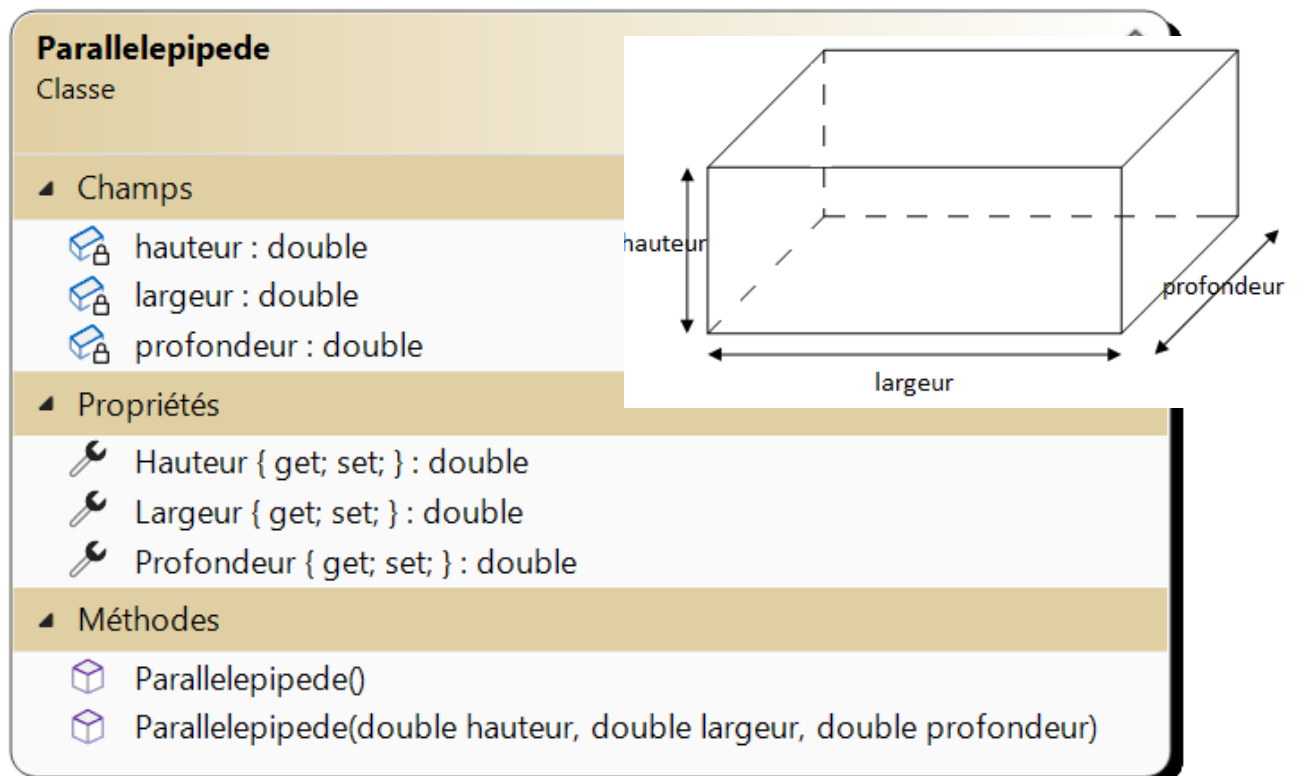


## SEANCE 4 - REVISIONS DES CLASSES

### OBJECTIFS

- Faire une synthèse des notions : classe, champs, propriétés, constructeur, ...
- Savoir résoudre des problèmes fréquents
- Savoir différencier une méthode statique et une méthode d'instance

### PARTIE 1 : SYNTHESE + VOCABULAIRE



1. Définir une (vraie) classe, c'est définir un \_\_\_\_\_

Une classe sert à créer et à manipuler des \_\_\_\_\_

La classe est une définition, son code n'est déclenché qu'à travers les \_\_\_\_\_

2. Une classe est constituée de :

- \_\_\_\_\_ privés (variables internes à l'objet),
- \_\_\_\_\_ (\_\_\_\_\_)
- \_\_\_\_\_ (\_\_\_\_\_)

3. Il y a 4 catégories de méthodes :

1. Celles qui servent à instancier des objets : \_\_\_\_\_,
2. Celles qui sont communes à toutes les classes : \_\_\_\_\_  
\_\_\_\_\_. Dans leur signature : il y a un mot clef spécifique  
\_\_\_\_\_ pour indiquer qu'il s'agit de  
\_\_\_\_\_.
3. Celles qui servent à surcharger les \_\_\_\_\_.
4. Celles qui font des traitements liés au type que l'on définit.

4. Dans l'ordre, idéalement, il faut définir :

1. \_\_\_\_\_
2. \_\_\_\_\_
3. \_\_\_\_\_
4. \_\_\_\_\_
5. \_\_\_\_\_
6. \_\_\_\_\_

5. Niveaux d'accessibilité : internal / private / public

1. Une classe peut être qualifiée de \_\_\_\_\_ ( utilisable dans d'autres projets, nécessaire quand on fait une librairie ) ou \_\_\_\_\_ ( uniquement utilisable au sein du projet )
2. Les méthodes sont généralement \_\_\_\_\_ (utilisable dans d'autres classes du projet et dans d'autres projets si la classe est public Ex : Program) ou \_\_\_\_\_ (utilisable uniquement dans la classe de définition )

Au sein des propriétés, des contrôles sur les valeurs ont été programmés afin que les objets de classe Parallélépipède ne puissent avoir de dimension à 0. Rien ne marche comme prévu dans la classe ci-dessous : expliquez et corrigez.

1. Quand on veut modifier la hauteur d'un objet Parallelepipede :

```
Parallelepipede p = new Parallelepipede();
p.Hauteur = 10;
```

On obtient :

```
System.ArgumentOutOfRangeException : 'Specified argument was out of the range of valid values. Arg_ParamName_Name'
```

2. Quand on veut modifier la profondeur :

```
Parallelepipede p = new Parallelepipede();
p.Profondeur = 10;
```

On obtient :

```
Stack overflow.
Repeat 24100 times:
-----
at TD18.Parallelepipede.set_Profondeur(Double)
-----
at TD18.Program.Main(System.String[])
```

3. Quand on veut modifier la largeur :

```
Parallelepipede p = new Parallelepipede();
p.Largeur = 10;
```

On obtient aucune erreur d'exécution mais la largeur reste à 0.

```
internal class Parallelepipede
{
    private double hauteur;
    private double largeur;
    private double profondeur;

    public double Hauteur
    {
        get
        {
            return this.hauteur;
        }

        set
        {
            if (this.hauteur <= 0)
                throw new ArgumentOutOfRangeException("hauteur doit être > 0");
        }
    }
}
```

```

        this.hauteur = value;
    }
}

public double Largeur
{
    get
    {
        return this.largeur;
    }

    set
    {
        if (value <= 0)
            throw new ArgumentOutOfRangeException("largeur doit être > 0");
    }
}

public double Profondeur
{
    get
    {
        return this.profondeur;
    }

    set
    {
        if (value <= 0)
            throw new ArgumentOutOfRangeException("profondeur doit être > 0");
        this.Profondeur = value;
    }
}

```

4. Ces problèmes résolus, on a défini le constructeur ci-dessous et on est confronté à un nouveau problème : il est possible de créer un Parallélépipède avec toutes ses dimensions égales à 0. Pourquoi ? que faut-il corriger ?

Parallelepipede p = new Parallelepipede(0,0,0);

```

public Parallelepipede (double hauteur, double largeur, double profondeur)
{
    this.hauteur = hauteur;
    this.largeur = largeur;
    this.profondeur = profondeur;
}

```

5. En générant ensuite les substitutions des méthodes usuelles : ToString, Equals et GetHashCode, on a obtenu le code ci-dessous :

```
public override bool Equals(object obj)
{
    return base.Equals(obj);
}

public override int GetHashCode()
{
    return base.GetHashCode();
}

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

Que va afficher le code suivant :

```
class Program
{
    static void Main(string[] args)
    {
        Parallelepipede p1 = new Parallelepipede (120, 200, 40);
        Parallelepipede p2 = new Parallelepipede (120, 200, 40);

        Console.WriteLine(p1);
        Console.WriteLine("p1.Equals(p2) ? " + p1.Equals(p2));
        Console.WriteLine("p1 == p2 ? " + (p1 == p2));
    }
}
```

A quoi faut- il veiller quand on génère Equals et GetHashCode ?

Que faut- il faire pour améliorer : ToString ? Doit-on garder base.ToString(); ?

**PARTIE 3 : METHODES STATIQUES OU D'INSTANCES**

1. Définissez la méthode d'instance (méthode non statique) CalculeVolume. A-t-on besoin de paramètres ?

---



---



---



---

```
class Program
{
    static void Main(string[] args)
    {
        Parallelepipede p1 = new Parallelepipede (120, 200, 40);
        Console.WriteLine("Volume : " + p1.CalculeVolume());
        // CalculeVolume s'appuie sur une instance (=objet) p1.
    }
}
```

**Une méthode d'instance s'appuie sur un objet : elle a donc accès aux données contenues dans l'objet. Elle peut très souvent se passer de paramètres...**

Remarque : On pourrait aussi concevoir une approche plus « ancestrale » .  
Exemple : faire une Classe Volume dans laquelle on stockerait tous les calculs liés aux volumes en général . Dans ce cas, les méthodes sont forcément statiques et on doit passer en paramètre les données. Les traitements sont alors séparés des classes Parallelepipede et Cylindre.

```
public class Volume
{
    public static double CalculeVolumeParallelepipede (Parallelepipede p )
    {
        return p.Hauteur * p.Largeur * p.Profondeur;
    }
    public static double CalculeVolumeCylindre(Cylindre c)
    {
        return c.Rayon * c.Rayon * Math.PI * c.Hauteur;
    }
}
```

Le but de la prog objet est de définir des types bien structurés avec des traitements et opérateurs : le tout bien rangé dans une seule classe. **C'est pourquoi, il faut préférer la méthode d'instance !**

**Parfois, le traitement est fait au sein de la classe. Et il est surchargé pour avoir une méthode d'instance et une méthode statique.**




Ex : Dans la classe Rectangle fournie dans l'espace de nom System.Drawing : il y a 2 méthodes pour déterminer s'il y a une intersection entre 2 rectangles :

- public **static** Rectangle Intersect (Rectangle a, Rectangle b);
- public bool IntersectsWith (Rectangle rect);

<pre>Rectangle r1, r2; r1 = new Rectangle(0, 0, 200, 20); r2 = new Rectangle(0, 0, 100, 20);</pre>	
Utilisation méthode statique	Utilisation méthode d'instance
<pre>Rectangle int = <u>Rectangle</u>.Intersect(<u>r1,r2</u>); if (!int.IsEmpty)     Console.WriteLine("intersection"); else     Console.WriteLine("pas intersection");</pre>	<pre>if (<u>r1</u>.IntersectsWith(<u>r2</u>))     Console.WriteLine("intersection"); else     Console.WriteLine("pas intersection");</pre>

Remarque : on aurait pu aussi choisir une autre conception : ne pas faire de méthode CalculVolume, mais une propriété accessible uniquement en lecture (get) , qui n' a pas de champ associé car c'est une propriété calculée à partir des autres champs.

```
public double Volume
{
    get
    {
        return this.hauteur * this.largeur * this.profondueur;
    }
}
```

Propriétés	
	Largeur { get; set; } : double
	Profondeur { get; set; } : double
	Volume { get; } : double

2. On veut pouvoir comparer 2 parallélépipèdes. Comment et que faut-il faire ?

```
class Program
{
    static void Main(string[] args)
    {
        Parallelepipede p1 = new Parallelepipede (120, 200, 40);
        Parallelepipede p2 = new Parallelepipede (120, 200, 40);

        Console.WriteLine("p1 > p2 ? " + (p1 > p2));
    }
}
```

---

---

---

---

3. On veut pouvoir comparer la hauteur de 2 parallélépipèdes. On peut dans le main directement faire la comparaison. On aimerait délocaliser cette comparaison dans la classe au sein d'une méthode d'instance : EstPlusHaut. Indiquez les modifs à faire dans le main et le code à faire dans la classe.

```
namespace Exo2
{
    class Program
    {
        static void Main(string[] args)
        {
            Parallelepipede p1 = new Parallelepipede (120, 200, 40);
            Parallelepipede p2 = new Parallelepipede (120, 200, 40);

            if (p1.Hauteur > p2.Hauteur)
                Console.WriteLine(p1 + " est plus haut que " + p2);
        }
    }
}
```

---

---

---



4. Puisque c'est possible (bien que pas forcément utile), veuillez donner la signature de la méthode statique PlusHaut.

---

---

5. On veut une méthode pour pouvoir redimensionner un parallélépipède à partir d'un coefficient ! RedimensionneProportionnellement. Indiquez la signature : est ce une méthode d'instance, statique ?

---

---

6. On veut une méthode pour calculer la surface totale d'un parallélépipède ! CalculeSurface. Indiquez la signature : est-ce une méthode d'instance, statique ?

---

---

7. Créez un répertoire Classe-Parallelepipede et un projet TD-Classe-Parallelepipede pour reprendre tout ce qui vient d'être fait par écrit !