**Héritage en C#**

**1 .Héritage**

Nous abordons ici la notion d'héritage. Le but de l'héritage est de "personnaliser" une classe existante pour qu'elle satisfasse à nos besoins. Supposons qu'on veuille créer une classe *Enseignant* : un enseignant est une personne particulière. Il a des attributs qu'une autre personne n'aura pas : la spécialité et le salaire par exemple. Mais il a aussi les attributs de toute personne : prénom, nom et âge. Un enseignant fait donc pleinement partie de la classe *Personne* mais a des attributs supplémentaires. Plutôt que d'écrire une classe *Enseignant* à partir de rien, on préfèrerait reprendre l'acquis de la classe *Personne* qu'on adapterait au caractère particulier des enseignants. C'est le concept d'**heritage** qui nous permet cela.

***Le principal intérêt de l'héritage est de pouvoir ajouter des attributs et méthodes à une classe tout en profitant de ceux de la classe mère.***

Pour exprimer que la classe *Enseignant* hérite des propriétés de la classe *Personne*, on écrira :

*public class Enseignant : Personne*

*Personne* est appelée la classe mère(ou super classe) et *Enseignant* la classe fille (ou sous classe). Un objet *Enseignant* a toutes les qualités d'un objet *Personne* : il a les mêmes attributs et les mêmes méthodes. Ces attributs et méthodes de la classe parent ne sont pas répétées dans la définition de la classe fille : on se contente d'indiquer les attributs et méthodes rajoutés par la classe fille.

Nous supposons que la classe *Personne* est définie comme suit :

class Personne

{ private string nom;

private string prenom;

public int Age { get; set; }

private static int nbPersonnes;

public Personne()

{

nbPersonnes++;

}

public Personne(string nom,string prenom, int age)

{

this.nom = nom;

this.prenom = prenom;

this.Age = age;

nbPersonnes++;

}

public Personne(Personne p)

{

nom = p.Nom;

prenom = p.Prenom;

Age = p.Age;

nbPersonnes++;

}

public string Nom

{

get { return this.nom; }

set { this.nom=value; }

}

public string Prenom

{

get { return this.prenom; }

set { this.prenom = value; }

}

public static int NbPersonnes

{

get { return nbPersonnes; }

}

public void afficher()

{

Console.WriteLine("nom: " + nom + "\tprenom: " + prenom + "\tage: " + Age);

}

Nous créons une classe *Enseignant* héritant de la classe *Personne* :

// la classe Enseignant dérive de la classe Personne

class Enseignant:Personne

{

// attributs supplimentaire par rapport à la classe personne

private string specialite;

private float salaire;

public Enseignant()

: base() // appel du constructeur de la classe mère(Personne)

{ }

public Enseignant(string nom, string prenom, int age, string sp, float sal)

: base(nom, prenom, age)

{

this.specialite = sp;

this.salaire = sal;

}

// propriétés de l’attribut specialite et salaire

public string Specialite

{

get { return this.specialite; }

set { this.specialite = value; }

}

public float Salaire

{

get { return this.salaire; }

set { this.salaire = value; }

}

}

***Construction d'un objet Enseignant***

Une classe fille **n'herite pas des constructeurs** de sa classe Parent. Elle doit alors définir ses propres constructeurs. Le constructeur de la classe *Enseignant* est le suivant :

public Enseignant(string nom, string prenom, int age, string sp, float sal)

: base(nom, prenom, age)

{

this.specialite = sp;

this.salaire = sal ; }

La déclaration

public Enseignant(string nom, string prenom, int age, string sp, float sal)

: base(nom, prenom, age)

déclare que le constructeur reçoit cinq paramètres *nom*, *prenom*, *age*, *sp, sal* et en passe trois *(nom, prenom,age)* à sa classe mère (ici la classe *Personne)*. On sait que cette classe a un constructeur *Personne(string, string, int)* qui va permettre de construire une personne avec

les paramètres passsés *(prenom,nom,age)*.

Une fois la construction de la classe mère terminée, la construction de l'objet *Enseignant* se poursuit par l'exécution du corps du constructeur :

this.specialite = sp;

this.salaire = sal;

En résumé, le constructeur d'une classe dérivée :

* passe à sa classe mère les paramètres dont celle-ci a besoin pour se construire
* utilise les autres paramètres pour initialiser les attributs qui lui sont propres

Tentons un premier programme de test:

class Program

{

static void Main(string[] args)

{

Enseignant e = new Enseignant("asri", "kamal", 32, "développement", 13000);

e.afficher();

Console.Read();

}

}

Ce programme ce contente de créer un objet *Enseignant* et d’afficher ses informations. La classe *Enseignant* n'a pas de méthode affichermais sa classe mère en a une qui est de plus publique : elle devient par héritage une méthode publique de la classe E*nseignant*.

Les résultats obtenus sont les suivants :

Nom: asri prenom: kamal age: 32

On voit que : l’affichage obtenu est celui de l'objet *Personne*

***Redéfinition d'une méthode ou d'une propriéte***

Dans l'exemple précédent, nous avons eu l’affichage des informations de la partie P*ersonne* de l'enseignant mais il manque certaines informations propres à la classe *Enseignant* (la spécialité et le salaire). On est donc amené à écrire une méthode permettant d’afficher les informations de l'enseignant. Il faut ajouter le code suivant dans la classe enseignant :

public new void afficher()

{

base.afficher();

Console.WriteLine("\tspécialité: " + this.specialite + "\t salaire: " + this.salaire);

}

La méthode afficherde la classe *Enseignant* s'appuie sur la méthode afficherde sa classe mère (*base.afficher*) pour afficher sa partie "P*ersonne*" puis complète avec les champ *spécialité et salaire* qui sont propre à la classe *Enseignant*.

La méthode afficher est définie à la fois dans la classe *Enseignant* et sa classe mère *Personne*. Dans la classe fille *Enseignant*, la méthode doit être précédée du mot clé **new** pour indiquer qu'on redéfinit une nouvelle méthode *afficher* pour la classe *Enseignant.*

La classe E*nseignant* dispose maintenant de deux méthodes *afficher* :

* Celle héritée de la classe mère P*ersonne*
* La sienne propre

Si E est un ojet E*nseignant*, *E.afficher()* désigne la méthode *afficher* de la classe E*nseignant*. On dit que la méthode *afficher* de la classe fille **redéfinit** ou cache la méthode *afficher* de la classe mère.

De façon générale, si *O* est un objet et *M* une méthode, pour exécuter la méthode *O.M*, le système cherche la méthode *M* dans l'ordre suivant :

* dans la classe de l'objet O
* dans sa classe mère s'il en a une
* dans la classe mère de sa classe mère si elle existe

De même pour redéfinir une propriété, on utilise le mot clé new.

Supposons que l’âge d’un enseignant doit être supérieur à 23. Dans ce cas il faut redéfinir la propriété Age comme suit :

public new int Age

{

get { return base.Age; }

set

{

if (value > 23) base.Age = value;

else Console.WriteLine("erreur");

}

}

***Le polymorphisme***