Langage C# Romain Verhaeghe

Dans ce Markdown, je vais vous présenter 5 thèmes importants en **Langage C#**:

- Manipulation des chaînes de caractères
- Classe
- Record
- Types référence et types valeur
- Interface

Manipulation des chaînes de caractères

En C#, les chaînes de caractères sont des objets de types **String** (ou System.String class) qui représentent une suite de caractères.

```
String chaine1="Hello";
string chaine2="World!";
```

On peut effectuer différentes opérations sur celles-ci :

- concaténation (Concat(String,String))
- comparaison (Compare(String,String),Equals(String,String))
- extraction (Substring(Int32))
- recherche (Contains(String))
- changement (toUpper(),toLower())
- ...

Concaténation

La concaténation consiste à ajouter une chaîne de caractères à la suite d'une autre chaîne de caractères. On peut le faire de plusieurs manières : **+ (l'opérateur)**, **String.Concate(String,String)**, String.Join(), String.Format(), StringBuilder.Append(), et **String Interpolation**.

Avec **l'opérateur** :

```
Console.WriteLine("Hello"  +  " "  +  "String "  +  "!");
```

Avec la méthode **String.Concate(String,String)** :

```
string lastName =  "Verhaeghe";
string firstName =  "Romain";
string Name =  string.Concat(firstName, lastName);
```

Avec le **String Interpolation** :

```
string player =  "Neymar";
string club =  "PSG";
```

```
int year =  2017;
int price =  222;
string sentence =  $"{player} est le transfert le plus cher du
monde lorsqu'il a rejoint le {club} en {year} pour {price}
millions d'euros.";
Console.WriteLine(sentence);
```

Comparaison

La comparaison de chaîne de caractères se fait par valeurs avec différentes méthodes dont : **String.Compare**, **l'opérateur ==** et **String.Equals**.

Avec la méthode String.Compare(String,String), la comparaison se fait lettre par lettre et par ordre alphabétique. Elle retourne 0 si les deux chaînes sont égales, 1 si la première chaîne est plus grande (dans l'ordre alphabétique), et -1 si elle est plus petite que la deuxième.

```
string s1 = "hello";
string s2 = "world";
string s3 = "hello";
string s4 = "csharp";
Console.WriteLine(string.Compare(s1,s2)); // Output = -1
Console.WriteLine(string.Compare(s1,s3)); // Output = 0
Console.WriteLine(string.Compare(s3,s4)); // Output = 1
```

L'opérateur == permet de comparer le contenu des chaînes de caractères, mais celui-ci est sensible à la casse et peut comparer des valeurs null.

```
string str1 = "Csharp";
string str2 = "csharp";
str1 == str2; //faux
```

Tandis que la méthode **String.Equals(String)** n'est pas sensible à la casse (avec StringComparison.OrdinalIgnoreCase), et retourne l'exception NullReferenceException si la chaîne est null.

```
string s1 = "hello";
string s2 = "Hello";
string s3 = "csharp";
string s4 = "CSHARP";
Console.WriteLine(s1.Equals(s2));//faux
Console.WriteLine(s2.Equals(s3));//faux
Console.WriteLine(s4.Equals(s3, StringComparison.OrdinalIgnoreCase));//vrai
```

Extraction

L'extraction en C# consiste **à récupérer une chaîne de caractères à l'intérieur d'une chaîne de caractères**. Elle se fait à l'aide de la méthode String.Substring(Int32 StartIndex) ou String.Substring(Int32 StartIndex, Int32 EndIndex) :

```
string sentence = "Ceci est un string";
string extract1 = sentence.Substring(8);//"un string"
string extract2 = sentence.Substring(4,8);//"est un s"
string extract3 = sentence[^6..]; //"string"
```

Cela fonctionne aussi avec la première occurrence d'un caractère ou chaîne de caractères :

```
string players = "Cristiano Ronaldo, Neymar, Lionel Messi, Kylian Mbappé";
int firstStringPosition = players.IndexOf("Ronaldo");
int secondStringPosition = players.IndexOf("Mbappé");
string stringBetweenTwoStrings = players.Substring(firstStringPosition,
secondStringPosition - firstStringPosition);
Console.WriteLine(stringBetweenTwoStrings);
//"Ronaldo, Neymar, Lionel Messi, Kylian "
```

Recherche

Pour recherche une chaîne de caractères à l'intérieur d'une chaîne de caractères, on utilise la méthode **String.Contains(String)** qui retourne une valeur booléenne (True ou False) :

```
string s1 = "Csharp ";
string s2 = "Hello";
string s3 = "Sharp";
Console.WriteLine(s1.Contains(s2));//False
Console.WriteLine(s1.Contains(s3));//False
Console.WriteLine(s1.Contains(s3,StringComparison.OrdinalIgnoreCase));//True
```

Changement

La conversion d'une chaîne de caractères peut être fait de manières : **String.ToLower()** pour tout mettre en minuscule, et **String.ToUpper()** pour tout mettre en majuscule :

```
string s1 = "Hello C#";
string s2 = s1.ToLower();
string s3 = s1.ToUpper();
Console.WriteLine(s2);//"hello c#"
Console.WriteLine(s3);//"HELLO C#"
```

La conversion peut aussi se faire de string à int, et inversement avec les méthodes : **Int32.ToString()** et **Int32.Parse().**

```
string s1 = "12345";
int s1nbr = Int32.Parse(s1);
int s2nbr = 6789;
string s2 = s2nbr.ToString();
Console.WriteLine(s1nbr);//s1nbr=12345
Console.WriteLine(s2);//s2="6789"
```

Exercice

À partir de ce string :

```
string sentence = "sSDzoTeVhZ0hEJfHanVTX8xSttlRRBEBfD6tFY9E8jSTXTRMcRFuHUoufv
G4rNDfVzCBNwCCottRVxpe4acYyA6U3KYMOTSEWugx8h";
```

Récupérer les données suivantes :

- le nombre créé par **tous** les chiffres de ce string (sous la forme d'un Int32)
- le string créé entre le **caractère à l'index 8** et l'apparition de la **sous-chaîne string "uHU"**
- le string **en majuscule** créé par **toutes** les lettres et vérifier s'il est égal à "sSDzoTeVhZhEJfHanVTXxSttlRRBEBfDtFYEjSTXTRMcRFuHUoufvGrNDfVzCBNwCCottRVxpeacYyAUKYMOTSEWugxh" de **3 manières différentes** (==, Equals(), et Equals() avec l'insensibilité à la casse)

Correction

```
using System;
namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            string sentence = "sSDzoTeVhZ0hEJfHanVTX8xSttlRRBEBfD6tFY9E8jSTX
            TRMcRFuHUoufvG4rNDfVzCBNwCCottRVxpe4acYyA6U3KYMOTSEWugx8h";
            nbr(sentence);
            extract1(sentence);// 869844638
            extract2(sentence);// False False True
        }

        static void nbr(string str)
        {
            string nbr = "";
            foreach (var c in str)
            {
                if (char.IsNumber(c))
                {
                    nbr += c;
                }
            }
            int nbr1 = Int32.Parse(nbr);
            Console.WriteLine($"{nbr1} est créé par tous les chiffres
            du string {str} (sous la forme d'un Int32)");
        }

        static void extract1(string str)
        {
            int StartIndex = 8;
            int EndIndex = str.IndexOf("uHU");
            string extract = str.Substring(StartIndex, EndIndex-StartIndex);
            Console.WriteLine(extract);
        }

        static void extract2(string str)
        {
            string str1 = "";
            foreach (var c in str)
            {
                if  (char.IsNumber(c)==false)
                {
                    str1 += c;
                }
            }
            string str2 = str1.ToUpper();
```

```
            Console.WriteLine($"{str2} est le string en majuscule créé
            par toutes les lettres du string {str} (sous la forme d'un Int32)");
            Console.WriteLine(str1==str2);
            Console.WriteLine(str1.Equals(str2));
            Console.WriteLine(str1.Equals(str2,StringComparison.
            OrdinalIgnoreCase));
        }
    }
}
```

Classe

La notion de **classe** est à la base de la programmation orientée objet. Elle définit **un type d'objet**. Ce type définit les variables (créées pour chaque objet et appelées « attributs ») et des fonctions (appelées « méthodes »).

Ces attributs et méthodes sont les « membres » de la classe. Ils peuvent avoir chacun un niveau de **protection** différent : **public** (autorisé à tous), **protected** (autorisé depuis la classe seulement), **private** (autorisé depuis la classe et ses sous-classes seulement) et **internal**(autorisé depuis l'assembly seulement).

Exemple de déclaration d'une classe :

```
    public class MaClasse
    {
        //attributs

        public MaMéthode(mes arguments)//constructeur
        {
            //this. mes attributs = mes arguments
        }
     }
```

Héritage et classe abstraite

L'héritage est l'un des mécanismes fondamentaux de la programmation orientée objet. C'est un mécanisme qui consiste à **définir une classe à partir d'une classe existante**. Une classe, héritant d'une autre, possède les caractéristiques de la classe initiale et peut définir ses propres éléments.

Lorsque toutes les classes filles doivent comporter obligatoirement une méthode, on utilise une **classe abstraite**. Elle possède au moins une méthode abstraite ou une propriété abstraite, c'est-à-dire ne comportant aucune implémentation (pas de code). Une telle classe ne peut être instanciée avec l'opérateur ***new***. Il faut utiliser une sous-classe implémentant les méthodes abstraites de la classe de base.

Exemple :

```
public abstract class Animal
    {
        public int nbPattes;//mes attributs
        public string sound;

        public Animal(int nbPattes, string sound)//constructeur
        {
```

```
            this.nbPattes = nbPattes;
            this.sound = sound;
        }

        public abstract void AfficherAnimal();//méthodes abstraites
    }
    public class Chien : Animal
    {
        public string name;
        public Chien(string name) : base(4,"Wouf")//reprends les arguments de
                                                  de la classe parente
        {
            this.name = name;
        }

        public override void AfficherAnimal()//implémentation de la méthode
        {
            Console.WriteLine("Je m'appelle "+this.name+ " et je suis un " + this.nbPattes + " et je f
        }
    }
    static void Main(string[] args)
    {
        Chien doggo = new Chien("Scooby");
        doggo.AfficherAnimal();
    }
```

Exercice

Créer un héritage avec comme **classe mère Véhicule** (avec comme **attribut nbWheels**), et **deux classes filles Voiture et Moto** (avec comme attributs myBrand et myModel),puis créer un objet pour chacune de ces deux classes, de plus implémenter une méthode qui présente le véhicule.

Correction

```
using System;
namespace ConsoleApp1
{
    class Program
    {
        public abstract class Vehicule
        {
            public int nbWheels;

            public Vehicule(int nbWheels)
            {
                this.nbWheels = nbWheels;
            }

            public abstract void AfficherVehicule();
        }
        public class Voiture : Vehicule
        {
            public string myBrand;
            public string myModel;
```

```
            public Voiture(string Brand,string Model):base(4)
            {
                this.myModel = Model;
                this.myBrand = Brand;
            }

            public override void AfficherVehicule()
            {
                Console.WriteLine(this.myBrand + " " +this.myModel
                + " et j'ai "+this.nbWheels+" roues");
            }
        }

        public class Moto : Vehicule
        {
            public string myBrand;
            public string myModel;

            public Moto(string Brand, string Model) : base(2)
            {
                this.myModel = Model;
                this.myBrand = Brand;
            }

            public override void AfficherVehicule()
            {
                Console.WriteLine(this.myBrand + " " + this.myModel
                + " et j'ai " + this.nbWheels + " roues");
            }
        }

        static void Main(string[] args)
        {
            Voiture maVoiture = new Voiture("Ford", "Fiesta");
            maVoiture.AfficherVehicule();
            Moto maMoto = new Moto("Ducati", "Streetfighter");
            maMoto.AfficherVehicule();
        }
    }
}
Output :
Ford Fiesta et j'ai 4 roues
Ducati Streetfighter et j'ai 2 roues
```

Record

C# possède un **type d'objets** dont le but est de fournir **une syntaxe simple** pour déclarer des objets de type référence contenant des propriétés. Ces objets peuvent être définis en utilisant le mot-clé **`record`**. Un objet `record` est compilé sous forme d'une classe. La différence entre le 2 objets est que le compilateur **génère implicitement davantage de fonctions** pour l'objet *record* comme `ToString()`, `PrintMembers()`, `Equals()`, `GetHashCode()`...

Exemple :

```
public record Car (string Brand, string Model);
                    =
public record Car
{
    public  string Brand { get; set; }
    public  string Model { get; set; }
}
```

ATTENTION : Le type `record` n'est utilisable que dans la version .NET 5.0.

Exercice

Reprendre l'exercice précédent sur l'héritage et les classes abstraites, en **enlevant** l'implémentation de la méthode abstraite. Modifier le code pour **remplacer les classes** par **des** **`record`** et créer des objets, puis les afficher
Indice : *override la fonction ToString()*

Correction

```
using System;
namespace ConsoleApp2
{
    class Program
    {
        public record Vehicule (int nbWheels);
        public record Voiture (string brand, string model): Vehicule(4)
        {
            public override string ToString()
            {
            return base.ToString();
            }
        }
        public record Moto (string brand, string model): Vehicule(2)
        {
            public override string ToString()
            {
                return base.ToString();
            }
        }
        static void Main(string[] args)
        {
            Voiture maVoiture = new Voiture ( "Ford", "Fiesta" );
            Console.WriteLine(maVoiture);
            Moto maMoto = new Moto("Ducati", "Streetfighter");
            Console.WriteLine(maMoto);
        }
    }
}
Output :
Vehicule { nbWheels = 4, brand = Ford, model = Fiesta }
Vehicule { nbWheels = 2, brand = Ducati, model = Streetfighter }
```

Types référence et types valeur

Le langage de programmation C# compte 2 types de variable :

- **Les variables de type valeur** stockent des données
- **Les variables de type référence** stockent les références aux données

Les types valeur

Lorsqu'une variable est déclarée à l'aide d'un des **types de données** intégrées de base ou d'une structure définie par l'utilisateur, il s'agit d'un type valeur (exception pour la classe string qui est un type référence). Les variables de type valeur **contiennent directement leurs valeurs**. Cela signifie qu'il contient la valeur de la variable directement sur son propre espace mémoire et les types de valeur utiliseront la mémoire de **pile** pour stocker les valeurs des variables.

Exemple :

```
class Program
{
    static void Double(int a, int b)
    {
        a = a + a;
        b = b + b;
        Console.WriteLine(a + " " + b);
    }
    static void Main(string[] args)
    {
        int num1 = 10;
        int num2 = 20;
        Console.WriteLine(num1 + " " + num2);
        Double(num1, num2);
        Console.WriteLine(num1 + " " + num2);
    }
}
Output = 10 20
         20 40
         10 20
```

Les types référence

Les **types de référence** contiendront **un pointeur qui pointe vers un autre emplacement mémoire** contenant les données. Une référence à l'emplacement mémoire de l'objet est affectée à la variable lors qu'une instance de l'objet est créé à l'aide de l'opérateur new ou qu'un objet créé ailleurs à l'aide de new lui est assigné. Les **types de référence** ne **stockent pas la valeur de variable directement dans sa mémoire à la place**, ils stockent l'adresse mémoire de la valeur de variable pour indiquer où la valeur est stockée.

Exemple :

```
    class Person
    {
        public int age;
```

```
    }
    static void Double(Person a, Person b)
    {
        a.age = a.age + a.age;
        b.age = b.age + b.age;
        Console.WriteLine(a.age + " " + b.age);
    }
    static void Main(string[] args)
    {
        Person p1 = new Person();
        Person p2 = new Person();
        p1.age = 10;
        p2.age = 20;
        Console.WriteLine(p1.age + " " + p2.age);
        Double(p1, p2);
        Console.WriteLine(p1.age + " " + p2.age);
    }
Output: 10 20
        20 40
        20 40 // changement de valeur à l'adresse des personnes
```

Exercice

Créer une méthode qui **permute deux chaînes de caractères** par leurs **références**.

Correction

```
using System;
namespace ConsoleApp2
{
    class Program
    {
        static void SwapStrings(ref string x, ref string y)
        {
            string temp = x;
            x = y;
            y = temp;
        }
        static void Main(string[] args)
        {
            string str1 = "Je suis str1";
            string str2 = "Je suis str2";
            Console.WriteLine("Before swapping: str1={0} str2={1}", str1, str2);
            SwapStrings(ref str1, ref str2);
            Console.WriteLine("After swapping: str1={0} str2={1}", str1, str2);
        }
    }
}
Output :
Before swapping: str1=Je suis str1 str2=Je suis str2
After swapping: str1=Je suis str2 str2=Je suis str1
```

Interface

Nous avons vu précédemment les classes abstraites, on peut dire que **les interfaces ressemblent beaucoup aux classes abstraites** et partagent le fait de ne pas pouvoir être instanciés. Cependant, les interfaces se situent encore **plus au niveau conceptuel** que les classes abstraites, car **aucun corps de méthode n'est autorisé**. Les interfaces sont utilisées avec des classes pour définir ce que l'on appelle un contrat : la classe qui en hérite est tenue d'implémenter toutes les méthodes et propriétés de celle-ci.

En C#, une interface peut être définie à l'aide du mot clé **`interface`** . Les interfaces peuvent contenir des méthodes, des propriétés, des indexeurs et des événements en tant que membres.

Exemple :

```
interface MyInterface
{
// Toutes les méthodes sont publiques par défaut et n'ont pas de corps
    void method1();
    void method2();
}
class  MyClass : MyInterface
{
    void method1()
    {
        // code
    }
    void method2()
    {
        // code
    }
}
```

Comme les classes, les interfaces **peuvent hériter** d'autres interfaces :

```
interface A{}
interface B : A{}
class C : B
{
//Implémenter tous les membres de A et B.
}
```

La différence entre une classe abstraite et l'interface réside dans **le fait de vouloir implémenter ou non les méthodes à toutes les classes filles**, si oui il vaut mieux utiliser une classe abstraite, sinon il vaut mieux utiliser une interface pour l'implémenter dans les classes que l'on souhaite.

Exercice

Reprendre votre code de l'exercice de la partie **Classe** et créer une interface qui présente les véhicules.

Correction

```
using System;

namespace ConsoleApp1
{
    class Program
    {
        interface IPresentation
        {
            public void AfficherVehicule();
        }
        public class Vehicule
        {
            public int nbWheels;

            public Vehicule(int nbWheels)
            {
                this.nbWheels = nbWheels;
            }
        }
        public class Voiture : Vehicule, IPresentation
        {
            public void AfficherVehicule()
            {
                Console.WriteLine(this.myBrand + " " + this.myModel
                + " et j'ai " + this.nbWheels + " roues");
            }


            public string myBrand;
            public string myModel;

            public Voiture(string Brand,string Model):base(4)
            {
                this.myModel = Model;
                this.myBrand = Brand;
            }
        }

        public class Moto : Vehicule,IPresentation
        {
            public string myBrand;
            public string myModel;

            public Moto(string Brand, string Model) : base(2)
            {
                this.myModel = Model;
                this.myBrand = Brand;
            }

            public void AfficherVehicule()
            {
                Console.WriteLine(this.myBrand + " " + this.myModel
                + " et j'ai " + this.nbWheels + " roues");
            }
        }

        static void Main(string[] args)
        {
            Voiture maVoiture = new Voiture("Ford", "Fiesta");
            maVoiture.AfficherVehicule();
```

```
            Moto maMoto = new Moto("Ducati", "Streetfighter");
            maMoto.AfficherVehicule();
        }
    }
}
Output :
Ford Fiesta et j'ai 4 roues
Ducati Streetfighter et j'ai 2 roues
```

Sources

- https://waytolearnx.com/category/csharp
- https://docs.microsoft.com/fr-fr/dotnet/csharp/
- https://fr.wikibooks.org/wiki/Programmation_C_sharp
- https://www.c-sharpcorner.com/