# Windows Forms

## 1 Introduction

### 1.1 Objectifs

Durant ce TP, vous allez découvrir un nouveau langage, le `C#`. Vous serez confrontés à une nouvelle façon de coder, avec un nouveau langage, sous un nouvel environnement. Le but n'étant pas de vous perdre tout de suite, nous allons intégrer différentes fonctionnalités liées au `C#` et à *Visual Studio* afin de vous guider vers le bon chemin.
Cette semaine nous allons utiliser les *WinForms*, qui permettent de réaliser des interfaces graphiques, et ce de façon simple et efficace. Pour se fixer un objectif concret, vous devrez réaliser un pendu avec un affichage d'images, compteurs et récupération de texte, etc.

### 1.2 Le `C#`

Le `C#` – prononcé 'Ci Sharp' – est un langage de programmation orienté objet, créé par la société Microsoft, et notamment par un de ses employés, Anders HEJLSBERG, le créateur du langage `Delphi`.

Il a été créé afin que la plate-forme Microsoft `.NET` soit dotée d'un langage permettant d'utiliser toutes ses capacités. Il est très proche du `Java` dont il reprend la syntaxe générale, ainsi qu'au niveau des concepts – la syntaxe reste cependant relativement semblable à celle de langages tels que le `C++` et le `C`.

Contrairement au `Caml` qui est *fonctionnel* et avec typage inféré, le `C#` est un language *impératif* et son typage n'est pas inféré. Vous devrez donc spécifier le type choisi pour chacune des variables créées. Des règles basiques seront précisées plus tard.

### 1.3 Visual Studio

*Microsoft Visual Studio* est une suite de logiciels de développement pour Windows conçu par Microsoft. Il s'agit d'un ensemble complet d'outils de développement permettant de générer des applications. Ils permettent de développer des applications dans divers langages s'appuyant sur `.NET` tels que `C#` , `F#`, `Visual C++` et `Visual Basic`. Il est possible de développer des types d'applications très variés : applications graphiques, en console, des bibliothèques de classes, des services Windows ou encore des sites web, le tout grâce à l'environnement de développement intégré *IDE* – Integrated Development Environment.

Les étudiants d'Epita peuvent télécharger gratuitement *Visual Studio* en allant sur le site *Microsoft DreamSpark for Academic Institutions*[1] ou en téléchargeant la version *express* gratuite sur le site Microsoft.

1. https://www.dreamspark.com/Institution/Access.aspx : Rechercher EPITA.

# 2 Rendu

Le TP devra être rendu en réalisant une archive `zip`.

## 2.1 Fichier `AUTHORS`

Ce fichier contient votre *login*, sous la forme suivante : une étoile `*`, un espace, votre *login* (`login_x`) et un retour à la ligne – représenté par le caractère `$` dans l'exemple ci-dessous.

```
* login_x$
```

## 2.2 Arborescence de l'archive `zip`

```
rendu-tp0-login_x.zip
    | login_x/
        | AUTHORS
        | HelloWorld/
            | HelloWorld/
            | HelloWorld.sln
        | ImageViewer/
            | ImageViewer/
            | ImageViewer.sln
        | HangedGame
            | HangedGame/
            | HangedGame.sln
```

# 3 Les bases du langage `C#`

Le `C#` comporte des mécanismes de programmation différents du `Caml`. Ces mécanismes seront abordés au fur et à mesure des TP. Ce premier TP ne nécessite pas beaucoup de programmation, il consiste à découvrir *Visual Studio* et la création de fenêtres.

La documentation du langage `C#` se trouve sur *MSDN* :
http://msdn.microsoft.com/fr-fr/library/67ef8sbd.aspx

## 3.1 Notions de base

- **Commentaires** : Pour commenter une ligne en utilisez un double slash `//`. Pour former un bloc de commentaire, utilisez `/* Mon commentaire. */`.
- **Bloc** : Un bloc est formé par deux accolades `{ }`. Un bloc contient zéro ou plusieurs instructions. Un bloc peut contenir des bloc imbriqués.
- **Instruction** : Les actions effectuées par un programme sont exprimées dans des *instructions*. Une instruction peut être constituée d'une ligne de code qui se termine par un point-virgule `;`, ou d'une série d'instructions sur une ligne dans un autre bloc. Exemples d'instructions :
  - déclaration de variables ;
  - attribution de valeurs ;
  - appel de méthodes ;
  - exécution de boucles ;
  - branchement à un autre bloc de code en fonction d'une condition donnée.

— **Expression** : Une instruction est une suite d'un ou plusieurs opérandes ou plusieurs opérateurs qui permettent de calculer une valeur.
— **Opérateurs** : Les opérateurs arithmétiques ne sont pas *typés* comme le `Caml` : `+ - * /`, *i-e* on faire des calculs entre un nombre entier et un flottant. Il existe de nombreux opérateurs en `C#`, un autre qui sera utile pour ce TP est le point `.` qui permet d'accéder à un membre – lors des manipulations d'objets par exemple.
— **Types** : Le `C#` définit les types suivants : `int`, `float`, `double`, `char`, `string`, `objects`, etc.
— **Fonction** : Le *prototype* d'une fonction est composé du type de retour, nom de la fonction, et d'éventuels paramètres avec leur type entre parenthèses.

**Exemple : fonction factorielle**

La fonction factorielle – récursive – en `C#` peut s'écrire de la façon suivante :

Code source

```
1 int fact(int i)
2 {
3     if (i <= 1)
4         return 1;
5     else
6         return i * fact(i - 1);
7 }
```

Remarquez que les accolades de *bloc* pour le `if` et `else` ne sont pas obligatoires si le bloc contient une seule instruction.

## 3.2 Création d'un nouveau projet Windows Forms

— Lancez Visual Studio – si besoin précisez `C#` au lancement.
— Cliquez sur **Fichier**, **Nouveau** et **Projet...**
— Sélectionnez **Application Windows Forms**

L'IDE [2] Visual Studio est votre outils de développement. N'hésitez pas à le personnaliser pour le rendre confortable. Pour lancer l'exécution d'un projet, appuyez sur la touche **F5**. N'oubliez pas d'indenter votre code, le raccourci suivant permet de le faire : `Ctrl + K + Ctrl + D`.

L'explorateur de solutions – situé sur le bord droit de l'IDE – permet d'afficher une vue d'ensemble du projet. Il contient :
— Solution `WindowsFormsApplication` (1 projet)
  — WindowsFormsApplication : l'application.
    — *Properties* : Ressources du projet.
    — *Références* : Bibliothèques utilisées dans l'application.
    — `Form1.cs` : Contient votre code et affiche l'éditeur graphique.
      — `Form.Designer.cs` : code généré par l'éditeur graphique.
      — `Form1.resx` : **Contient votre code**.
      — `Program.cs` : Point d'entrée du programme.

2. Integrated Development Environment

### 3.3 Windows Forms

Les *Windows Forms* permettent de créer des interfaces graphiques sous Windows assez simplement. Visual Studio fournit une *boîte à outils* où on trouve la liste des *WinForms* disponibles. Pour les utiliser, il suffit d'effectuer un simple glisser-déposer.

Lorsqu'un *Form* est sélectionné, on peut le paramétrer dans la fenêtre *propriétés* de VS où se trouve différents *champs* du *WinForm*. À partir de là, on peut le personnaliser – taille, couleur, position, contenu, etc.

Dans la fenêtre *propriétés* se trouve également un onglet *éclair*. On peut alors paramétrer diverses actions de l'utilisateur – clic de souris, appui sur une touche du clavier, etc. En effectuant un double-clique sur le *champ* `Click`, on se retrouve dans une fenêtre d'édition de code. Une *méthode* a été automatiquement créée par VS. Cette *méthode* permet de créer un *callback* pour définir le comportement de l'application pour une action utilisateur donnée. Le code source dans la fenêtre d'édition contient les différentes méthodes de la fenêtre `Form`.

## 4 Exercice 1 : `HelloWorld`

*Dans les exercices qui vont suivre, vous profitez d'une liberté de création. Expérimentez !*

### 4.1 Manipulations

Vous devez commencer par créer un nouveau projet Windows Forms qui prendra le nom de l'exercice : `HelloWorld`. Créez une interface graphique qui contiendra – au moins – les éléments suivants :

- Un *Button* `button_say` qui contient le texte `Say`.
  - Modifiez le nom du bouton `(Name)` dans les *properties*.
  - Modifiez le contenu du *champ* `Text` dans les *properties*.
- Un *Label* `label_say` qui ne contient pas de texte.
  - Modifiez le nom de l'étiquette `(Name)` dans les *properties*.
  - Supprimez le texte du *champ* `Text` dans les *properties*.

L'interface graphique a été conçue et paramétrée. Il faut maintenant la faire fonctionner en créant des connexions entre les *WinForms*. Double-cliquez sur le bouton `button_say` de l'interface graphique *ou* dans l'onglet représenté par un *éclair* et double-cliquez sur l'action `Click`. L'éditeur de code s'affiche. Assignez le texte du label `label_say` :

Code source

```
1 private void button_say_Click(object sender, EventArgs e)
2 {
3     label_say.Text = "Hello World !";
4 }
```

Appuyez sur **F5** pour lancer l'application avec le *débuggeur*.

### 4.2 Améliorez

*Liberté de création dans cette partie.*

- Dites *bonjour* en plusieurs langues en utilisant une `ComboBox`.

— Récupérez le nom de la personne à qui vous dites *bonjour* en utilisant une `TextBox`.
— Dites *bonjour* en couleur en ajoutant un `Button` et la boîte de dialogue `ColorDialog`.

# 5 Exercice 2 : `ImageViewer`

## 5.1 Manipulations

Le but de cet exercice est de réaliser un lecteur d'image. L'interface graphique devra comporter les éléments suivants :

— Un *Button* : `button_open` avec le texte `Open`.
— Une *PictureBox* : `pictureBox_viewer`
— Une boîte de dialogue *OpenFileDialog* : `dialog_open`

Affecter l'action de `button_open` :

Code source

```
1 private void button_open_Click(object sender, EventArgs e)
2 {
3     dialog_open.ShowDialog();
4     Image img = Image.FromFile(dialog_open.FileName);
5     pictureBox_viewer.Image = img;
6 }
```

## 5.2 Améliorez

*Liberté de création dans cette partie.*

— Permettre au lecteur d'image d'afficher des images haute résolution en effectuant un redimensionnement.
— Afficher le nom de l'image et ses caractéristiques – hauteur, largeur, format de fichier en récupérant l'extension, etc.
— Habiller l'interface graphique avec des couleurs.

# 6 Exercice 3 : `HangedGame`

Maintenant que vous avez quelques bases en `C#` sous Visual Studio, vous allez enfin pouvoir mettre en pratique pour un vrai petit projet. On va tout d'abord rappeler tout ce qui a été vu pour le moment. Vous savez utiliser des **boutons**, des **textbox**, **afficher des images**, etc. On va donc réutiliser ces connaissances pour faire le jeu du pendu.

## 6.1 L'interface

Des éléments obligatoires sont à intégrer dans cette interface :

— 3 `button` s :
  — 'Nouvelle partie'
  — 'Fin de partie'
  — 'Valider'
— Une `TextBox` pour récupérer la lettre proposée par l'utilisateur.
— Une `RichTextBox` pour afficher le mot avec les lettres trouvées.

— Une `ProgressBar` pour l'avancement de la découverte du mot.
— Une `PictureBox` pour voir notre pendu.
— Un `Label` pour montrer le nombre restant de coups.

Vous pouvez donc, avec tout ces éléments créer un jeu du pendu respectable. La suite d'images est disponible en suivant ce lien : http://perso.epita.fr/~acdc/

## 6.2 But du jeu

Vous connaissez tous le jeu du pendu, mais voici quelques explications sur la procédure à suivre pour sa conception :

— 'Nouvelle partie' :
  — `ProgressBar` remise à 0.
  — Nombres d'essais mis à 7. - *par exemple* -
  — Image de base.
  — `RichTextBox` nettoyée.
  — Bouton 'Nouvelle Partie' grisé.
  — Boutons 'Fin de partie' et 'Valider' activés.

— Bouton 'Valider' cliqué :
  — Si la lettre est présente dans le mot, l'ajouter à la `RichTextBox` et augmenter la `ProgressBar` de 1.
  — Sinon le nombre de tests restants diminue et l'image suivante est affichée.

— La partie est perdue si le nombre d'essais arrive à 0.

— On peut stopper la partie en cours avec le bouton 'Fin de partie'.

## 6.3 Boni

*Vous êtes totalement libres sur cette partie, si vous laissez les parties obligatoires présentes lors du rendu.*

Quelques exemples :

— Ajout d'un historique des essais dans une `ListBox`.

Code source

```
1 // Permet d'ajouter str dans la liste.
2 MyList.Items.Add(str);
3 // Permet d'effacer la liste.
4 MyList.Items.Clear();
```

— Ajout d'un dictionnaire avec un certain nombre de mots. La selection du mot se fait aléatoirement. *Il faut se renseigner sur l'utilisation d'un tableau – ajout, accès.*