

COURS PROGRAMMATION ORIENTÉ OBJET C#(Csharp)

KUTEMA M'BA Roland & OGBONE INOUSSA Chérifa

March 6, 2024

Plan

- 1 Introduction
- 2 Classes & Objets
- 3 Héritage
- 4 Polymorphisme
- 5 Abstraction
- 6 Interface

Historique

C# est un langage de programmation orienté objet développé par Microsoft en 2002 qui s'exécute sur .NET Framework. Il est proche d'autres langages populaires comme C et Java.

Les Classes & Les Objets

Classe

Une classe est un modèle qui sert à construire des objets . Elle est composée d'**attributs**(données) et les **méthodes** (traitements). Elle est créée en utilisant le mot clé **class** suivit du nom de la classe avec des accolades

Objet

Un objet est une instance d'une classe. Il est crée à partir du mot clé **new** suivi du constructeur

Constructeur

Un Constructeur est une methode spéciale qui est utilisée pour instancier une classe .

Elle est appelé quand un objet d'une classe est crée. Elle peut être utilisé pour initialiser les valeurs des attributs.

Encapsulation & Propriétés

Encapsulation

C'est une manière organisée et structurée de concevoir les objets en regroupant les données(attributs) et les méthodes au sein d'une classe,tout en limitant l'accès direct aux données internes pour garantir une gestion contrôlée et cohérente de l'objet. Pour y parvenir, vous devez:


- Déclarer les attributs en tant que **private**
- Fournir les méthodes **get** et **set** en tant que **public**, par le biais de propriétés, pour accéder à la valeur d'un attribut.

Propriété

Une propriété est comme une combinaison d'une variable et d'une méthode, et elle a deux méthodes: une méthode **get** et une méthode **set**.

Encapsulation & Propriétés

Exemple



```
1 //methode 1
2 class Person
3 {
4     private string name; // field
5
6     public string Name // property
7     {
8         get { return name; } // get method
9         set { name = value; } // set method
10    }
11 }
12 // methode 2
13 class Person
14 {
15     public string Name // property
16     { get; set; }
17 }
```

Encapsulation & Propriétés

Exemple



```
1  class Program
2  {
3      static void Main(string[] args)
4      {
5          Person myObj = new Person();
6          myObj.Name = "Tamba";
7          Console.WriteLine(myObj.Name);
8      }
9  }
```


Définition

C'est un mécanisme où une classe (sous classe ou classe dérivée) hérite des propriétés et méthodes d'une autre classe (super classe ou la classe de base). Il nous permet de réutiliser de code et une organisation hiérarchique. Pour hériter d'une classe, on utilise le symbole :

Note: L'héritage multiple n'est pas pris en charge en C# par classe.

Exemple



```
1  class Animal
2  {
3      //
4  }
5
6  class Chat : Animal
7  {
8      //
9  }
```

En Csharp , Nous avons deux concept:

Principe

- Au moment de l'exécution, les objets d'une classe dérivée peuvent être traités comme des objets d'une classe de base dans les paramètres de méthode et les collections ou les tableaux.
- Les classes de base peuvent définir et implémenter des méthodes virtuelles, et les classes dérivées peuvent les substituer, ce qui signifie qu'elles fournissent leur propre définition et implémentation.

Note:

Une classe dérivée ne peut substituer un membre de classe de base que si le membre de classe de base est déclaré comme étant **virtual** ou **abstract**. Le membre dérivé doit utiliser le mot clé **override** pour indiquer explicitement que la méthode est conçue pour participer à l'appel virtuel.

TP

- Créer une classe Animal puis définir une méthode qui affiche le cri de l'animal.
- Créer deux classes Pig et Dog qui héritent de la classe Animal puis définir la méthode qui affiche le cri de chaque animal crée.

Abstraction

Exemple



```
1  public abstract class Animal
2  {
3      public abstract void SeDeplacer();
4  }
5
```

Exemple

Interface

Une interface est une classe complètement abstraite, qui ne peut contenir que des méthodes et des propriétés abstraites. Par convention, une interface commence par la lettre "I" et par défaut les membres d'une interface dont **abstract** et **public**. Pour utiliser une interface, on utilise le symbole : (tout comme avec l'héritage).

Note: Les interfaces peuvent contenir des propriétés et des méthodes, mais pas les champs.

Example

```
public interface IAnimal
{
    void animalSound();
}

class Pig : IAnimal
{
    public void animalSound()
    {
        Console.WriteLine("The pig says: wee wee");
    }
}
```

conclusion