1 Introduction

1.1 Description générale du langage

Le C# est un **langage de programmation Orienté Objet**.

Il peut être utilisé pour créer des applications type Windows (**Client Lourd**), de la programmation de sites en lignes (**Client Léger**) et des applications qui se rapprochent d'applications Windows mais fonctionnent dans un navigateur (**Client Riche**).

Le code C# est compilé dans un code de langage intermédiaire (CIL) qui sera par la suite transformé en un CLR pour Common Langage Runtime et enfin en exécutable binaire. Cette particularité du c# le rends plus **facile à exporter dans d'autres systèmes d'exploitations** (passer de Windows à linux ou mac ou encore être capable de faire fonctionner un programme qui n'à pas été spécialement écrit pour un type particulier d'architecture processeur.

On peut ainsi écrire des *.EXE* qui sont des applications et des *.dll* qui sont des librairies partagées par plusieurs applications *.EXE*.

Les fichiers de code C# : portent **l'extension *.cs*** et **commencent par une majuscule**

Le C# est un langage qui est sensible à la case. Cela signifie qu'une majuscule dans le nom d'une variable ou d'une fonction indique une autre variable ou une autre fonction.

1.2 Fiche Technique

Type de langage :

Orienté objet

Case

PascalCase

Particularités techniques:

Contrairement à son cousin Java, on peut (Pas conseillé par les règles de Clean Code):

- écrire la définition d'une classe dans un fichier qui porte un nom différent
- écrire plusieurs définition de classes dans le même fichier
- répartir la définition d'une classe sur plusieurs fichier (Utile dans le cas d'un travail ou il faut coordonner plusieurs équipes ou le cas de l'utilisation de Framework)

1.3 Programme de base

```cs
using System;                                   // *1

namespace TestingPurposes                       // *2
{
    class Program                               // *3
    {
        static void Main(string[] args)         // *4
        {
            Console.WriteLine("Hello World!");  // *5
        }
    }
}
```

1.3.1 Les Importations

Un Projet informatique peut demander d'utiliser des fonctionnalités programmées par d'autres. On peut utiliser ces fonctions grâce à l'importation. Similaire à ce qui se fait en Java avec *import*, On utilise en C# les mots clés **using** et **import**.

using

using est utilisé pour préciser que la classe utilise une librairie existante. Contrairement à import en java, using porte sur un **namespace tout entier** et non sur une (ou plusieurs) classe(s).

import

import porte sur une ou plusieurs classes.

1.3.2 Les namespaces

Un **namespace** (ou espace de noms en français) est une **structure hiérarchique groupant des classes** par utilité ou par contexte d'utilisation [1]. On peut aussi utiliser cette répartition pour différencier les différentes couches d'un programme informatique.

1.3.3 Les Classes

Les Classes sont, en Orienté Objet la base de la formation des objet. L'instanciation d'une classe donnera un objet.

1.3.4 Les fonctions

Les fonctions représentent des opérations qu'un programme peut effectuer. Elle regroupent des instructions. Par exemple : la fonction manger peut faire intervenir les instructions : prendre nourriture avec cuillère, avaler, digérer.

1.3.5 Les instructions

Il s'agit d'une instruction

1.4 Les Variables

Les attributs commencent par une minuscule en C#. Il faut faire attention car le C# est un langage sensible à la case. cela signifie que l'utilisation d'une majuscule à la place d'une minuscule sera interprétée différemment par le programme.

En C#, les variables sont initialisées au zero du type.

1.4.1 Les Entiers (int)

Les entiers sont avec les doubles, l'un des 2 types de variables les plus utilisées. Ils permettent de stocker des nombres entiers.

1.4.2 Les Réels (double)

Utilisé pour stocker des nombres réels.

Attention aux erreurs d'arrondis

1.4.3 Les chaînes de caractères (string)

String et string sont équivalent.

En C#, On peut comparer 2 strings entre eux en utilisant des ==.

Il est en effet possible de redéfinir la signification de certains symboles en c# comme (+-* et ==).

1.4.4 Les caractères (char)

1.4.5 Les booléens (bool)

Les nombres booléens valent soit true, soit false (vraix/faux).

1.4.6 Les nombres décimaux (decimal)

En comparaison aux doubles, *decimal* n'arrondi pas la partie décimale et permets donc une plus grande précision.

1.4.7 Boxing

La conversion entre int et Integer est appelée **boxing**.

L'**auto-boxing** est quand le boxing est effectué automatiquement par le programme.

2. L'orienté Objet

Un objet est identifié par son nom, son état et son comportement. L'état est l'ensemble des valeurs données aux attributs de la classe dont il est une instance et ce, à un moment donné. L'état peut évoluer au cours du temps.

2.1 Les constructeurs

Un constructeur est une fonction au sein d'une classe qui à pour objectif lors de son appel d'instancier l'objet défini par la classe.

```
string firstName;
string lastName;
DateTime birthday;
string boy;

public Player(string firstName, string lastName, DateTime birthday, bool
boy)
{
    this.firstName = firstName;
    this.lastName = lastName;
    this.birthday = birthday;
    this.boy = boy;
}
```

This est utilisé pour définir l'objet que l'on vient de créer.

2.1.1 Constructeurs Multiples

Si l'on souhaite définir plusieurs constructeurs dans une classe, il faut faire un appel explicit au contrôler principal. celui-ci est le plus générique.

```
public Player(string firstName, string lastName, DateTime birthday) :
this(lastName, firstName, birthday, true)
{ }
```

Cette méthode permets d'obtenir un **point de modification unique**.[2]

2.4 Encapsulation

L'encapsulation est le fait de cacher aux yeux de l'extérieur une partie de l'objet **déclarée privée**,et de ne la rendre accessible,si c'est souhaité, uniquement via des **méthodes publiques** (getteurs et setteurs).

Les Tests Unitaires

Pour vérifier le bon fonctionnement d'un programme, il est courant de pratiquer des test sur chaque partie indépendamment afin de garantir la stabilité du programme.

```
static void AssertBool (string test, bool expected, bool observed)
{
    Console.WriteLine("Test: " + test);
    Console.WriteLine("Expected: " + expected + ", observed: " + observed);
    Console.WriteLine(expected == observed ? "Ok!" : "KO !!!");
```

```
    Console.WriteLine();
}

static void AssertString (string test, string expected, string observed)
{
    Console.WriteLine("Test: " + test);
    Console.WriteLine("Expected: " + expected + ", observed: " + observed);
    Console.WriteLine(expected == observed ? "Ok!" : "KO !!!");
    Console.WriteLine();
}

static void TestValidLogin()
{
    AssertBool("Herbert", true, ForumUtils.ValidLogin("Herbert"));
    AssertBool("empty string", false, ForumUtils.ValidLogin(""));
    AssertBool("fart", false, ForumUtils.ValidLogin("fart"));
    AssertBool("FART", false, ForumUtils.ValidLogin("FART"));
    AssertBool("FaRt", false, ForumUtils.ValidLogin("FaRt"));
}

```

Sécurisation (Hash)

Exemple from : https://www.godo.dev/tutorials/csharp-md5/

```
using System.Security.Cryptography;
using System.Text;

public static string MD5Hash(string text)
        {
            MD5 md5 = new MD5CryptoServiceProvider();

            //compute hash from the bytes of text
            md5.ComputeHash(ASCIIEncoding.ASCII.GetBytes(text));

            //get hash result after compute it
            byte[] result = md5.Hash;

            StringBuilder strBuilder = new StringBuilder();
            for (int i = 0; i < result.Length; i++)
            {
                //change it into 2 hexadecimal digits
                //for each byte
                strBuilder.Append(result[i].ToString("x2"));
            }

            return strBuilder.ToString();
        }
```

3 Coder en c#

3.1 Les variables

Un attribut commence par une **minuscule** et fait maximum 511 char (alphaNum, accents, 'ç', Mu et '_'). par défaut, chaque attribut standard est initialisé à 0 du type.

Le Framework .NET comprends beaucoup de types prédéfinis, nottament des alias et un type décimal qui retiens les chiffres des nombres avec une décimale importante (vs approximation binaire des autres types)

On différencie aussi les classes et les structures dans l'écriture et la gestion de la mémoire.

Contrairement en java, en C#, les uint32 et les int ne sont pas deux entités similaires mais des synonymes.

Les strings ou Strings sont aussi synonymes. Ils peuvent être comparés entre eux via l'utilisation de '' contrairement au java. On peut ainsi les comparer car le C# authorise la redéfinition des méthodes usuelles (ex: ToString) mais aussi des comparaisons '', '+' ou '*'.

On peut définir une variable entière comme suit :

```
int nom_variable; // variable entière
string nom_variable; // variable texte
bool nom_variable; // variable boolenne true/false
decimal nom_variable; // variable qui retiens les décimaux (nécéssaire quand on traite avec de l'argent)

// mettre une valeur dans une variable
nom_variable = 10;
nom_variable++;//ajoute 1
nom_variable--;//retire 1
nom_variable+=2;//équivalent à var= var+2
nom_variable-=2;//idem suppra
nom_variable/=2; //idem suppra
```

> >string.Empty est équivalent à "".

On peut concaténer des strings avec l'opérateur +, '' est le caractère d'échapement et il permets nottament d'écrire :

```
string tabulation = "\t";
string retourLigne = "\n";
string slash = "\\";
string slash = @"\"; // le @ évite de devoir écrire le double slash
```

3.2 Conditions

3.2.1 IF

```
if (cond){}
else if(cond){}
else{}
```

3.2.2 Switch

```
switch (variable){
    case 1: // si la variable variable vaut 1 on exécute le code
        /* code */
        break;
    case 2:
        /* code */
        break;
    default: // si tous les case échouent
        /* code */
        break;
}
```

On peut aussi grouper plusieurs case qui auraient le même effet :

```
switch (variable){
    case 1: // si la variable variable vaut 1,2 ou 3 on exécute le code
    case 2:
    case 3:
        /* code */
        break;
    case 4:
        /* code */
        break;
    default: // si tous les case échouent
        /* code */
        break;
}
```

3.3 Les Méthodes

Une mathode s'écrit sous la forme suivante :

```
static void AffichageBienvenue(string nom,string prenom){/* code */}
```

- *static* : ...
- *void* : est la valeur de retour de la fonction (ici elle ne retourne rien)
- *AffichageBienvenue* : nom de la fonction
- *string nom,string prenom* : arguments de la fonction

> >une méthode statique ne peut appeler que des méthodes statiques

3.4 Les Tableaux et listes

Il existe une différence entre les tableaux et les listes. Les *tableaux* peuvent être **multidimentionels** tandis que les *listes* n'ont **pas de taille prédéfinie**

3.4.1 Tableaux

```
string[] joueurs = {"Bob","Marc","Tom"};
Console.WriteLine (joueurs[2]); //--> Tom
Array.Sort(joueurs); //classe les éléments du tableau
```

3.4.2 Listes

```
List<int> chiffres = new List<int>();
chiffres.Add(4); // la liste contient : 4
chiffres.Add(5); // la liste contient : 4,5
chiffres.RemoveAt(1); // la liste contient : 4

foreach(int chiffre in chiffres){...} //effectue une ittération sur tous les éléments de la liste

IndexOf(var);//recherche var dans la liste et retourne son indice
```

3.4.3 Enumérations

Une énumération est une liste de valeurs possibles.

```
enum jours{Lundi,Mardi,Mercredi,Jeudi,Vendredi}//stocké sous forme int mais display

jours variable = jours.lundi;
console.(variable);
```

3.5 Dates

Ces deux lignes sont équivalentes :

```
console.writeline(DateTime.now);
System.console.WriteLine(System.DateTime.now);
```

3.6 Boucles

```
for (int i=0;i<tab.Length;i++){/*code*/};
while(cond){/*code*/};
foreach(string jour in jours){/*code*/} // Attention car foreach est en lecture seulement. Il peut boucler sur tout type énumérable
break; // dans une boucle = sortie de Boucle
continue; // Passe à l'itération suivante de la boucle
```

> Attention aux **Overflows** ou dépassement de capacités. (dépassement de la taille maximale d'une variable)

3.9 Attributs et constructeurs

Créations de getteurs et setteurs

```
public string GetName(){
   return this.lastName +""+ this.firstName;
}
```

Constructeurs

En java, un constructeur peut faire appel à un autre constructeur avec this(...) mais celui-ci doit être la première instruction du code. C'est **différent en c#**
On a un construceur complet et d'autres qui y font référence.

```
public Player(string name, DateTime birthday, string firstName):
this(lastName,birthday,firstName,true)
{ }
```

3.10 Redéfinition de méthodes

Le super de java est remplacé par base en c#. On peut cependant redéfinir une méthode via :

```
public override string ToString()
{
string output="";
// À vous de compléter
return output;
}
```

---

1. En entreprise, les namespaces sont souvent l'endroit ou se situent les coordonnées de l'auteur/de l'entreprise (Ex : Henallux.IG.ProjetTest.DAO ou Henallux.IG.ProjetTest.DAO) ↩

2. Le point de modification unique permets de limiter les erreurs dans le code liée à la redondance de code. ↩