Organisation du Code

Namespace **> classe > methodes et propriétés**

- Classe
- Methode public static
- Propriété public static
- *Namespace*
- *Using*
- Bonne pratique

Classe

- **Définition** : Une classe est un conteneur qui regroupe du code. Elle est souvent utilisée pour organiser des fonctions et des propriétés ayant un thème commun.
- **Syntaxe** :

```
public class NomDeClasse
{
    // Propriété ici
    // Methode ici
}
```

- Les classes permettent de structurer le code, mais dans ce cours, elles servent seulement à regrouper des méthodes sans entrer dans les concepts de l'orienté objet.

Méthode `public static`

- **Définition** : Une méthode `public static` est une fonction accessible depuis n'importe où dans le code, sans avoir besoin de créer une instance de la classe. Ce principe nous permet de réunir nos fonctions dans des classes.
- **Syntaxe** :

```
public static void NomDeMethode()
{
    // Code ici
}
```

- Les méthodes `static` simplifient l'accès aux fonctions, et leur association avec une classe spécifique permet de regrouper les fonctions par thématique.

Propriété `public static`

- **Définition** : Une propriété `public static` est une variable ou une constante associée à la classe. La propriété est accessible dans toutes les methodes static de la classe.
- **Syntaxe** :

```
    public static int NomDePropriete;
```

- Les propriétés `static` permettent de partager des valeurs entre plusieurs méthodes static de la classe.

Namespace

- **Définition** : Un *namespace* est un espace de noms qui regroupe plusieurs classes, structurant ainsi les fonctionnalités par domaine. Tout les classes d'un même *namespace* peuvent s'appeller entre elles.
- **Syntaxe** :

```
namespace NomDuNamespace
{
    public class Exemple
    {
        // Code ici
    }
}
```

- Les namespaces permettent d’éviter les conflits de noms et d'organiser les classes en catégories logiques.

Using

- **Définition** : Une directive `using` importe un namespace pour utiliser ses classes et méthodes sans avoir à les préfixer.
- **Syntaxe** :

```
using NomDuNamespace;
```

- Cela facilite l'écriture et la lecture du code en rendant accessibles les fonctionnalités des autres namespaces.

Notez qu'en c#, certain namespace sont importé implicitement. C'est le cas du namespace `System` *, qui contient les classes* `Math` *,* `File` *,* `Console` *, etc.*

L'import implicite peut-être désactivé. Voyons un exemple.

Bonne Pratique

- **Un namespace pour tout le projet.** Pour l'instant c'est suffisant. Même pour le TP, nos algorithmes ne sont pas assez gros pour justifier plusieurs namespaces.
- **Un fichier par classe.** L'avantage des classes est que nous pouvons les isoler dans différents fichiers. La bonne pratique est d'avoir un fichier par classe, et de nommer le fichier avec le même nom que la classe.
- **Nommer les classes de façon claire et cohérente.** Les noms de classes doivent représenter un objet clairement. Par exemple, `Playlist`, `Grille`, `VueConsole`, etc.
- **Nommer les méthodes de façon claire et cohérente.** Les noms de méthodes doivent indiquer leur action clairement. Par exemple, `AjouterLivre()` pour ajouter un livre à une liste, et `ValiderISBN()` pour valider un code ISBN.

Bonne Pratique

- **Appliquer le *S*OLID de responsabilité unique**, même avec les classes. Par exemple, une bonne pratique serait de :
  - Réunir dans une classe toutes les méthodes qui portent sur l'affichage.
  - Réunir dans une classe toutes les méthodes qui font de la validation de données.
  - Réunir dans une autre classe les méthodes qui portent sur la logique algorithmique.
- **Éviter les méthodes trop complexes.** Une méthode devrait idéalement faire une seule chose. Si une méthode devient trop complexe, c'est un signe qu'elle pourrait être séparée en plusieurs petites méthodes.