

Aide-mémoire POO – Csharp :

Théorie	Pratique
<p>I) La classe Sert à créer des objets ayant le même état (attributs) et le même comportement (méthodes)</p> <p>Structure d'une classe (squelette d'une classe)</p> <pre> public class nom_classe { //attributs (variables privées) composent un objet //constructeurs (initialiser les attributs) // destructeur (fonction traitée avant destruction de l'objet°) ~ //propriétés (permettent de gérer l'encapsulation) elles sont toujours publiques //Accesseurs (accéder à l'attribut) Permet de renvoyer un attribut // Modifieurs (modifier un attribut) Permet de réinitialiser un attribut //méthodes Fonctions ou procédures privées ou publiques dont l'instance d'une classe (objet) est responsable grâce à la visibilité publique } </pre> <p>Instanciation d'une classe :</p> <p>Le mécanisme de création d'objets s'appelle l'instanciation</p> <p>==> l'instance d'une classe est l'OBJET CREE !!!</p> <p>En Csharp :</p> <p>Classe nom_objet = new Classe(.....,etc) ;</p> <p>Modifier attribut d'un objet</p> <p>nom_objet.Propriété=valeur ; //en Csharp !! nom_objet.SetAttribut(variable) ; // Dans tous les langages !!</p>	<p>I) La classe</p> <p>Ex : <code>public class Voiture</code></p> <pre> { // attributs private string marque=null ; private string modèle=null ; private int puissance ; private bool étatDémarré=false ; private bool étatRoulé=false ; //constructeurs public Voiture() //par défaut {} public Voiture(string _marque, string _m, int puissance) //constructeur surchargé { this.marque=_marque ; // this représente l'objet courant this.modèle=_m ; this.puissance= puissance: //pas d'ambiguité ! } //Destructeur ~Voiture() { Console.WriteLine("je vais me crasher!!!"); } //propriétés public string Marque { set { marque=value ;} // modifieur , rend accessible l'attribut marque en écriture get { return marque ;} // accesseur, permet de retourner l'attribut marque } //ou //accesseur public string getMarque() { return marque ; } //modifieur public void setMarque(string m) { this.marque=m; } } </pre>

Accéder à un attribut

type variable=nom_objet.Propriété ; // en Csharp !!
type variable=nom_objet.GetAttribut() ; // dans
tous les langages !!

Exploiter une méthode publique par un objet

nom_objet.Méthode(.....,.....) ;
type variable=nom_objet.Méthode(..... ,.....) ;

```
public string Modèle
{
    set { modèle=value ;} // modifieur ,
rend accessible l'attribut modèle en
écriture
    get { return modèle ;} // accesseur,
permet de retourner l'attribut modèle
}
```

```
public int Puissance
{
    set { puissance=value ;} // modifieur ,
rend accessible l'attribut puissance en
écriture
    get { return puissance ;} // accesseur,
permet de retourner l'attribut
puissance
}
```

```
public bool EtatDémarré
{
    set { étatDémarré =value ;} //
modifieur ,
rend accessible l'attribut marque en
écriture
    get { return étatDémarré ;} //
accesseur,
permet de retourner l'attribut marque
}
```

//méthodes

```
public bool Démarrer()
{

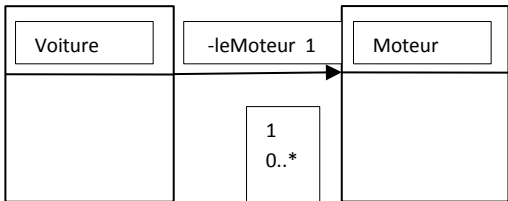
    return étatDémarré=true ;

}
```

```
public bool Rouler()
{
    if (étatDémarré==true)
        étatRoulé=true;

    return étatRoulé;
}
```

```
public override string ToString()
{
```

	<pre> return "La marque est:"+"marque+" Le modèle est:"+"modèle+" La puissance est :"+"puissance+" CV" ; } </pre>
	<p>Ex d'application exploitant la classe Voiture</p> <pre> public static void Main() { int p=8 ; //création d'une instance de type Voiture Voiture v=new Voiture() ; // initialiser les attributs par les modifieurs v.Marque= "Peugeot" ; v.Modèle="307" ; v.Puissance=p ; //Accéder à un attribut par un accesseur string marque=v.GetMarque() ; //ou string marque=v.Marque ; //création d'une Voiture v1 initialisée au départ Voiture v1=new Voiture("Renault","Laguna",10); //Afficher la Voiture v et v1 Console.WriteLine(v //v.ToString()); ; Console.WriteLine(v1.ToString()); ; //Démarrer la voiture v1 et afficher l'étatDémarré Bool Ok=false ; Ok=v1.Démarrer() ; if (Ok==true) Console.WriteLine("La voiture a démarré !"); else Console.WriteLine("La voiture n'a pas démarré !") ; } </pre>
<p>II) Association de classes</p> <p>-Un objet d'une classe A DIALOGUE avec un objet d'une autre classe B par la <u>relation d'association</u></p>  <pre> classDiagram class Voiture class Moteur Voiture "1" -- "0..*" Moteur : -leMoteur </pre>	<p>Ex : <code>public class Voiture</code></p> <pre> { // attributs Private Moteur leMoteur ; //propriétés //constructeurs //méthodes public bool Rouler() { </pre>

⇒ La Relation d'ASSOCIATION est INDISPENSABLE pour assurer l'envoi de MESSAGES entre Objets !!

```
public Voiture
{
    private Moteur leMoteur ; //objet unique
    private ArrayList lesMoteurs ; //plusieurs
    objets
    (0..*)

    //méthodes
    .....
    public bool Démarrer()
    {
        leMoteur=new Moteur() ;
        return leMoteur.Démarrer() ;
    }
}

public Moteur
{
    public bool Démarrer();
    { .....
    }
}
```

// idem au code précédent

```
}

public bool Démarrer()
{
    leMoteur=new Moteur() ;
    étatDémarré=leMoteur.Démarrer() ;
    return étatDémarré ;
}
```

}

public class Moteur

```
{ //attribut
    private bool étatMoteur=false ;
```

```
//constructeur
//méthodes
```

```
public bool Démarrer()
{
    return étatMoteur=true ;
}
```

}

Ex d'application exploitant la classe Voiture

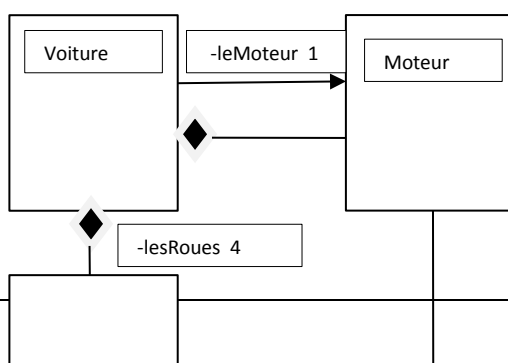
```
public static void Main()
```

```
{
```

//idem au code précédent

```
}
```

III) Composition de classes



Ex : public class Voiture

```
{ // attributs
```

.....

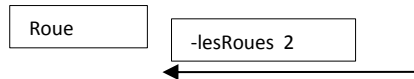
```
Private Moteur leMoteur ;
Private ArrayList lesRoues=new
ArrayList() ;
```

```
//propriétés
```

```
//constructeurs
```

```
public Voiture()
```

```
{ //la composition
```



-La relation de Composition (losange plein)
indique qu'un Objet est composé d'un ou plusieurs objets.
Et que l'ensemble est indissociable !!!!

La composition se fait toujours dans le Constructeur de la classe !!

Dans l'exemple du haut , la Voiture est composé d'un Moteur et de 4 Roues !

⇒ **LA COMPOSITION** indique la **DEPENDANCE** des Objets entre eux !!

```

    leMoteur=new Moteur() ;
    For (int i=0 ;i<4 ;i++)
        lesRoues.Add(new Roue());
    }
    //méthodes
    public bool Rouler()
    {
        if (étatDémarré == true)
            étatRoulé=leMoteur.EntrainRoues(lesRoues);
    }

    public bool Démarrer()
    {
        leMoteur=new Moteur() ;
        étatDémarré=leMoteur.Démarrer() ;
        return étatDémarré ;
    }
}

public class Moteur
{
    //attributs
    private bool étatMoteur=false ;
    private ArrayList lesRoues ;

    //constructeur
    //méthodes
    public bool Démarrer()
    {
        return étatMoteur=true ;
    }

    public bool EntrainRoues(ArrayList LesR)
    {
        bool ok = false;
        lesRoues = LesR;
        //dire à 2 roues de tourner !!!
        Roue r1 = (Roue)lesRoues[0];
        Roue r2 = (Roue)lesRoues[1];
        if (r1.Tourner() && r2.Tourner())
            ok = true;
        return ok;
    }
}

public class Roue
{
    //attributs
    private bool etatTourné=false;
    //constructeur
  
```

IV Aggrégation de classes

-La relation d'aggrégation (losange vide) est une Composition faible où l'aggrégé peut exister seul !! Mais pas l'aggrégat !!!

Ex : En reprenant l'exemple précédent, la Voiture peut très bien exister sans le Moteur, mais dès que le Moteur est présent, il doit être rattaché à la Voiture

-En codage, il suffit de créer une propriété (Modifieur) qui va rattacher le Moteur à la Voiture !!!

```
//propriétés
//méthodes
public bool Tourner()
{
    return etatTourné = true;
}
```

```
}
```

Ex d'application exploitant la classe Voiture

```
Public static void Main()
{
```

```
..... idem code précédent !!
```

```
// A la fin du code
```

```
//Démarrer la voiture v1 et afficher l'étatDémarré
```

```
bool Ok = false;
```

```
Ok = v1.Démarrer();
```

```
if (Ok == true)
```

```
    Console.WriteLine("La voiture a démarré !");
```

```
else
```

```
    Console.WriteLine("La voiture n'a pas
démarré !");
```

```
// faire rouler la voiture v1
```

```
Ok = false;
```

```
Ok = v1.Rouler();
```

```
if (Ok == true)
```

```
    Console.WriteLine("La voiture roule !");
```

```
else
```

```
    Console.WriteLine("La voiture ne roule
pas!");
```

```
}
```

Ex : public class Voiture

```
{ // attributs
```

```
.....
```

```
private Moteur leMoteur ;
```

```
private ArrayList lesRoues=new
ArrayList() ;
```

```
//propriétés
```

```
public Moteur LeMoteur
```

```
{
```

```
    set { leMoteur=value ;}
```

```
}
```

```
//constructeurs
```

```
public Voiture()
```

```
{
```

```
    For (int i=0 ;i<4 ;i++)
```

```
        lesRoues.Add(new Roue());
```

	<pre> } //méthodes public bool Rouler() { if (étatDémarré == true) étatRoulé=leMoteur.EntrainarRoues(lesRoues); } public bool Démarrerr() { leMoteur=new Moteur() ; étatDémarré=leMoteur.Démarrer() ; return étatDémarré ; } } </pre> <p>Ex d'application exploitant la classe Voiture</p> <pre> public static void Main() { int p=8 ; //création d'une instance de type Voiture Voiture v=new Voiture() ; // initialiser les attributs par les modifieurs v.Marque= "Peugeot" ; v.Modèle="307" ; v.Puissance=p ; //Accéder à un attribut par un accesseur string marque=v.GetMarque() ; //ou string marque=v.Marque ; //création d'une Voiture v1 initialisée au départ Voiture v1=new Voiture("Renault","Laguna",10); //création d'un moteur et rattachement à v1 Moteur m=new Moteur() ; v1.LeMoteur=m ; </pre>