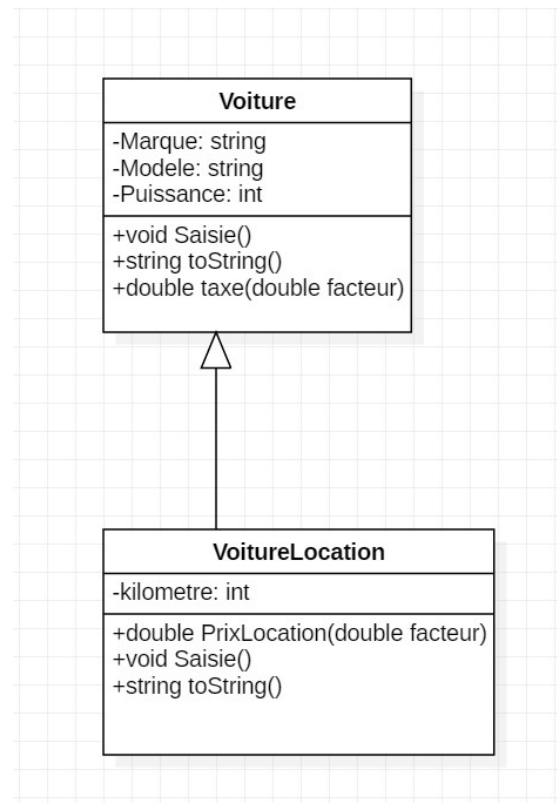


Héritage et Polymorphisme

Aline Ellul

Diagramme de classes du CdC



Classe Voiture

```
class Voiture
{
    string marque;
    string modele;
    int puissance;
    public Voiture(string marque, string modele, int puissance)
    {
        this.marque = marque;
        this.modele = modele;
        this.puissance = puissance;
    }
    public Voiture()
    {}
    public string toString()
    {
        return "\n " + " Marque " + marque + " Modele " + modele + " Puissance " + puissance + "\n";
    }
    public void saisie()
    {
        Console.Write("Marque : ");
        marque = Console.ReadLine();
        Console.Write("Modele : ");
        modele = Console.ReadLine();
        Console.Write("Puissance : ");
        puissance = Convert.ToInt32(Console.ReadLine());
    }
    public double taxe(double facteur)
    {
        return facteur * (double)puissance;
    }
}
```

Programme Principal

```
Voiture v = new Voiture();
v.saisie();
Console.WriteLine("Taxe " + v.taxe(1.2));
Console.WriteLine(v.toString());
```

Exécution du Programme Principal

```
Marque : renault
Modele : clio
Puissance : 12
Taxe 14,4
```

```
Marque renault Modele clio Puissance 12
```

Héritage simple

: pour indiquer l'héritage.

Une classe C# ne peut qu'hériter
d'une seule autre classe

```
class Voiture
{
    string marque;
    string modele;
    int puissance;
    public Voiture(string marque, string modele, int puissance) {...}
    public Voiture()
    {}
    public string toString() {...}
    public void saisie() {...}
    public double taxe(double facteur) {...}
}
```

```
class VoitureLocation : Voiture
{
    double kilometre;

    public VoitureLocation(string marque, string modele, int puissance, double kilometrage) : base(marque, modele, puissance)
    {
        this.kilometre = kilometrage;
    }

    public VoitureLocation()
    {
    }

    public new void saisie()
    {
        base.saisie();
        Console.WriteLine("Kilometrage : ");
        kilometre = double.Parse(Console.ReadLine());
    }

    public double prixLocation(double facteur)
    { return facteur * kilometre; }
}
```

Constructeur Classe Fille

Si classe mère a des attributs private

```
class VoitureLocation : Voiture
{
    double kilometre;

    public VoitureLocation(string marque, string modele, int puissance, double kilometrage) : base(marque, modele, puissance)
    {
        this.kilometre = kilometrage;
    }

    public VoitureLocation()
    {
    }

    public new void saisie()
    {
        base.saisie();
        Console.WriteLine("Kilometrage : ");
        kilometre = double.Parse(Console.ReadLine());
    }

    public double prixLocation(double facteur)
    {
        return facteur * kilometre;
    }
}
```

1. **base** : mot clef pour signifier la classe mère; Ici on récupère du constructeur Voiture les 3 paramètres
2. Obligation de les mentionner dans les paramètres du constructeur de la classe fille

Constructeur Classe Fille

Si classe mère a des attributs protected

```
class Voiture
{
    protected string marque;
    protected string modele;
    protected int puissance;
}

class VoitureLocation : Voiture
{
    double kilometre;

    public VoitureLocation(string marque, string modele, int puissance, double kilometrage)
    {
        this.marque = marque;
        this.modele = modele;
        this.puissance = puissance;
        this.kilometre = kilometrage;
    }
}
```

Les attributs sont alors accessibles. Inutile d'appeler le constructeur par le mot base car appel du constructeur par défaut de la classe Voiture. Si la classe Voiture n'a pas de constructeur par défaut, le constructeur VoitureLocation a la même signature qu'avec les attributs privés.

De quoi hérite t-on de la classe mère ?

- V1 est une variable de référence qui pointe sur un espace mémoire défini par un modèle, une marque, une puissance et un kilométrage
- *Des attributs*
- *Des propriétés*
- Des méthodes

```
VoitureLocation v1 = new VoitureLocation("peugeot", "206", 12,10000);
```

Héritage de méthode simple entre classes mère et fille

- La méthode d'instance `taxe` est définie dans la classe `Voiture`.

```
public double taxe(double facteur)
```

```
VoitureLocation vl = new VoitureLocation("peugeot", "206", 12,10000);  
Console.WriteLine(vl.taxe(1.2));
```

- Toute instance de la classe `VoitureLocation` peut invoquer la méthode `taxe`

Polymorphisme

- *Définition* : Qui peut prendre plusieurs formes
- Il est possible de redéfinir une méthode dans des classes héritant d'une classe de base.
- Par ce mécanisme, une classe qui hérite des méthodes d'une classe de base peut modifier le comportement de certaines méthodes héritées pour être adaptées aux besoins de la classe fille.
- Contrairement à la surcharge, une méthode redéfinie doit non seulement avoir le même nom que la méthode de base, mais **le type et le nombre de paramètres doivent être identiques à ceux de la méthode de base.**

Héritage avec méthodes de même signature

```
class VoitureLocation : Voiture
{
    double kilometre;

    public VoitureLocation(string marque, string modele, int puissance, double kilometrage) : base(marque, modele, puissance)
    {
        this.kilometre = kilometrage;
    }

    public VoitureLocation()
    {
    }

    public new void saisie()
    {
        //base.saisie();
        Console.WriteLine("Kilometrage : ");
        kilometre = double.Parse(Console.ReadLine());
    }

    public double prixLocation(double facteur)
    { return facteur * kilometre; }

    public new string toString()
    { string retour = " kilometrage " + kilometre; ;
      // retour = base.toString() + " kilometrage " + kilometre;
      return retour;
    }
}
```

new : mot clef à utiliser sur une méthode de la classe fille si même signature que la classe mère. La méthode est ainsi redéfinie pour la classe fille sans rapport avec la classe mère

New

sans appel de la méthode de la classe mère

```
class VoitureLocation : Voiture
{
    double kilometre;

    public VoitureLocation(string marque, string modele, int puissance, double kilometrage) : base(marque, modele, puissance)
    {
        this.kilometre = kilometrage;
    }

    public VoitureLocation()
    {
    }

    public new void saisie()
    {
        //base.saisie();
        Console.Write("Kilometrage : ");
        kilometre = double.Parse(Console.ReadLine());
    }

    public double prixLocation(double facteur)
    { return facteur * kilometre; }

    public new string toString()
    { string retour = " kilometrage " + kilometre; ;
      // retour = base.toString() + " kilometrage " + kilometre;
      return retour;
    }
}
```

Programme Principal

```
Voiture v = new Voiture();
v.saisie();
Console.WriteLine("Taxe " + v.tax(1.2));
Console.WriteLine(v.toString());
VoitureLocation vl = new VoitureLocation("peugeot", "206", 12, 10000);

Console.WriteLine(vl.toString());
```

Exécution du Programme Principal

```
Marque : renault
Modele : clio
Puissance : 12
Taxe 14,4

Marque renault Modele clio Puissance 12
Kilometrage 10000
```

New

avec appel de la méthode de la classe mère

```
class VoitureLocation : Voiture
{
    double kilometre;

    public VoitureLocation(string marque, string modele, int puissance, double kilometrage) : base(marque, modele, puissance)
    {
        this.kilometre = kilometrage;
    }

    public VoitureLocation()
    {
    }

    public new void saisie()
    {
        base.saisie();
        Console.Write("Kilometrage : ");
        kilometre = double.Parse(Console.ReadLine());
    }

    public double prixLocation(double facteur)
    { return facteur * kilometre; }

    public new string toString()
    { //string retour = "Kilometrage " + kilometre;
      string retour = base.toString() + " kilometrage " + kilometre;
      return retour;
    }
}
```

Programme Principal

```
Voiture v = new Voiture();
v.saisie();
Console.WriteLine("Taxe " + v.taxe(1.2));
Console.WriteLine(v.toString());
VoitureLocation vl = new VoitureLocation("peugeot", "206", 12, 10000);

Console.WriteLine(vl.toString());
```

Exécution du Programme Principal

```
marque : renault
modele : clio
puissance : 12
Taxe 14,4
```

```
Marque renault Modele clio Puissance 12
```

```
Marque peugeot Modele 206 Puissance 12 kilometrage 10000
```

Attention : new considère le type déclaré de la variable

- La variable vl :

- Est déclarée Voiture et vue par le compilateur comme Voiture
- A l'exécution, vl devient effectivement une VoitureLocation

Programme Principal

```
Voiture vl = new VoitureLocation("peugeot", "206", 12, 10000);  
vl.saisie();  
Console.WriteLine(vl.toString());
```

Exécution du Programme Principal

```
Marque : renault  
Modele : clio  
Puissance : 12  
  
Marque renault Modele clio Puissance 12
```

- Avec new :

- La méthode choisie est celle qui fait référence à la déclaration c-à-d Voiture

Virtual / Override

```
class Voiture
{
    protected string marque;
    protected string modele;
    protected int puissance;

    public int Puissance
    { get { return puissance; } }

    public Voiture(string marque, string modele, int puissance)
    {
        this.marque = marque;
        this.modele = modele;
        this.puissance = puissance;
    }
    public Voiture()
    {}
    public string toString()
    {
        return "\n " + " Marque " + marque + " Modele " + modele + " Puissance " + puissance ;
    }
    public void saisie()...
    public virtual string toString1()
    {
        return "\n " + " Marque " + marque + " Modele " + modele + " Puissance " + puissance;
    }
    public double taxe(double facteur)
    {
        return facteur * (double)puissance;
    }
}
```

```
class VoitureLocation : Voiture
{
    double kilometre;

    public VoitureLocation(string marque, string modele, int puissance, double kilometrage)
    {
        this.marque = marque;
        this.modele = modele;
        this.puissance = puissance;
        this.kilometre = kilometrage;
    }

    public VoitureLocation()
    {
    }
    public new void saisie()
    {
        base.saisie();
        Console.Write("Kilometrage : ");
        kilometre = double.Parse(Console.ReadLine());
    }

    public double prixLocation(double facteur)
    { return facteur * kilometre; }

    public new string toString()
    { //string retour = "Kilometrage " + kilometre;
      string retour = base.toString() + " kilometrage " + kilometre;
      return retour;
    }
    public override string toString1()
    {
        string retour = base.toString() + " kilometrage " + kilometre;
        return retour;
    }
}
```

Virtual / Override

```
class Voiture
{
    protected string marque;
    protected string modele;
    protected int puissance;

    public int Puissance{...}

    public Voiture(string marque, string modele, int puissance){...}
    public Voiture(){...}
    public string toString(){...}
    public void saisie(){...}

    public virtual string toString1(){...}

    public double taxe(double facteur){...}
}
```

```
class VoitureLocation : Voiture
{
    double kilometre;

    public VoitureLocation(string marque, string modele, int puissance, double kilometrage){...}

    public VoitureLocation(){...}
    public new void saisie(){...}
    public double prixLocation(double facteur){...}

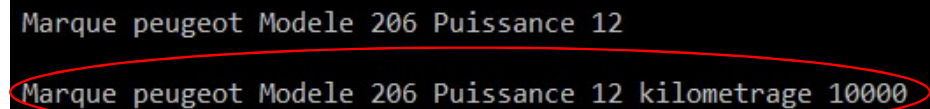
    public new string toString(){...}

    public override string toString1(){...}
}
```

Programme Principal

```
Voiture vl = new VoitureLocation("peugeot", "206", 12,10000);
Console.WriteLine(vl.toString());
Console.WriteLine(vl.toString1());
```

Exécution du Programme Principal



```
Marque peugeot Modele 206 Puissance 12
Marque peugeot Modele 206 Puissance 12 kilometrage 10000
```

Vl est du type VoitureLocation

Sealed et Abstract

- Une classe Sealed ne peut être classe mère
- Feuille terminale de l'arbre d'héritage
- Une classe Abstract ne peut être instanciée
- Peut servir de racine à un arbre d'héritage

```
sealed class VoitureLocation : Voiture
```

```
abstract class Voiture
```