Attributs et Méthodes Statiques

Considérons la classe « Car » que nous avons vue dans d'autres documents :

```
class Car
{
    // Le contenu de la classe est sans importance dans ce contexte
}
```

Si je l'utilise dans un programme qui va instancier cette classe à de nombreuses reprises, je vais peut-être devoir me poser la question à un moment : "Mais combien de voiture est-ce que j'ai produit ?".
Pour répondre à cette question, il suffit de compter les objets … pour autant qu'on en aie bien gardé la trace !
Une autre possibilité est d'avoir recours à un attribut statique comme ceci:

```
class Car
{
    public static int nbCars = 0;

    public Car()
    {
        nbCars++;
    }
}
```

Définitions:

> Les attributs (ou variables) statiques sont des variables qui appartiennent à la classe plutôt qu'à une instance de la classe. Cela signifie qu'il n'existe qu'une seule copie de cet attribut, partagée par toutes les instances de la classe.

> Les méthodes statiques sont des fonctions qui appartiennent à la classe et non à une instance particulière. Elles peuvent être appelées sans avoir besoin de créer une instance de la classe.

Dans l'exemple ci-dessus, le compteur **`nbCars`** appartient à la classe **`Car`** et pas à un objet instancié à partir de cette classe. A chaque fois qu'on instancie un objet, le compteur de la classe est incrémenté.
Conséquence ?

```
Car porsche = new Car();
Car ferrari = new Car();
Car skoda = new Car();
Console.WriteLine($"{Car.nbCars}voitures ont été produites");
```

Ce code affichera "3 voitures ont été produites"

A quoi ça sert ?

L'exemple du compteur ci-dessus où une variable est partagée par tous les objets est intéressant, mais il n'a que rarement une véritable utilité.
On va beaucoup plus souvent avoir recours à des méthodes statiques. Ce sont des méthodes qui ne dépendent pas des données d'un objet. Exemple:

```
class Car
{
    public static int nbCars = 0;

    public Car()
    {
        nbCars++;
    }

    public static List<Car> Generate(int nb)
    {
        List<Car> list = new List<Car>();
        for (int i = 0; i < nb; i++)
        {
            list.Add(new Car());
        }
        return list;
    }
}
```

On peut ensuite se préparer un groupe de voitures avec:

```
List<Car> myCars = Car.Generate(10);
```

Méthodes utilitaires

Durant le développement d'une application, il est fréquent que l'on identifie des fonctions dont on a besoin régulièrement à divers endroits du code. Exemple: savoir si un nombre entier est pair ou non:

```
if (val % 2 == 0)
```

Pour ne pas répéter ce code à de multiple endroits, la bonne pratique consiste à le mettre dans une méthode statique d'une classe (dont on choisit le nom):

```
class Helpers
{
    public static bool IsEven(int v)
    {
        return v % 2 == 0;
    }
}
```

Ensuite:

```
if (Helpers.IsEven(val))
```

Ce genre de classe (« Helpers ») n'a souvent aucun attribut ou méthode non statique. On peut alors la déclarer elle-même statique :

```
static class Helpers
{
```

Il existe de nombreuses classes statiques prédéfinies par .NET :

Pour écrire dans la console :

```
namespace System
{
    public static class Console
    {
```

Pour faire des calculs :

```
namespace System
{
    public static class Math
    {
```

Pour interagir avec l'environnement de l'application :

```
namespace System
{
    public static partial class Environment
    {
```

Etc..

Constantes

Toutes les constantes sont par définition statiques.

On va se servir des des classes pour donner plus de sens au noms des constantes.

Exemple:

```
public class Car
{
    public const int MINIMUM_PRICE = 10000;
```

Pour utiliser cette constante, il faut faire :

```
Car.MINIMUM_PRICE
```

On voit bien qu'il s'agit là du prix minimum d'une voiture, pas d'un vélo!
Et finalement, on peut combiner les deux principes pour faire une classe ne contenant que des constantes. Exemple:

```
static class Config
{
    public const int MAX_HEIGHT = 80;      // Of the window
    public const int MAX_WIDTH = 120;

}
```

This

Il vous arrivera probalement, dans le feu de l'action, de perdre de vue ce qui est statique et ce qui ne l'est pas. Vous serez peut-être tenté d'écrire ceci:

```
public class Car
{
    public string color;

    public static void SomeMethod()
    {
        // some code ...

        this.color = "Blue";

        // some code ...
    }
```

Cela ne marchera – ne compilera – pas !
En effet `this` ne peut être appelé que depuis un objet, mais `SomeMethod()` appartient à la classe, donc à aucun objet en particulier.