

## Les propriétés en C#

Pour mieux comprendre l'utilité des propriétés, voyons un exemple simple traité de différentes manières:

### Cas 1: utilisation d'un champ public (cela ne devrait pas se faire)

```
class Eleve
{
    public int age; // normalement un champ ne devrait pas être public!
}

static void Main(string[] args)
{
    Eleve isidor = new Eleve();
    isidor.age = 12;
    Console.WriteLine("Age: " + isidor.age);
    Console.ReadLine();
}
```

### Cas 2: utilisation de "geteurs" et "seteurs"

L'aspect privé du champ est respecté, mais le code est relativement lourd à l'utilisation. Cette solution est utilisée en Java, par exemple.

```
class Eleve
{
    private int age; // par défaut un champ est "private"
    public void SetAge(int a)
    {
        age = a;
    }
    public int GetAge()
    {
        return age;
    }
}

static void Main(string[] args)
{
    Eleve isidor = new Eleve();
    //isidor.age = 12; // erreur age est inaccessible
    isidor.SetAge(12);
    Console.WriteLine("Age: " + isidor.GetAge());
    Console.ReadLine();
}
```

### Cas 3: utilisation des propriétés

Les propriétés sont une spécificité de C# et présentent beaucoup d'avantages.

```
class Eleve
{
    private int _age; // par défaut un champ est "private"
    public int age // la propriété age permet d'accéder au champ _age
    {
        get { return _age; }
        set { _age = value; }
    }
}

static void Main(string[] args)
{
    Eleve isidor = new Eleve();
    isidor.age = 12; // passe par la propriété age
    Console.WriteLine("Age: " + isidor.age);
    Console.ReadLine();
}
```

Un des avantages d'utiliser les propriétés est de pouvoir changer le code de la propriété. Dans notre cas, on aimerait interdire l'affectation d'un nombre négatif pour l'âge.

L'équivalent du cas 2 devient:

```
class Eleve
{
    private int age; // par défaut un champ est "private"
    public void SetAge(int a)
    {
        if (a < 0)
            age = 0;
        else
            age = a;
    }
    public int GetAge()
    {
        return age;
    }
}

static void Main(string[] args)
{
    Eleve isidor = new Eleve();
    isidor.SetAge(-4);
    Console.WriteLine("Age: " + isidor.GetAge()); // affiche "Age: 0"
    Console.ReadLine();
}
```

Alors qu'en utilisant les propriétés on aurait la situation suivante:

```
class Eleve
{
    private int _age;           // par défaut un champ est "private"

    public int age             // la propriété age permet d'accéder au champ _age
    {
        get { return _age; }
        set
        {
            if (value < 0)     // il est possible de mettre du code ici
                _age = 0;
            else
                _age = value;
        }
    }
}

static void Main(string[] args)
{
    Eleve isidor = new Eleve();
    isidor.age = -4;           // passe par la propriété age
    Console.WriteLine("Age: " + isidor.age); // affiche "Age: 0"
    Console.ReadLine();
}
```

Autre illustration de l'avantage d'utiliser les propriétés, notamment en ce qui concerne l'allègement du code. Si on désire augmenter de 1 l'âge de notre élève isidor, avec get/set (Java-like) on aurait:

```
isidor.SetAge(isidor.GetAge() + 1);
```

alors qu'en utilisant la propriété, le code devient limpide:

```
isidor.age++;
```