LANGAGE C# - Développement de classes C#

Définition des classes

```
public class MaClasse
{
    //Attributs
    int monAttribut;
    String monDeuxiemeAttribut;

    //Constructeur par défaut
    public MaClasse()
    {
        monAttribut = 1;
        monDeuxiemeAttribut = "Je suis un attribut";
    }

    //Constructeur spécialisé (avec paramètres)
    public MaClasse(int atttributInt, String attributString)
    {
        monAttribut = atttributInt;
        monDeuxiemeAttribut = attributString;
    }

    //Properties
    public int MonAttribut
    {
        get { return monAttribut; }
        set { monAttribut = value; }
    }

    public String MonDeuxiemeAttribut
    {
        get { return monDeuxiemeAttribut; }
        set { monDeuxiemeAttribut = value; }
    }

    //Methodes
    private void IncrementerMonAttributInt()
    {
        MonAttribut += 1;
    }

    public String ConcatenationDeLAttributString(String concatenation)
    {
        return MonDeuxiemeAttribut + " " + concatenation;
    }
}
```

Une classe en C#, comme dans beaucoup d'autres langages, permet de créer des structures. Elle va nous permettre de créer des Objets, qui contient des informations et des mécanismes du sujet défini par la classe.

Une classe sera tout d'abord défini par son nom, prenant une majuscule. Dans l'exemple ci-dessus, notre classe s'appelle MaClasse et on créera des objets de type MaClasse à partir de celle-ci.

LANGAGE C# - Développement de classes C#

Dans une classe, nous aurons 4 parties :

- Les attributs
- Les constructeurs
- Les properties
- Les méthodes

Les attributs

Aussi appelés champs, ce sont les variables qui représentent l'objet que nous allons créer. Par exemple, pour une personne, les attributs seraient le nom et le prénom.

Les constructeurs

Le constructeur dans une classe permettra de créer un objet avec un état bien défini de ses attributs. Dans la forme, un constructeur peut être appréhendé à une fonction. Dans le cas où aucun paramètre n'est renseigné, le constructeur est appelé « constructeur par défaut ». Il sera la référence pour l'instanciation des attributs.
Un constructeur avec des paramètres est appelé « constructeur spécialisé » et permet de définir au moment de la création de l'objet, un ou plusieurs attributs.

Les properties

Les properties permettent de pouvoir manipuler les attributs en dehors de la classe après son instanciation. Les properties seront utile si une autre classe doit changer une valeur d'un objet par lui-même. On les appelle aussi dans d'autres langages les « getters et setters ».

Les méthodes

Il s'agit de fonctions et de procédure. Elles ne peuvent affecter que l'objet qui lui sont liées. On peut accéder aux attributs de l'objet depuis les méthodes.
Chaque élément de la classe aura une visibilité par rapport aux autres composants de l'application. La visibilité sera représentée par les mots clés suivants :

- PUBLIC
- PRIVATE
- PROTECTED

Le Mot clé Public

Lorsqu'un élément est public, il peut être utilisé dans n'importe autre classe dès que l'objet en question est instancié.

Le Mot clé Private

Lorsqu'un élément est private, il ne peut être utilisé qu'à l'intérieur de la classe (et dont de l'objet) à laquelle il appartient.

Le Mot clé Protected

Un élément Protected est un élément private, mais l'élément sera aussi accessible par les classes filles de la classe de l'élément.

Création et utilisation d'un objet

```
MaClasse objetDeMaClasseParDefaut = new MaClasse();
MaClasse objetDeMaClasseSpecialise = new MaClasse(1, "Le Python c'est cool");

objetDeMaClasseParDefaut.MonAttribut = 2;
objetDeMaClasseSpecialise.MonDeuxiemeAttribut = "Le C# c'est mieux";

objetDeMaClasseParDefaut.IncrementerMonAttributInt(); //Génère une erreur car la
fonction est private

Console.WriteLine(objetDeMaClasseSpecialise.ConcatenationDeLAttributString("et c'est
beau !"));
```

Pour créer un objet par rapport à une classe donnée, on doit d'abord déclarer une variable qui sera du type de la classe. Puis on affectera une valeur à cette variable grâce au mot clé new suivi d'un des constructeurs de la classe.

Pour utiliser une fonction ou une méthode que l'objet peut utiliser, on utilisera un point juste derrière le nom de la variable contenant l'objet de la classe.
Ici on peut voir que la property MonAttribut a été utilisée et qu'on a affecté la valeur de 2 à l'attribut monAttribut de l'objet objetDeMaClasseParDefaut.
Même constat avec l'objet objetDeMaClasseSpecialise.

En ce qui concerne l'appel des méthodes, dans l'exemple, vous pouvez voir que ConcatenationDeLAttributString est souligné en rouge. Si on regarde bien dans la définition de la classe, la méthode est précédée du mot clé Private. Elle ne peut donc pas être utilisée dans une classe extérieure à la sienne.
L'autre méthode est quant à elle en Public et donc peut être utilisée.

Exercice 1 : Un premier pas avec les objets

Créer une classe Personne avec pour attributs le nom, le prénom et l'âge avec un constructeur spécialisé.
Créer les properties des attributs.
Créer une méthode qui renvoie un booléen pour savoir si la personne est majeure ou non.
Créer une méthode qui affiche la chaine de caractères suivante :
➔ « Le personne s'appelle -nom- - prenom- et elle a -age- ans. »
Créer ensuite 5 objets Personne dans le Main

Insérer les dans un tableau et ensuite compter le nombre de personnes majeures.
Afficher « Parmi les 5 personnes, X sont majeures »

Les interfaces

```
public interface IMonInterface
{
    void MethodeInterface();
}
class FooClass : IMonInterface
{
    public void MethodeInterface()
    {
        throw new NotImplementedException();
    }
}
```

Une interface permet de définir un contrat pour une classe. Quand une classe hérite d'une interface, elle est obligée d'implémenter les méthodes que l'interface définie.
On peut alors déclarer une variable du type de l'Interface et lui associé une classe qui hérite de l'interface.

```
IMonInterface objetQuiImplementeInterface = new FooClass();
objetQuiImplementeInterface.MethodeInterface();
```

Cependant, on ne pourra appeler que les méthodes qui sont définis dans l'interface, mais nous sommes sûrs que l'objet pourra les exécuter.

Les Listes et les Dictionnaires

Les objets qui suivent ne peut être utilisé que si on rajoute la bibliothèque suivante : System.Collections.Generic

Les listes

Lorsqu'on manipule un ensemble d'objets, nous sommes tentés d'utiliser un tableau. Le tableau est robuste et permet d'accéder à la donnée lorsque l'on sait qu'elle existe et où elle se trouve. L'un des principaux défauts d'un tableau est que le tableau est fixe. Un tableau déclaré avec 10 éléments d'un certain type aura toujours une taille de 10 éléments, qu'importe les évènements qui se produisent sur ce tableau.

Afin d'avoir de pouvoir avec des tableaux dynamiques, nous avons un objet lié au C#.

```
List<MaClasse> listeObjetsMaClasse = new List<MaClasse>();
```

Nous avons ici une liste qui peut contenir des objets MaClasse et uniquement des objets de cette classe. Une liste est dynamique, nous n'avons pas besoin de préciser la capacité.

Voici une liste non-exhaustive de ce qu'on peut faire avec une liste :

```
//On ajoute un objet à la fin de la liste
listeObjetsMaClasse.Add(new MaClasse());
listeObjetsMaClasse.Add(new MaClasse(1, "Le C# c'est trop bien !"));

//On supprime un objet à un endroit spécifique dans la liste
//Les objets sont ensuite replacés dans la liste pour ne pas laisser de "trous"
listeObjetsMaClasse.RemoveAt(1);

//On parcourt les objets de la liste
foreach (MaClasse objet in listeObjetsMaClasse)
{
	Console.WriteLine(objet.MonDeuxiemeAttribut);
}
```

Les dictionnaires

Les dictionnaires en C# (Dictionary) sont un ensemble de clé et de valeurs. Une clé ne peut être dupliquée mais les valeurs peuvent l'être.

Voici la déclaration d'un dictionnaire :

```
Dictionary<String, String> dict = new Dictionary<string, string>();
```

Par définition, la clé est toujours le premier paramètre renseigné. La clé et la valeur peuvent être de type complètement différent, et il n'y a pas de type prédéfini pour les clés.

```
//On ajoute des ensembles de clés/valeurs
dict.Add("clé1", "valeur");
dict.Add("clé2", "valeur2");
dict.Add("clé3", "valeur");

//On supprime un ensemble via la clé
dict.Remove("clé1");

//On parcourt l'ensemble des clés et des valeurs
foreach (KeyValuePair<String, String> kvp in dict)
{
    Console.WriteLine("{0}, {1}", kvp.Key, kvp.Value);
}
```

LANGAGE C# - Développement de classes C#

Exercice 2 : Interfaces, Listes et Dictionnaires

Créer une classe Footballeur avec un nom, un prénom, un âge et une position.
Pour la classe Footballeur, on aura un constructeur qui prendra 4 paramètres (un pour chaque attribut) et un constructeur qui prendra un Dictionnaire.
Créer une classe Tennisman avec un nom, un prénom, un âge et s'il est droitier ou non.
Pour la classe Footballeur, on aura un constructeur qui prendra 4 paramètres (un pour chaque attribut) et un constructeur qui prendra un Dictionnaire.
Créer une interface Competition dont les classe Footballeur et Tennisman vont hériter. Elle aura une fonction qui permettra d'afficher dans quelle compétition il peut jouer. Pour les Footballeurs, on affichera dans quelle ligue il peut jouer : il pourra jouer en Ligue Mineur s'il a en dessous de 16 ans et en Ligue Majeure s'il a 16 ans ou plus. Pour les Tennisman, on affichera s'il peut participer à Roland Garros s'il a 18 ans ou plus.
Créer une liste qui contient 3 Footballeurs et 3 Tennisman.
Parcourez la liste et afficher la compétition de chaque objet.