

Programmation orientée objet avec C#

Programmation .Net avec C#

Introduction

- Un objet ?
 - Instance d'une classe.
 - Contient
 - Des attributs
 - Des méthodes
 - Interagit avec d'autres objets.
- Principes de la POO :
 - Encapsulation
 - Héritage
 - Polymorphisme

Création d'une classe

```
6  namespace ConsoleApplication
7  {
8      public class TextClass
9      {
10         #region Attributs
11         #endregion
12
13         Evenements
14
15
16         Methodes
17
18     }
19 }
20
```

Définition d'une classe

Création d'une classe

- Définition d'attributs et méthodes.
 - public : accessible partout.
 - private : accessible uniquement dans les méthodes de la classe.
 - internal : accessible dans la classe et dans toutes les méthodes du même assembly.
 - protected : uniquement pour une utilisation dans les membres d'une classe.
Accessible dans la classe et dans les classes dérivées.
 - protected internal : l'union de protected et internal.

Création d'une classe

```
public class Personne
{
    int age;
    string prenom;
    public string Nom;
}
```

Déclaration des attributs

Création d'une classe

Déclaration des propriétés :

```
private int age;  
  
public int Age  
{  
    get { return age; }  
    set { age = value; }  
}
```

```
public int Age { get; set; }
```


Création d'une classe

Déclaration des méthodes.

- Avec ou sans paramètres.
- Avec ou sans type de retour.

```
public class Personne
{
    public void methode1(int x)
    {
        // Traitement
    }

    private int methode2()
    {
        // Traitement
        return 2;
    }
}
```


Surcharge

- Surcharge des méthodes :
 - Des méthodes de même nom peuvent accepter des paramètres différents.
 - Les paramètres passés permettent de savoir quelle méthode à utiliser.
- Utilisation et surcharge des constructeurs.

Surcharge

```
public class Personne
{
    public void methode1()
    {
        // Traitement
    }

    public void methode1(int x)
    {
        // Traitement
    }

    public void methode1(float x)
    {
        // Traitement
    }

    public void methode1(int x, int y)
    {
        // Traitement
    }
}
```

Surcharge des méthodes

Instanciación et initialisation des objets

- Peut se faire en une ligne de code de plusieurs façons :
 - Déclaration mais pas instanciación.
 - Instanciación.
 - Déclaration, instanciación et initialisation en utilisant le constructeur par défaut.
 - Déclaration, instanciación et initialisation en utilisant un autre constructeur.

Héritage

- Les classes dérivées héritent d'une classe de base.
- L'héritage multiple n'est pas supporté.
- On peut hériter selon la portée :
 - Propriétés, méthodes, évènements...
- Les mots clés :
 - sealed : on ne peut pas en hériter.
 - abstract : on ne peut pas instancier cette classe, elle doit être héritée dans une classe dérivées.

Réécriture et surcharge

- Les classes dérivées peuvent réécrire une méthode ou une propriété héritée.
- Les mots clés :
 - virtual : peut être réécrite.
 - abstract : doit être réécrite dans la classe dérivée.
 - sealed : on ne peut pas la réécrire (par défaut)
 - override : remplace la méthode de la classe héritée.

Exemple de l'héritage

```
public abstract class BaseClass
{
    public abstract void Action();

    public virtual void Reecrit()
    {
        Console.WriteLine("Base : Reecrit");
    }

    // Sealed par défaut
    public void Autre()
    {
        Console.WriteLine("Base : Autre");
    }
}
```

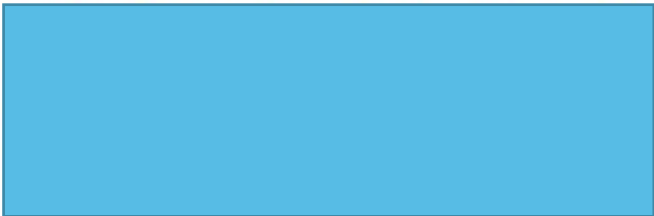
```
public class DerivedClass : BaseClass
{
    public sealed override void Action()
    {
        Console.WriteLine("Derived : Action");
    }

    public override void Reecrit()
    {
        Console.WriteLine("Derived : Reecrit");
    }

    public void Autre(int i)
    {
        Console.WriteLine("Derived : Autre");
    }
}
```

```
DerivedClass dc = new DerivedClass();
```

```
dc.Action();
dc.Reecrit();
dc.Autre();
dc.Autre(10);
```



Mots clés

- base :
 - Se réfère à la classe de base immédiate.
 - Peut uniquement accéder aux membres public ou protected de la classe de base.
 - N'est pas un vrai objet.
 - Ne peut pas être stocké dans une variable.

```
public class TestClass
{
    public override string ToString()
    {
        return base.ToString();
    }
}
```

```
public class TestClass : DerivedClass
{
    public TestClass()
    {
        base.Valeur = 10;
    }
}
```


Mots clés

- `this` :
 - Assure l'accès aux propriétés et méthodes de l'instance en cours.

```
public class TestClass
{
    public int Valeur { get; set; }

    public TestClass(int Valeur)
    {
        this.Valeur = Valeur;
    }
}
```


Interfaces

- Une interface définit les signatures des méthodes, propriétés et événements publics.
- La classe implémentant l'interface, doit définir toutes ses méthodes.
- Comme pour les classes, on peut surcharger les membres d'une interface.
- Une interface peut hériter d'autres interfaces.

Interfaces

```
interface Interface1
{
    void Methode();
    void Methode2(int a);
}
```

```
public class TestClass : Interface1
{
    public void Methode()
    {
    }

    public void Methode2(int a)
    {
    }
}
```


Static

- Variable « static » :
 - Permet à plusieurs instances de classes de se référer à une seule variables.
- Méthode « static » :
 - Pour partager des procédures sans avoir à déclarer d'instance de classe.
 - On n'accède qu'aux données « static ».
- Classe « static » :
 - Contient uniquement des membres statiques.
 - Ne peut pas être instanciée.
 - Toujours « sealed ».
 - Ne peut pas contenir des constructeurs.

Classes partielles

- Mot clé : « partial ».
- Permettent de diviser une même classe dans plusieurs fichiers.