changer dans notre TextBox. Nous allons juste recopier ce que l'utilisateur va insérer dans la TextBox dans le Label.

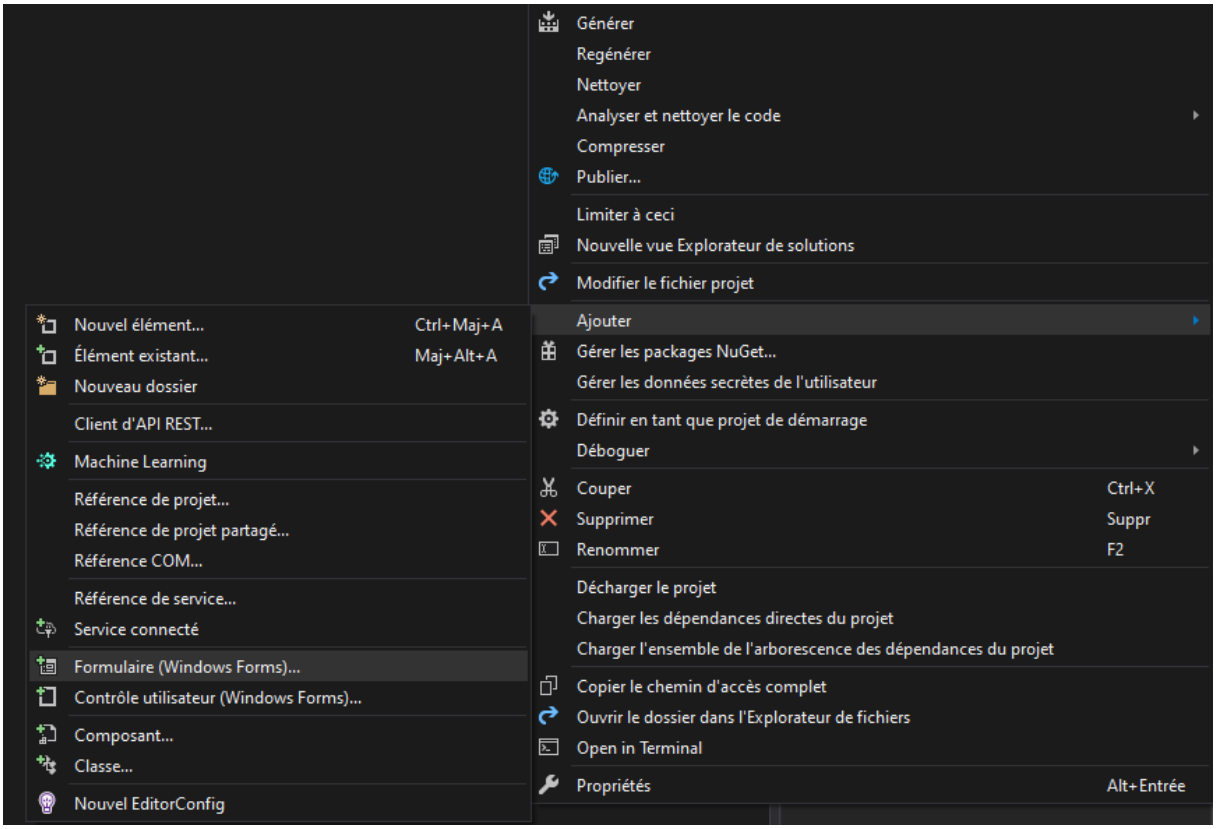
```
1 référence
private void textBox1_TextChanged(object sender, EventArgs e)
{
    label1.Text = textBox1.Text;
}
```

Exercice :

Faire un jeu de Plus ou Moins avec au moins un Label, un Bouton et une TextBox.

La navigation vers une nouvelle fenêtre

Nous allons tout d'abord ajouter à notre projet une nouvelle Form (qui correspond à une fenêtre). Pour cela, il faudra ajouter un Formulaire qui se trouve dans la partie Ajouter après avoir fait un clique droit sur le projet.



Dans cet exemple, nous l'appellerons Form2 (soit la valeur par défaut). Remarque : Rien ne vous empêche de mettre le nom que vous souhaitez. (N'oubliez pas la majuscule).

Voici le code afin de pouvoir lancer la nouvelle fenêtre que nous allons mettre dans la fonction Click d'un bouton :

```
1 référence
private void button1_Click(object sender, EventArgs e)
{
    Form2 form2 = new Form2();
    form2.ShowDialog();
}
```

Les messages d'alerte

Les messages d'alerte (ou popups) sont très pratiques pour avertir l'utilisateur. On peut aussi les configurer pour que l'utilisateur ait un choix à faire. Ici par exemple, si l'utilisateur n'appuie pas sur OK, on ne pourra pas lancer la navigation.

```
DialogResult result = MessageBox.Show("Voulez-vous changer de fenêtre ?", "Changement de fenêtre", MessageBoxButtons.OKCancel);
switch (result)
{
    case DialogResult.OK:
        Form2 form2 = new Form2();
        form2.ShowDialog();
        break;
    case DialogResult.Cancel:
        break;
}
```