

Utilisation de constructeurs (Guide de programmation C#)

20/07/2015 • 3 minutes de lecture •     


Dans cet article

[Spécification du langage C#](#)

[Voir aussi](#)

Quand une [classe](#) ou un [struct](#) est créé, son constructeur est appelé. Les constructeurs portent le même nom que la classe ou le struct, et ils initialisent généralement les membres de données du nouvel objet.


Dans l'exemple suivant, une classe nommée `Taxi` est définie en utilisant un constructeur simple. Cette classe est ensuite instanciée à l'aide de l'opérateur [new](#). Le constructeur `Taxi` est appelé par l'opérateur `new` immédiatement après l'allocation de la mémoire pour le nouvel objet.

C#	 Copier
<pre>public class Taxi { public bool IsInitialized; public Taxi() { IsInitialized = true; } } class TestTaxi { static void Main() { Taxi t = new Taxi(); Console.WriteLine(t.IsInitialized); } }</pre>	

Un constructeur qui ne prend pas de paramètres est appelé *constructeur sans paramètre*. Les constructeurs sans paramètre sont appelés chaque fois qu'un objet est instancié à l'aide de l'opérateur `new` et qu'aucun argument n'est fourni à `new`. Pour plus d'informations, consultez [Constructeurs d'instances](#).


À moins d'être [statiques](#), les classes sans constructeur se voient attribuer un constructeur public sans paramètre par le compilateur C#, afin d'activer l'instanciation de classe. Pour plus d'informations, consultez la page [Classes statiques et membres de classes statiques](#).

Vous pouvez empêcher qu'une classe soit instanciée en rendant le constructeur privé, comme suit :


C#	 Copier
<pre>class NLog { // Private Constructor: private NLog() { } public static double e = Math.E; //2.71828... }</pre>	

Pour plus d'informations, consultez [Constructeurs privés](#).

Les constructeurs des types [struct](#) ressemblent aux constructeurs de classe, mais les structs ne peuvent pas contenir de constructeur explicite sans paramètre, car le compilateur en fournit automatiquement un. Ce constructeur initialise chaque champ du struct aux valeurs par défaut. Pour plus d'informations, consultez [Tableau des valeurs par défaut](#). Toutefois, ce constructeur sans paramètre est appelé uniquement si le struct est instancié avec `new`. Par exemple, ce code utilise le constructeur sans paramètre pour [Int32](#). Vous avez ainsi la garantie que l'entier est initialisé :

C#	 Copier
<pre>int i = new int(); Console.WriteLine(i);</pre>	

Toutefois, le code suivant provoque une erreur de compilateur, car il n'utilise pas `new` et il essaie d'utiliser un objet qui n'a pas été initialisé :

C#	 Copier
<pre>int i; Console.WriteLine(i);</pre>	

Les objets basés sur des structs (notamment tous les types numériques intégrés) peuvent également être initialisés ou assignés, puis utilisés, comme dans l'exemple suivant :

C#

 Copier

```
int a = 44; // Initialize the value type...
int b;
b = 33;     // Or assign it before using it.
Console.WriteLine("{0}, {1}", a, b);
```

L'appel du constructeur sans paramètre pour un type valeur n'est donc pas obligatoire.

Les classes et les structs peuvent tous les deux définir des constructeurs qui prennent des paramètres. Les constructeurs qui prennent des paramètres doivent être appelés à l'aide d'une instruction `new` ou d'une instruction [base](#). Les classes et les structs peuvent également définir plusieurs constructeurs, et ni les classes ni les structs ne sont nécessaires pour définir un constructeur sans paramètre. Exemple :

C#

 Copier

```
public class Employee
{
    public int Salary;

    public Employee(int annualSalary)
    {
        Salary = annualSalary;
    }

    public Employee(int weeklySalary, int numberOfWeeks)
    {
        Salary = weeklySalary * numberOfWeeks;
    }
}
```

Cette classe peut être créée à l'aide de l'une ou l'autre des instructions suivantes :

C#

 Copier

```
Employee e1 = new Employee(30000);
Employee e2 = new Employee(500, 52);
```

Un constructeur peut utiliser le mot clé `base` pour appeler le constructeur d'une classe de base. Exemple :

C#

 Copier

```
public class Manager : Employee
{
    public Manager(int annualSalary)
        : base(annualSalary)
```

```
{  
    //Add further instructions here.  
}
```

Dans cet exemple, le constructeur de la classe de base est appelé avant que le bloc du constructeur ne soit exécuté. Le mot clé `base` peut être utilisé avec ou sans paramètres. Tous les paramètres du constructeur peuvent être utilisés comme paramètres pour `base` ou comme partie d'une expression. Pour plus d'informations, consultez [base](#).

Dans une classe dérivée, si un constructeur de classe de base n'est pas appelé explicitement à l'aide du mot clé `base`, le constructeur sans paramètre, s'il existe, est appelé implicitement. Cela signifie que les déclarations de constructeur suivantes sont en fait les mêmes :

C#



```
public Manager(int initialData)  
{  
    //Add further instructions here.  
}
```

C#



```
public Manager(int initialData)  
    : base()  
{  
    //Add further instructions here.  
}
```

Si une classe de base n'offre pas de constructeur sans paramètre, la classe dérivée doit faire un appel explicite à un constructeur de base à l'aide de `base`.


Un constructeur peut appeler un autre constructeur dans le même objet à l'aide du mot clé [this](#). Comme `base`, `this` peut être utilisé avec ou sans paramètres, et tous les paramètres dans le constructeur sont disponibles comme paramètres pour `this` ou comme partie d'une expression. Par exemple, le deuxième constructeur de l'exemple précédent peut être réécrit à l'aide de `this` :

C#



```
public Employee(int weeklySalary, int numberOfWeeks)  
    : this(weeklySalary * numberOfWeeks)  
{  
}
```

L'utilisation du mot clé `this` dans l'exemple précédent provoque l'appel de ce constructeur :

C#	 Copier
<pre>public Employee(int annualSalary) { Salary = annualSalary; }</pre>	

Les constructeurs peuvent être marqués comme [public](#), [private](#), [protected](#), [internal](#), [protected internal](#) ou [private protected](#). Ces modificateurs d'accès définissent la façon dont les utilisateurs de la classe peuvent construire la classe. Pour plus d'informations, consultez la page [Modificateurs d'accès](#).

Un constructeur peut être déclaré statique à l'aide du mot clé [static](#). Les constructeurs statiques sont appelés automatiquement, juste avant que des champs statiques soient accessibles, et ils sont généralement utilisés pour initialiser des membres de classe statique. Pour plus d'informations, consultez [Constructeurs statiques](#).

Spécification du langage C#

Pour plus d'informations, consultez [Constructeurs d'instances](#) et [Constructeurs statiques](#) dans la [spécification du langage C#](#). La spécification du langage est la source de référence pour la syntaxe C# et son utilisation.

Voir aussi

- [Guide de programmation C#](#)
- [Classes et structs](#)
- [Constructeurs](#)
- [Finaliseurs](#)

Cette page est-elle utile ?

 Oui  Non
