

La programmation orientée objet en C#



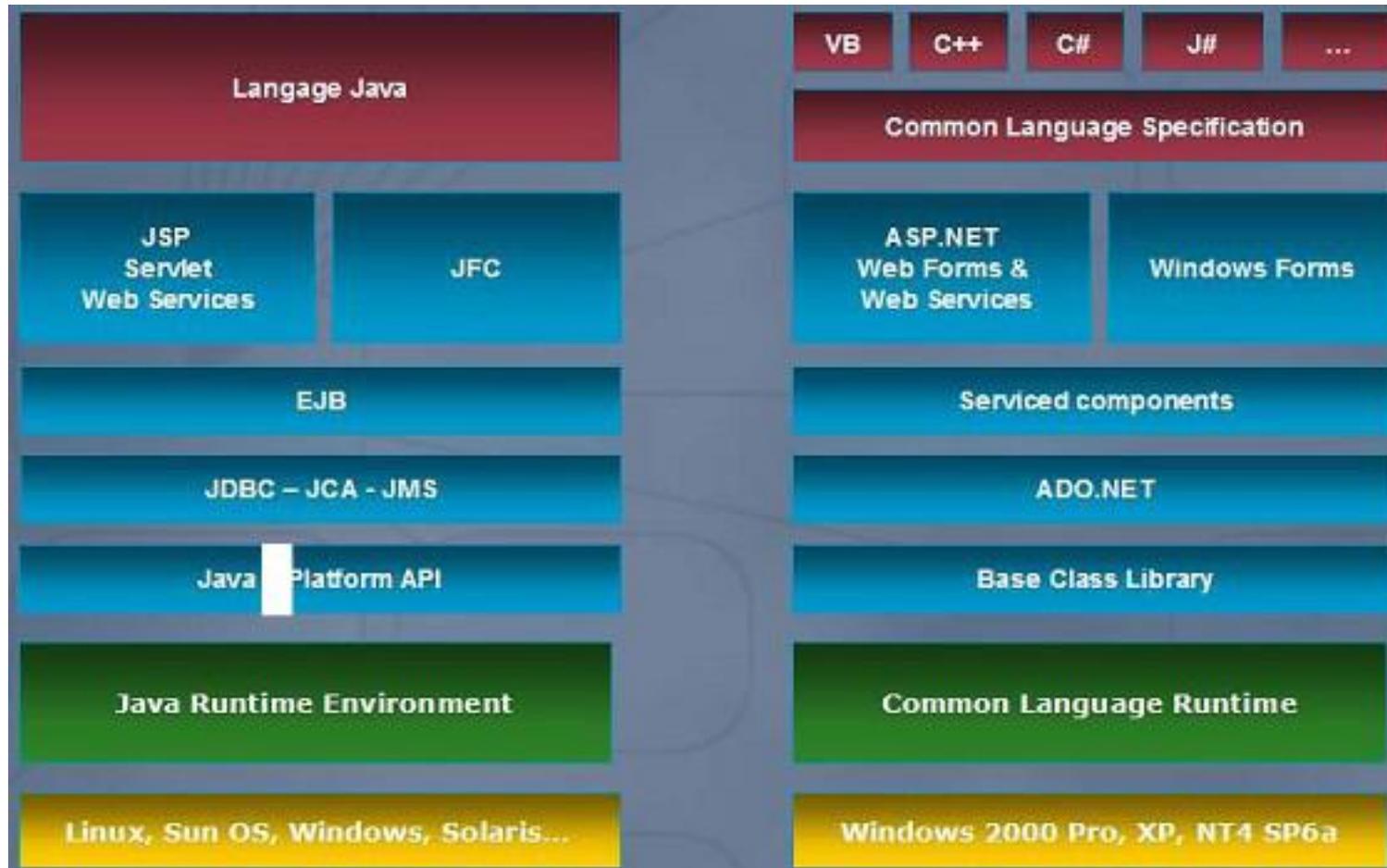
La programmation orientée objet

1. Introduction
2. La syntaxe des éléments de bases en c#
3. Le concept de classe
4. L'héritage
5. Les collections
6. Les exceptions
7. La sérialisation

Introduction



La plateforme .NET



Visual Studio 2008

- ⦿ **Microsoft Visual Studio** est une suite de logiciels de développement pour [Windows](#) conçu par [Microsoft](#).
- ⦿ Visual Studio est un ensemble complet d'outils de développement permettant de générer des [applications Web ASP.NET](#), des [Services Web XML](#), des applications bureautiques et des applications mobile
- ⦿ Supporte la technologie .Net depuis la version 2002
- ⦿ IDE, [Integrated Development Environment](#) : Environnement de développement intégré

Visual Studio 2008

⦿ Les nouveautés de la VS2008:

- Elle est fondée sur le .NET Framework 3.5
- Des outils de métriologie relatifs au code (indicateurs du nombre de lignes, profondeur des routines, calcul de la complexité cyclomatique), relatifs à la performance (mémoire utilisée, temps d'exécution)
- Gestion complète du développement collaboratif et des versions (auteurs et révisions du code) en intégrant l'outil PowerTools
- Possibilité d'automatiser les processus de compilation et intégration (avec des triggers)
- Meilleure gestion des tests avec possibilité d'élaborer des scénarios de test, module de test spécifique aux applications Ajax
- Suppression des wizards assistant et de la bibliothèque ATL pour faire des services web en C++
- Amélioration de prise en charge des fichiers XML (validation IE...).
- Le numéro de version interne de Visual Studio 2008 est 9.0.



Exemple de programme en c#

```
namespace Application1
{
public class BonjourMonde { // Nom de la classe

/* La fonction principale équivaut à la fonction main du C */
public static void main (String[] args) {

// Méthode d'affichage dans la fenêtre console
Console.WriteLine("Bonjour Monde");
}
}
}
```

La syntaxe des éléments de base en c#



La syntaxe et les éléments de bases de c#

- ▶ Les blocs de code sont encadrés par { . . . }
- ▶ Chaque instruction se termine par un ‘;’
- ▶ Une instruction peut tenir sur plusieurs lignes
- ▶ Commentaires ne sont pas pris en compte par le compilateur
(meilleure compréhension du code par le programmeur):
 - ✓ // commentaire sur une seule ligne
 - ✓ /* commentaires ligne 1
commentaires ligne 2 */

Identificateurs

- ▶ Chaque objet, classe, programme ou variable est associé à un nom;
- ▶ Le premier caractère doit être une lettre.
- ▶ Se composer de tous les caractères `_` et `$`, et des caractères alphanumériques.

Ex : `int n_matric2;`

Déclaration des variables

- ▶ Une variable possède un **nom**, un **type** et une **valeur**
- ▶ La déclaration d'une variable permet de **réserver la mémoire** pour en **stocker** la valeur
- ▶ Déclaration d'une variable doit donc contenir deux choses: **Nom** et le **type** de données qu'elle peut contenir
- ▶ Une variable est utilisable dans le bloc ou elle est définie.
- ▶ Le type d'une variable peut être un type élémentaire ou un objet :
 - ✓ float nombre;
 - ✓ int jour, mois, annee ;
 - ✓ int[] nombre = new int[10];

Types élémentaires

- ▶ **Entiers**

byte (8 bits), short (16 bits), int (32 bits), long (64 bits)

- ▶ **Réels**

représentation en virgule flottante
float (32 bits), double (64 bits)

- ▶ **Caractères :**

char (16bits,)

- ▶ **Valeurs logiques**

true et false: boolean

Exemple:

✓ `int n = 15;`

✓ `boolean b = true, A=(2==3);`

✓ `char code='D';`



```
char code;  
code='D';
```

Portée d'une variable

- ▶ La portée d'une variable va de la ligne de sa définition jusqu'à la fin du bloc dans lequel elle est définie.

Exemple:

```
{ ... // bloc englobant
  float k = -7;
    { // bloc B
      int k = 10;      // erreur de compilation

    } // fin du bloc B
} ... // fin du bloc englobant
```

Affectation

- ▶ le signe = est l'opérateur d'affectation et s'utilise avec une expression de la forme:

variable = expression

(a=55,c='B')

- ▶ a+=10 équivalent à : a = a + 10
- ▶ a-= équivalent à : a = a - 10
- ▶ a*= équivalent à : a = a * 10
- ▶ a%=10 reste de la division

Comparaisons

Opérateur	Exemple
>	a > 10
<	a < 10
>=	a >= 0
==	a == 0
!=	a != 0
&&	a && b
	a b

Incrémentation et décrémentation

- ▶ Les opérateurs d'incrément et de décrémentation sont :

`++n` `n++` , `--n` `n--`

- ▶ L'opérateur `++` renvoie la valeur avant incrément s'il est postfixé, après incrément s'il est préfixé :

- ✓ `Console.WriteLine(x++); // est équivalent à`

`Console.WriteLine(x);`

`x = x + 1;`

- ✓ `Console.WriteLine(++x); // est équivalent à`

`x = x + 1;`

`Console.WriteLine(x);`

Structures de contrôles

- ▶ branchements conditionnels:

```
if (boolean)  
    { ... }
```

```
else if (boolean)  
    { }
```

```
else{ ...}
```

```
switch (expression){  
case constante1:  
    instr11;  
    instr12;  
    break;  
case constante2 :  
    ...  
default :...  
}
```

Structures de contrôles

Exemple :

```
if (a < b) {  
    Console.WriteLine ("a est inférieur à b");  
}  
else {  
    Console.WriteLine ("a est supérieur ou égale b");  
}
```

Structures de contrôles

Exemple :

```
int i= 2;  
switch (i){  
  
    case 0: Console.WriteLine("Attention i est null");  
        break;  
    case 2: i=2-i; break;  
    case 3: i=0 ;  
    default: break;  
  
}
```

Structures de contrôles

Exercice 0: Ecrire un programme c# qui permet de diviser 2 nombres.

Exercice 1: Ecrire un programme c# qui permet d'afficher le max des trois nombres A, B, C.

Exercice 2: Ecrire un programme c# qui permet de résoudre l'équation:
 $ax+b=0$

Exercice 3: Ecrire un programme c# qui permet de lister un menu, et affiche à la fin le choix de l'utilisateur (**Utiliser Switch**)

Pour lire une valeur :

```
int b = int.Parse(Console.ReadLine());
```

Structures de contrôles

- ▶ **for (initialisation; condition; modification) {**
 Instruction1 ...
 Instruction 2 ...
}

Exemple:

```
for (int i=0; i < 10; i++) {  
    Console.WriteLine(i);  
}
```

```
for (int i=0, s=0; i < 5; s = i+s, i++) {  
    Console.WriteLine( i + ", " + s);  
}
```

Structures de contrôles

while (boolean)

```
{... // code a exécuter dans la boucle  
}
```

Le code est exécuté tant que le booléen est vrai

Exemple:

```
int i=5;  
while (i > 1) {  
    Console.WriteLine(i);  
    i--;  
}
```

Structures de contrôles

do

```
{... // code a exécuter dans la boucle  
} while (boolean);
```

Exécutée au moins une fois quelque soit la valeur du booléen

Exemple:

```
int i=1;  
do{  
    Console.WriteLine(i);  
} while (i > 1);
```

Structures de contrôles

Exercice 4:

Ecrire un programme c# qui permet d'afficher les carrés des nombres entre 0 et n.

Exercice 5:

Ecrire un programme c# qui permet de calculer la somme et la moyenne de n nombres.

Exercice 6:

Ecrire un programme c# qui permet de représenter un entier en binaire.

Exercice 7:

Ecrire un programme c# qui permet de calculer le factoriel de n ($3! = 1 * 2 * 3$)

Déclaration des tableaux

- ▶ Un tableau est un ensemble indexé de données d'un même type.
- ▶ L'utilisation d'un tableau se décompose en trois parties :
 - Création du tableau ;
 - Remplissage du tableau ;
 - Lecture du tableau ;

int [] monTableau = new int[10];

- ▶ L'opérateur [] permet d'indiquer qu'on est en train de déclarer un tableau.

int [] premierTableau, deuxiemeTableau;

int troisiemeTableau[], variable;

Déclaration des tableaux à deux dimensions

Un tableau à 2 dimension se déclare et s'instancie :

```
float tableau[][] = new float[10][10];
```

- La taille des tableaux de la seconde dimension peut ne pas être identique pour chaque occurrence :

```
int dim1[][] = new int[3][];
```

```
    dim1[0] = new int[4];
```

```
    dim1[1] = new int[9];
```

```
    dim1[2] = new int[2];
```

- ▶ Chaque élément du tableau est initialisé selon son type par l'instruction new :
 - + 0 pour les numériques, '\0' pour les caractères,
 - + false pour les booléens et null pour les chaînes de caractères et les autres objets.

L'initialisation explicite d'un tableau

- ▶ La taille du tableau n'est pas obligatoire si le tableau est initialisé à sa création :

```
int tableau[5] = { 10,20,30,40,50};
```

```
int tableau[] = { 10,20,30,40,50};
```

- ▶ `int tableau[3][2] = {{5,1},{6,2},{7,3}};`

- ▶ Le nombre d'élément de chaque lignes peut ne pas être identique:

```
int[][] tabEntiers = {{ 1,2,3,4,5,6},{ 1,2,3,4},{ 1,2,3,4,5,6,7,8,9}  
};
```

Le parcours d'un tableau

- La variable `length` retourne le nombre d'éléments du tableau.
- L'indice 0 désigne le premier élément du tableau.

```
for (int i = 0; i < monTableau.length; i++) {  
    int element;  
    element = monTableau[i];  
    Console.WriteLine(element);  
}
```

Le parcours d'un tableau

- La variable `length` retourne le nombre d'éléments du tableau.
- L'indice 0 désigne le premier élément du tableau.

```
for (int i = 0; i < monTableau.Length; i++) {  
    int element;  
    element = monTableau[i];  
    Console.WriteLine(element);} 
```

```
int i, j;  
for(i=0; i< dim1.Length; i++) {  
    for(j=0; j< dim1[i].length; j++) {  
        Console.WriteLine(dim1[i][j]);  
    }  
}
```

Les conversions de types

- La conversion int en float :

```
int x = 10;  
float z = (float) x / 3;
```

- La conversion d'un entier int en float:

```
int x = 10;  
double z = (double)10 / 3;
```

- La conversion d'un int en chaîne String;

```
int x = 10;      String s;  
s = ""+x;
```

- La conversion d'une chaîne en int , float, double;

```
String s="12";  
int x = int.Parse(s);  
float z = float.Parse(s);  
double y = double.Parse(s);
```

Les conversions

Méthodes	Rôle
string	pour les chaînes de caractères
int	pour les valeurs entières (integer)
long	pour les entiers long signés (long)
float	pour les nombres à virgules (float)
double	pour les nombres à virgule en double précision (double)

Les classes portent le même nom que le type élémentaire sur lequel elles reposent.



La manipulation des chaînes de caractères

La définition d'une chaîne se fait grâce à l'objet String :
il faut utiliser les méthodes de la classe String d'un objet instancié pour effectuer des manipulations.

```
String s = "bonjour";  
Console.WriteLine("la taille de la chaine est "+ s.Length);  
s=s.Replace('o', 'z');  
Console.WriteLine("la chaine devient : "+s);  
s=s.Insert(3,"-y-");  
Console.WriteLine("la chaine devient : " + s);  
if (s.Contains("ur")==true)  
Console.WriteLine("la chaine contient ur");  
if (s.Equals("bonjour")==false)  
Console.WriteLine("la chaine n egale pas à bonjour");  
String m = s.Substring(0, 2);  
Console.WriteLine("les deux premieres lettres sont :"+ m);  
Console.WriteLine("la chaine en majuscule :"+ s.ToUpper());  
Console.WriteLine("la chaine en minuscule:"+ s.ToLower());
```

Le Concept de Classe

Le concept de classe

- Ensemble de données et de fonctions regroupées dans une même entité.
- Une classe peut être comparée à un moule, qui, lorsque nous le remplissons, nous donne un objet ayant la forme du moule, et toutes ses caractéristiques
- Une classe est une description **abstraite** d'un **objet**
- **Instancier** une classe consiste à **créer un objet** sur son modèle

La syntaxe de déclaration d'une classe

- La syntaxe de déclaration d'une classe est la suivante :

```
modificateurs class nom_de_classe {  
// ...  
  
}
```

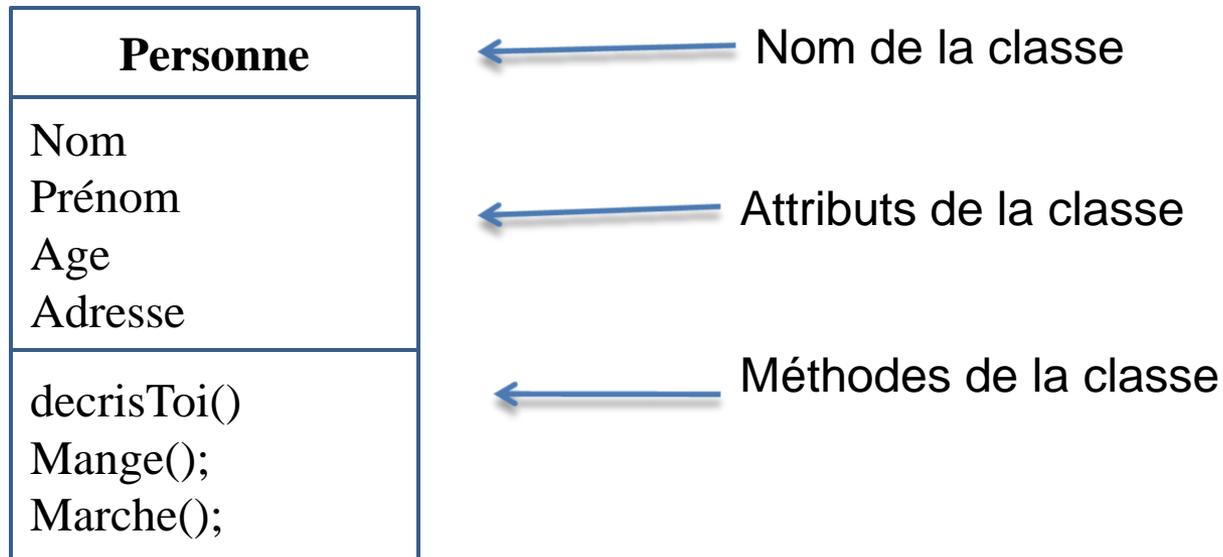
- Exemple :

```
public class Personne{  
    // déclaration des attributs ;  
    // Prototype des méthodes (Traobjents);  
}
```

Les modificateurs de classe

Modificateur	Rôle
public	La classe est accessible partout
private	La classe n'est accessible qu'à partir du fichier où elle est définie
final	<ul style="list-style-type: none">✓ La classe ne peut pas être modifiée, sa redéfinition grâce à l'héritage est interdite.✓ Les classes déclarées final ne peuvent donc pas avoir de classes filles.
abstract	<ul style="list-style-type: none">✓ la classe contient une ou des méthodes abstraites, qui n'ont pas de définition explicite.✓ Une classe déclarée abstract ne peut pas être instanciée : il faut définir une classe qui hérite de cette classe et qui implémente les méthodes nécessaires pour ne plus être abstraite.

Les Classes



Créer la classe Personne

► Le code c#:

```
public class Personne{  
  
// déclarer les attributs  
    String nom;  
    String prenom;  
    int age;  
    String ville;  
  
//les méthodes  
    public String decrisToi()  
        {  
String s = " Je m'appelle "+nom+ " " +prenom+ " J'ai "+ age+ " an J'habite à "+ville;  
Return s;  
        }  
}
```

Instancier la Classe Personne

- La classe est la description d'un objet.
- Un objet est une instance d'une classe.
- Pour chaque instance d'une classe, le code est le même, seules les données sont différentes à chaque objet.
- L'opérateur **new** se charge de créer une instance de la classe et de l'associer à la variable :

```
public class Program {  
    public static void main(String[] args)  
    {  
        Personne p = new Personne ();  
        Console.WriteLine(p.decrisToi());  
    }  
}
```

Instancier la Classe Personne

▶ Le constructeur  Personne():

```
public class Personne{  
// déclarer les attributs  
// Constructeur par défaut  
public Personne(){ }  
//les méthodes  
}  
}
```

• *Le constructeur est en fait une méthode qui n'a aucun type de retour (void, double...) et il porte le même nom que la classe !*

Ceci est une règle immuable : le (les) constructeur(s) d'une classe doit (doivent) porter le même nom que la classe !

Instancier la Classe Personne

- Les constructeurs paramétriques

```
public class Personne{
// déclarer les attributs
    String nom; String prenom; int age; String ville;

// Constructeur paramétrique

    public Personne(String nom){
this.nom= nom;
    }

    public Personne(String nom, String prenom, int age, String ville){
        this.nom= nom; this.prenom= prenom; this.age= age; this.ville= ville;
    }
}
```

➤ **This:** Cette variable sert à référencer dans une méthode l'instance de l'objet en cours d'utilisation d'objets instanciés de la classe.



Instancier la Classe Personne

```
public class Program {  
public static void main(String[] args)  
    {  
  
        Personne p1 = new Personne ();  
        Personne p2 = new Personne ( "n1");  
        Personne p3= new Personne ( "n2", "p2", 29, "rabat");  
        Console.WriteLine( p1.decrisToi());  
        Console.WriteLine( p2.decrisToi());  
        Console.WriteLine(p3.decrisToi());  
  
    }  
}
```

Instancier la Classe Personne

```
public class Program {
public static void main(String[] args)
{
    Console.WriteLine( « entrer un nom »);
    string n=Console.ReadLine();

    Console.WriteLine( « entrer un prenom »);
    string pr=Console.ReadLine();

    Console.WriteLine( « entrer un age»);
    int a=int.Parse(Console.ReadLine());

    Console.WriteLine( « entrer une ville»);
    string v=Console.ReadLine();

    Personne p= new Personne ( n, pr, a, v);

    Console.WriteLine(p.decrisToi());
}
}
```

Instancier la Classe Personne

```
public class Program {  
  
    public static void main(String[] args)  
    {  
        // déclaration d'un tableau des personnes  
        personne[] p = new personne[3];  
  
        p[0]=new personne("n1","p1",12,"v1");  
        p[1] = new personne("n2", "p2", 13, "v2");  
        p[2] = new personne("n3", "p3", 10, "v3");  
  
        foreach (personne s in p)  
        {  
            Console.WriteLine(s.decrisToi());  
        }  
    }  
}
```

Exercice

- ▶ Ecrire une classe « rectangle » qui contient la largeur et la longueur .
- ▶ Créer un constructeur vide ;
- ▶ Créer un constructeur paramétrique;
- ▶ Ecrire les méthodes description() , perimetre() , surface() et iscarree() ;
- ▶ créer dans la classe « program » 3 objets de type rectangle
- ▶ Appeler les méthodes déjà citées

Les modificateurs d'accès

- Placer avant le type de l'objet
- S'appliquent aux classes et/ou aux méthodes et/ou aux attributs
- Assurent le contrôle des conditions d'héritage, d'accès aux éléments et de modification de données par les autres objets

Les modificateurs d'accès pour les variables

Modificateur	Rôle
public	Une variable déclarée public est visible par tout les autres objets
private	C'est le niveau de protection le plus fort. Les composants ne sont visibles qu'à l'intérieur de la classe
Static	<p>➤ Définir une variable de classe qui est partagée entre toutes les instances d'une même classe. Ex:</p> <pre>public class Cercle { static float pi = 3.1416f; float rayon; public Cercle(float rayon) { this.rayon = rayon; } public float surface() { return rayon * rayon * pi; } }</pre>

Les modificateurs d'accès pour les variables

Modificateur	Rôle
Protected	Si une variable est déclarée protected , seules les méthodes présentes dans le même package que cette classe ou ses sous classes pourront y accéder
const	<p>➤Rendre entité non modifiable une fois qu'elle est déclarée pour une méthode ou une classe et initialisée pour une variable</p> <p>Exemple:</p> <pre>public class Constante2 { public const int constante; public initialisation() { this.constante = 10; } }</pre>

Les types des variables

Pour le moment, sachez qu'il y a trois grands types de variables dans une classe objet :

- ▶ **les variables d'instances** : ce sont elles qui définiront les caractéristiques de notre objet ;
- ▶ **les variables de classes** : celles-ci sont communes à toutes les instances de votre classe ;
- ▶ **les variables locales** : ce sont des variables que nous utiliserons pour travailler dans notre objet.

Les variables de classes

- Elles ne sont définies qu'une seule fois quel que soit le nombre d'objets instanciés de la classe.
- Leur déclaration est accompagnée du mot clé static :

```
public class MaClasse() {  
    static int compteur = 0;  
}
```

```
MaClasse m = new MaClasse();
```

```
int c1 = m.compteur;
```

NB

- ▶ Les variables d'instance d'une classe sont déclarées `private` .
- ▶ Les variables de classe sont déclarées `static`.

Les méthodes

- Les méthodes sont des fonctions qui implémentent les traobjets de la classe..

```
modificateurs type_retourné nom_méthode ( arg1, ... ) { ... }
```

- Le type retourné peut être élémentaire ou correspondre à un objet
- Si la méthode ne retourne rien, alors on utilise void.

```
public int add(int a, int b) {  
    return a + b;  
}  
public void add(int a, int b) {  
    Console.WriteLine(a);  
    Console.WriteLine(b);  
}
```

Les méthodes

Modificateur	Rôle
public	la méthode est accessible aux méthodes des autres classes
private	l'usage de la méthode est réservé aux autres méthodes de la même classe
protected	la méthode ne peut être invoquée que par des méthodes de la classe ou de ses sous classes
const	la méthode ne peut être modifiée (redéfinition lors de l'héritage interdite)
abstract	Toutes les méthodes de cette classe abstract ne sont pas implémentées et devront être redéfinies par des méthodes complètes dans ses sous classes
static	la méthode appartient simultanément à tous les objets de la classe

Exercice

- ▶ Créer la classe « user » qui contient le login et le mots de passe et un compteur qui compte le nombre des instances créées
- ▶ Les variables doivent être déclarer private
- ▶ la méthode toString();
- ▶ La méthode compter();
- ▶ Les constructeurs
- ▶ Créer la classe « program » pour instancier 3 objets de type user , appeler les méthodes

La transmission de paramètres

- Lorsqu'un objet est passé en paramètre, ce n'est pas l'objet lui-même qui est passé mais une référence sur l'objet.
- La référence est bien transmise par valeur et ne peut pas être modifiée mais l'objet peut être modifié via un message (appel d'une méthode).

L'émission de messages

- Un message est émis lorsqu'on demande à un objet d'exécuter l'une de ses méthodes.
- La syntaxe d'appel d'une méthode est :
`nom_objet.nom_méthode(parametre, ...) ;`
- Si la méthode appelée ne contient aucun paramètre, il faut laisser les parenthèses vides.

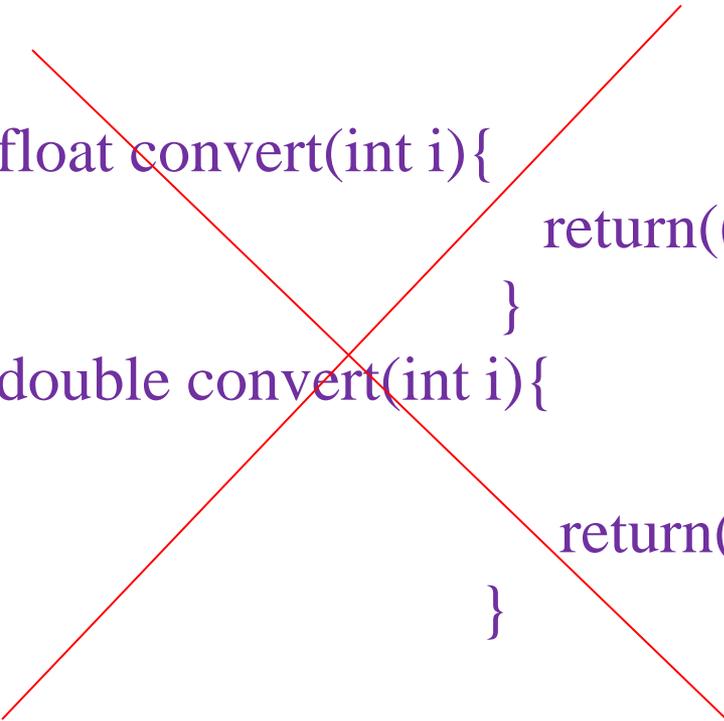
La surcharge de méthodes

- La surcharge d'une méthode permet de définir plusieurs fois une même méthode avec des arguments différents.
- Une méthode est surchargée lorsqu'elle exécute des actions différentes selon le type et le nombre de paramètres transmis.

```
class Affiche{  
    public void afficheValeur(int i) {  
        Console.WriteLine(" nombre entier = " + i);  
    }  
    public void afficheValeur(float f) {  
        Console.WriteLine(" nombre flottant = " + f);}}}
```

La surcharge de méthodes

```
class Affiche{  
    public float convert(int i){  
        return((float) i);  
    }  
    public double convert(int i){  
        return((double) i);  
    }  
}
```



La création d'objets identiques

```
public class Personne{  
// déclarer les attributs  
    String nom;  
    String prenom;  
    int age;  
    String ville;  
// Constructeur paramétrique ou constructeur par copie  
    public Personne(Personne p){  
        this.nom= p.nom;  
        this.prenom= p.prenom;  
        this.age= p.age;  
        this.ville= p.ville;  
    }  
}
```

La création d'objets identiques

```
MaClasse A = new MaClasse();  
MaClasse B = A;
```

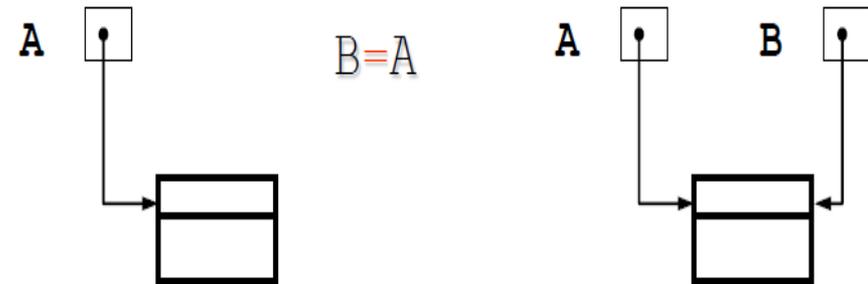
- m1 et m2 contiennent la même référence et pointent donc tous les deux sur le même objet

A et B des objets

- Pour créer une copie d'un objet:

```
MaClasse m1 = new MaClasse();
```

```
MaClasse m2 = new MaClasse(m1)
```



m1 et m2 ne contiennent plus la même référence et pointent donc sur des objets différents.

L'ENCAPSULATION

- ▶ C'est le regroupement dans une même entité appelée objet des données et des traobjents. On ne peut atteindre les attributs d'un objet que par l'intermédiaire de ses traobjents appelés aussi méthodes ou services
- ▶ Les accesseurs et mutateurs : Un accesseur est une méthode qui va nous permettre d'accéder aux variables des objets en lecture et un mutateur, en écriture .
- ▶ On parle de **Getters** et de **Setters**.
- ▶ Les **Getters** sont du même type que la variable qu'ils doivent retourner
- ▶ Les **Setters** sont, par contre, de type void. ces méthodes ne retournent aucune valeur, elles se contentent de les mettre à jour

Getters et Setters

```
public String getNom() {  
    return nom;  
}  
  
public String getPrenom() {  
    return prenom;  
}  
  
public int getAge() {  
    return age;  
}  
  
public String getVille() {  
    return ville;  
}
```

```
public void setNom(String nom) {  
    this.nom = nom;  
}  
  
public void setPrenom(String prenom)  
{  
    this.prenom = prenom;  
}  
  
public void setAge(int age) {  
    this.age = age;  
}  
  
public void setVille(String ville) {  
    this.ville = ville;  
}
```

Getters et Setters

```
public String Nom
{
    get { return nom; }
    set { nom = value; }
}
```

```
public String Prenom
{
    get { return prenom; }
    set { prenom = value; }
}
```

```
public String Ville
{
    get { return ville; }
    set { ville = value; }
}
```

```
public int Age
{
    get { return age; }
    set { age = value; }
}
```

Exercices

Exercice 1: Ecrivez une classe Livre avec les attributs suivants: titre, auteur, prix, annee.

✓ La classe Livre doit disposer des constructeurs suivants :

Livre(), Livre(titre), Livre(titre, auteur), Livre(titre, auteur, prix), Livre(titre, auteur, prix, annee), Livre(Livre).

✓ La classe Livre doit contenir des accesseurs et mutateurs pour les différents attributs.

✓ La classe livre doit aussi contenir une méthode **afficher()** pour afficher les attributs des livres. Une méthode **compter()** pour avoir le nombre des instances créées et une méthode **type()** qui va prendre 1 si le prix de livre <100 et la valeur 2 si le $100 \leq \text{prix} < 500$ et 3 si $500 < \text{prix}$.

✓ Ecrivez aussi une classe de testLivre afin de tester la classe Livre

Exercices

Exercice 2:

Cahier des Charges :

Définir une classe Salarie avec ses données membres et deux méthodes :

- ❑ Une méthode membre, **CalculSalaire()** : qui renvoie le calcul du salaire annuel (salaire *12)
- ❑ Une méthode d'affichage des infos **Affichage()**.
- ❑ Le nombre des salaries inscrits

Définir une classe TestSalarie pour tester

la classe Salaire:

-Créer un objet S de Type Salarie de salaire 10000

-Changer le salaire de S en 12000.

- Afficher le nom de salarie de référence S.

Salarie	
- m_iMatricule	: int
- m_cStatut	: char
- m_sService	: String
- m_sNom	: String
- m_dSalaire	: double
+ CalculSalaire ()	: double
+ Affichage ()	: void

Exercices

Définir une classe ville, qui va contenir une variable dont le rôle sera de stocker le nom, une autre stockera le nombre d'habitants et la dernière se chargera du pays.

- ❑ Une méthode **afficher()** .
- ❑ Une méthode **comparer (ville)** : pour comparer les nombres d'habitants des deux ville.
- ❑ Le nombre des villes qui ont le nombre d'habitant < 10000
- ❑ Le nombre des villes qui ont le nombre d'habitant entre 10000 et 1000000
- ❑ Le nombre des villes qui ont le nombre d'habitant > 1000000
- ❑ La méthode **catégorie ()** : dans quelle tranche se trouve la ville en question. Selon le nombre d'habitants, le caractère renvoyé changera.

{0, 1000, 10000, 100000, 500000, 1000000, 5000000, 10000000};

{ '?', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H' } ;

Par exemple si le nombre d'habitants ≤ 1000 la méthode renvoie le char 'A'.

si le nombre d'habitants =0 la méthode renvoie le char '?'.

- ❑ Classe testville pour tester la classe ville 😊
- ❑ Changer le nombre d'habitants d'une ville V.
- ❑ Affiche le pays de la Ville V.

exercice

- ▶ Soit un livre défini par son nom, le nom de l'auteur, le numéro d'édition et le prix.
- ▶ Créer une classe livre
- ▶ Ajouter les propriétés de chaque attribut
- ▶ Créer les deux constructeurs
- ▶ Ajouter une méthode qui permet d'ajouter 100dh au prix d'un livre donné
- ▶ Créer un programme qui saisit les informations d'un livre, modifie son prix et affiche le nombre de livres créés.

Initialisation d'un objet

```
//Que fournit le programme suivant ?
class A {
private int nbre = 20 ;
private int decal ;
public A (int coeff){
nbre *= coeff ;
nbre += decal ;
}
public void affiche ()
{ Console.WriteLine ("nbre = " + nbre +
" decal = " + decal) ;
}
}
public class InitChmp
{ public static void main (String args[])
{ A a = new A (5) ;
a.affiche() ;
}
}
```

La création d'un objet de type *A* entraîne successivement :

1. l'initialisation par défaut de ses champs *nbre* et *decal* à une valeur "nulle" (ici l'entier 0),
2. l'initialisation explicite de ses champs lorsqu'elle existe ici *nbre* prend la valeur 20,
3. l'appel du constructeur : *nbre* est multiplié par la valeur de *coeff* (ici 5), puis incrémenté de la valeur de *decal* (0).

En définitive, le programme affiche :

nbre = 100 decal = 0

Champs constants

// Quelle erreur a été commise dans cette définition de classe ?

```
class ChCt
{ public ChCt (float r)
  { x = r ;
  }
....
private const float x ;
private const int n = 10 ;
private const int p ;
}
```

Le champ *p* déclaré **const** doit être initialisé au plus tard par le constructeur, ce qui n'est pas le cas. En revanche, les autres champs déclarés **const** sont correctement initialisés, *n* de façon explicite et *x* par le constructeur.

Affectation et comparaison d'objets

```
// Que fournit le programme suivant ?
class Entier
{ public Entier (int nn) { n = nn ; }
  public void incr (int dn) { n += dn ; }
  public void imprime () { Console.WriteLine (n) ; }
  private int n ;
}
public class TstEnt
{ public static void main (String args[])
{ Entier n1 = new Entier (2) ; Console.WriteLine("n1 = ") ;
  n1.imprime() ;
  Entier n2 = new Entier (5) ; Console.WriteLine("n1 = ") ;
  n2.imprime() ;
  n1.incr(3) ; Console.WriteLine("n1 = ") ; n1.imprime() ;
  Console.WriteLine ("n1 == n2 est " + (n1 == n2)) ;
  n1 = n2 ; n2.incr(12) ; Console.WriteLine("n2 = ") ;
  n2.imprime() ;
  Console.WriteLine("n1 = ") ; n1.imprime() ;
  Console.WriteLine ("n1 == n2 est " + (n1 == n2)) ;
}
}
```

```
n1 = 2
n1 = 5
n1 = 5
n1 == n2 est false
n2 = 17
n1 = 17
n1 == n2 est true
```

Champs et méthodes de classe (1)

//Quelles erreurs ont été commises dans la définition de classe //suivante et dans son //utilisation ?

```
class A
{ static int f (int n)
{ q = n ;
}
void g (int n)
{ q = n ;
p = n ;
}
static private const int p = 20 ;
private int q ;
}
public class EssaiA
{ public static void main (String args[])
{ A a = new A() ; int n = 5 ;
a.g(n) ;
a.f(n) ;
f(n) ;
}
}
```

La méthode statique f de A ne peut pas agir sur un champ non statique ; l'affectation $q=n$ est incorrecte.

l'affectation $p=n$ ne l'est pas puisque p est *const* (il doit donc être initialisé au plus tard par le constructeur et il ne peut plus être modifié par la suite).

Dans la méthode *main*, l'appel $a.f(n)$ se réfère à un objet, ce qui est inutile mais toléré. Il serait cependant préférable de l'écrire $A.f(n)$.

Quant à l'appel $f(n)$ il est incorrect puisqu'il n'existe pas de méthode f dans la classe *EssaiA1*. Il est probable que l'on a voulu écrire $A.f(n)$.

Exercice

Réaliser une classe *Cercle*. Chaque *Cercle* sera caractérisé par un nom (de type *char*), un centre de type *char*, une abscisse (de type *double*) et l'ordonnée (de type *double*) et un rayon. On prévoira :

1. un constructeur recevant en arguments le nom, le centre et l'abscisse et l'ordonnée du centre, et le rayon.
2. une méthode *affiche imprimant* (en fenêtre console) le nom du cercle le centre et son abscisse et ordonnée et le rayon,
3. Une méthode *translate(int dx, int dy)* effectuant une translation du centre qui fait changer l'abscisse ($x + dx$) et l'ordonnée ($y+dy$).
4. Une méthode pour calculer le périmètre()
5. Une méthode pour calculer la surface()
6. Une méthode *nbcercles()* pour afficher le nombre des cercles ayant le rayon $< 10\text{cm}$ et le nombre des cercles dont le rayon $> 10\text{cm}$

Écrire un petit programme utilisant cette classe pour créer un cercle, en afficher les caractéristiques, afficher le périmètre et la surface.

le déplacer avec $dx=2$ et $dy=4$ et en afficher à nouveau les caractéristiques.

1. Créer un cercle C de nom C, de centre O (9,4) et de rayon 12.
2. Changer le rayon à 13cm et afficher le nouveau rayon.

L'héritage

L'héritage

1. La notion d'héritage
2. Le polymorphisme
3. Les classes abstraites
4. Les interfaces

Problème

▶ Classe Personne

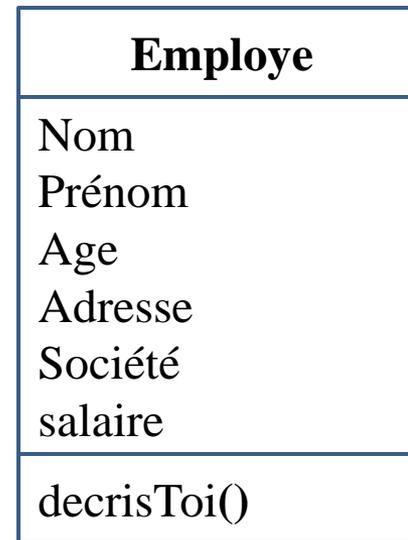
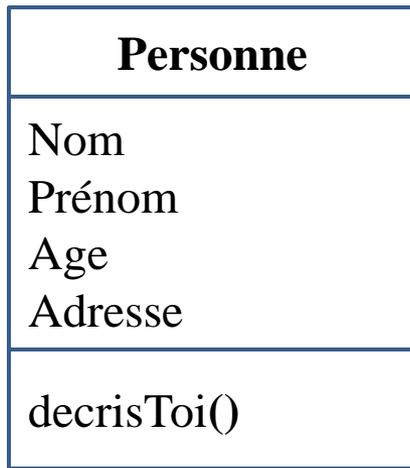
▶ Une personne contient un nom , prenom, age et adresse

▶ La classe Employe

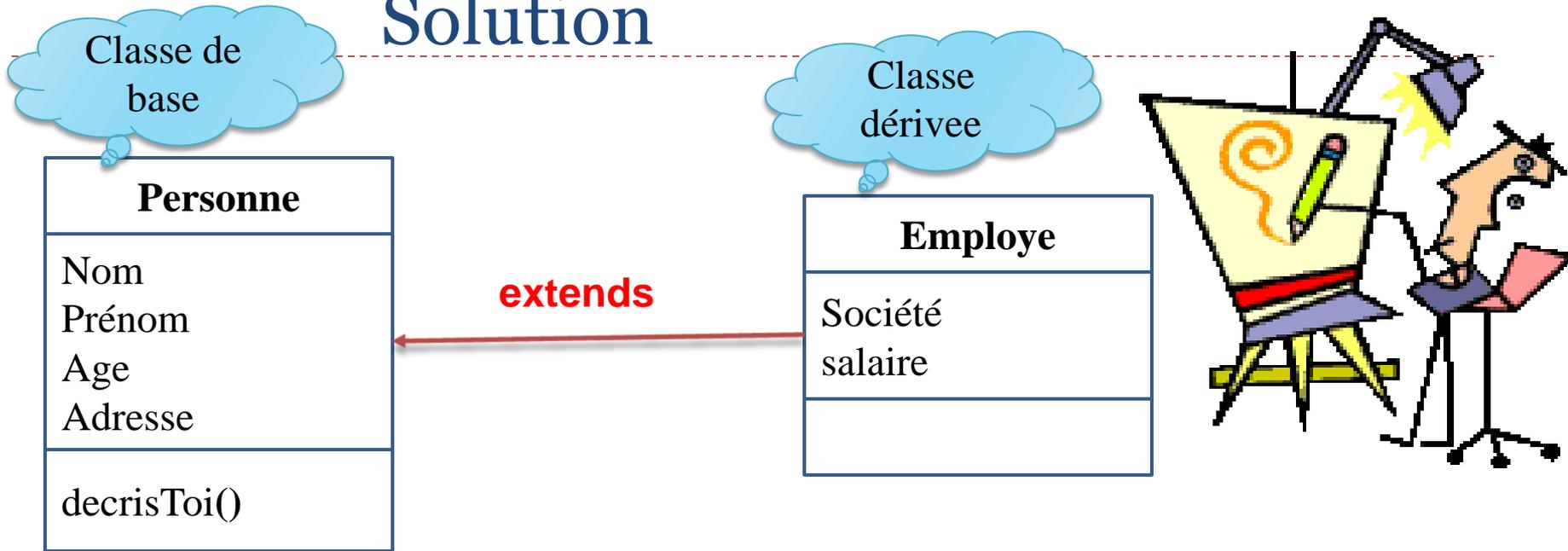
Un employé **est une personne** qui travaille dans une société.

Son cahier des charges reprend donc celui de la classe Personne.

Un employé a une société et un salaire.



Solution



❑ la notion d'héritage est l'un des fondements de la programmation orientée objet. Grâce à elle, nous pourrons créer des classes héritées (appelées aussi classes dérivées ou classes filles) de nos classes mères (appelées aussi classes de base).

Le mot clé **extends** (qui veut dire hérite) permet de spécifier le nom de la classe mère. La classe fille (Employe) hérite des propriétés et méthodes de la classe Personne.

La notion d'héritage

```
public class Personne{
// déclarer les attributs
protected String nom;
protected String prenom;
protected int age;
protected String ville;
Personne(){ };
// Constructeur paramétrique
public Personne(String nom, String
prenom, int age, String ville){
this.nom= nom;
this.prenom= prenom;
this.age= age;
this.ville= ville;
}
public String decrisToi(){
Return (" Je m'appelle "+nom+" "+
prenom+ " J'ai "+ age+ " an "+ "
J'habite à "+ville;
}
}
```

```
public class Employe : Personne{
// déclarer les attributs
private String societe;
private double salaire;
}
```

```
public class Main {
public static void main(String[] args) {
Employe E= new Employe();
C.W(E.decrisToi());
}
}
```

Constructeur par défaut

Héritage et constructeurs

► *Appel implicite au constructeur d'Individu*

```
Employe E= new Employe();  
C.W(E.decrisToi());
```

L'objet *E* se présente comme s'il était une personne car il hérite de la méthode *decrisToi()*. On obtient l'affichage :

Je m'appelle J'ai 0 an J'habite à

Sans qu'on l'ait précisé dans le constructeur de *Employe*, la variable d'instance *nom* a pris la valeur « null ». En effet, pour construire l'instance de *Employe*, le compilateur appelle d'abord le constructeur par défaut de la classe *Personne* :



Comment initialiser le nom, le prenom, l'age et la ville d'un employé ?

Comment initialiser le nom, le prenom, l'age et la ville d'un employé ?

- ▶ On va utiliser le constructeur suivant

```
public Employe(String n, String p, int a, String v, String s, double sa){  
    This.nom=n,  
    This.prenom=p;  
    this.age=a;  
    This.ville=v;  
    This.societe=s;  
    This.salaire=sa;  
}
```

base(n,p,a,v)

```
public Personne(String nom, String  
    prenom, int age, String ville){  
    this.nom= nom;  
    this.prenom= prenom;  
    this.age= age;  
    this.ville= ville;  
}
```

Remarque : **base** est le mot-clé pour désigner l'appel au constructeur de la classe de base (ou classe mère).

Comment initialiser le nom, le prenom, l'age et la ville d'un employé ?

► La solution est:

```
public Employe(String n, String p, int a, String v, String s, double sa) : base(n, p, a, v)
{
    This.societe=s;
    This.salaire=sa;
}
```

• La Classe Main :

```
public static void main(String[] args) {
    Employe p= new Employe("nom1", "prenom1", 32, "Ville1", "NT", 20000);
    C.W(p.decrisToi());
}}
```

• Le Résultat est:

```
Je m'appelle nom1 prenom1 J'ai 32 an J'habite à Ville1
```

Redéfinir une méthode héritée: le polymorphisme

```
public String decritois() {
```

Appel de la méthode `decritois()` de la classe `Mère :Personne`

```
Return base.decrisToi() + " je travaille chez " + societe+ "  
avec un salaire de "+salaire);  
}
```

- Le Résultat devient :

```
Je m'appelle nom1 prenom1 J'ai 32 an J'habite à Ville1  
je travaille chez NT avec un salaire de 20000.0
```

Mots clé virtual, new et override

- ▶ Le mot clé virtual:

- ▶ Est utilisé par les méthodes de la classe mère pour autoriser les classe filles à modifier la méthode

- ▶ Exemple :

```
class Personne
{
    public virtual String Test()
    {
        Return "C'est une personne ";
    }
}
```

Mots clé virtual, new et override

- ▶ La redéfinition des méthodes virtuelles dans la classe fille on utilise 2 mots clés :
 1. **new** : indique au compilateur que vous ajoutez une méthode à une classe dérivée avec le même nom que la méthode dans la classe de base, mais sans aucune relation entre elles

Exemple :

```
class Employe : Personne
{
    public new String Test()
    {
        Return « je suis un employe" );
    }
}
```

Mots clé virtual, new et override

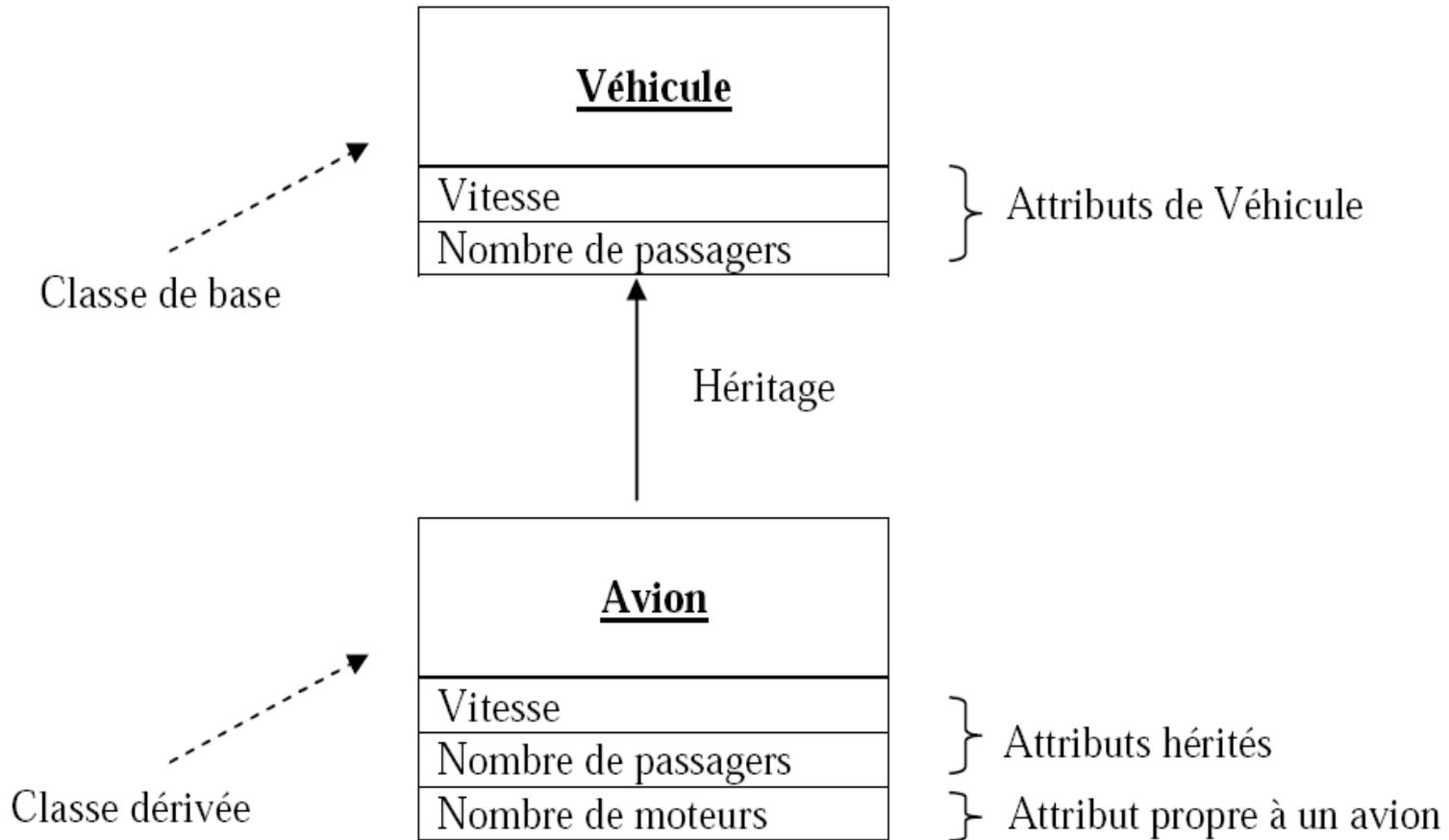
- ▶ La redéfinition des méthodes virtuelles dans la classe fille on utilise 2 mots clés :
- 2. **override**: indique au compilateur que les deux méthodes sont liés

Exemple :

Class **Employe1: Personne**

```
{  
    public override String Test()  
    {  
        return "C'est la méthode redéfinie de personne";  
    }  
}
```

Exemple 2



Solution

```
class vehicule
{
    private float vitesse;
    private int nbrPassagers;
    // constructeur
    public vehicule(float vitesse, int
nombreDePassagers) {
        this.vitesse = vitesse;
        this.nbrPassagers =
nombreDePassagers;
    }
    // les propriétés
    public float Vitesse {
        get { return vitesse; }
        set { value = vitesse; }
    }
}
```

```
public int NbrPassagers
{
    get { return nbrPassagers; }
    set { value = nbrPassagers; }
}
// la méthode afficher
public virtual String affichage()
{
    return "le vehicule a comme vitesse
" + this.vitesse + " et nombre de passers "
+ this.nbrPassagers;
}
```

Solution

```
class avion : vehicule
{
    private int nbrMoteurs;
    // constructeur
    public avion(float vitesse, int
nbrPassagers, int nbrMoteurs)
        : base(vitesse, nbrPassagers)
    {
        this.nbrMoteurs = nbrMoteurs;
    }
    // les propriétés
    public int NbrMoteurs
    {
        get { return nbrMoteurs; }
        set { value=nbrMoteurs; }
    }
}
```

```
// la redifinition de la methode afficher
public override String affichage()
{
    return base.affichage()+ "l'avion a
comme nombre de moteurs "
        + this.nbrMoteurs;
}
}
```

Solution

```
public class test
{
    public static void Main(string[] args)
    {

        vehicule v = new avion(120,3,4) ;

        Console.WriteLine( v. affichage() );

        Console.ReadLine();
    }
}
```

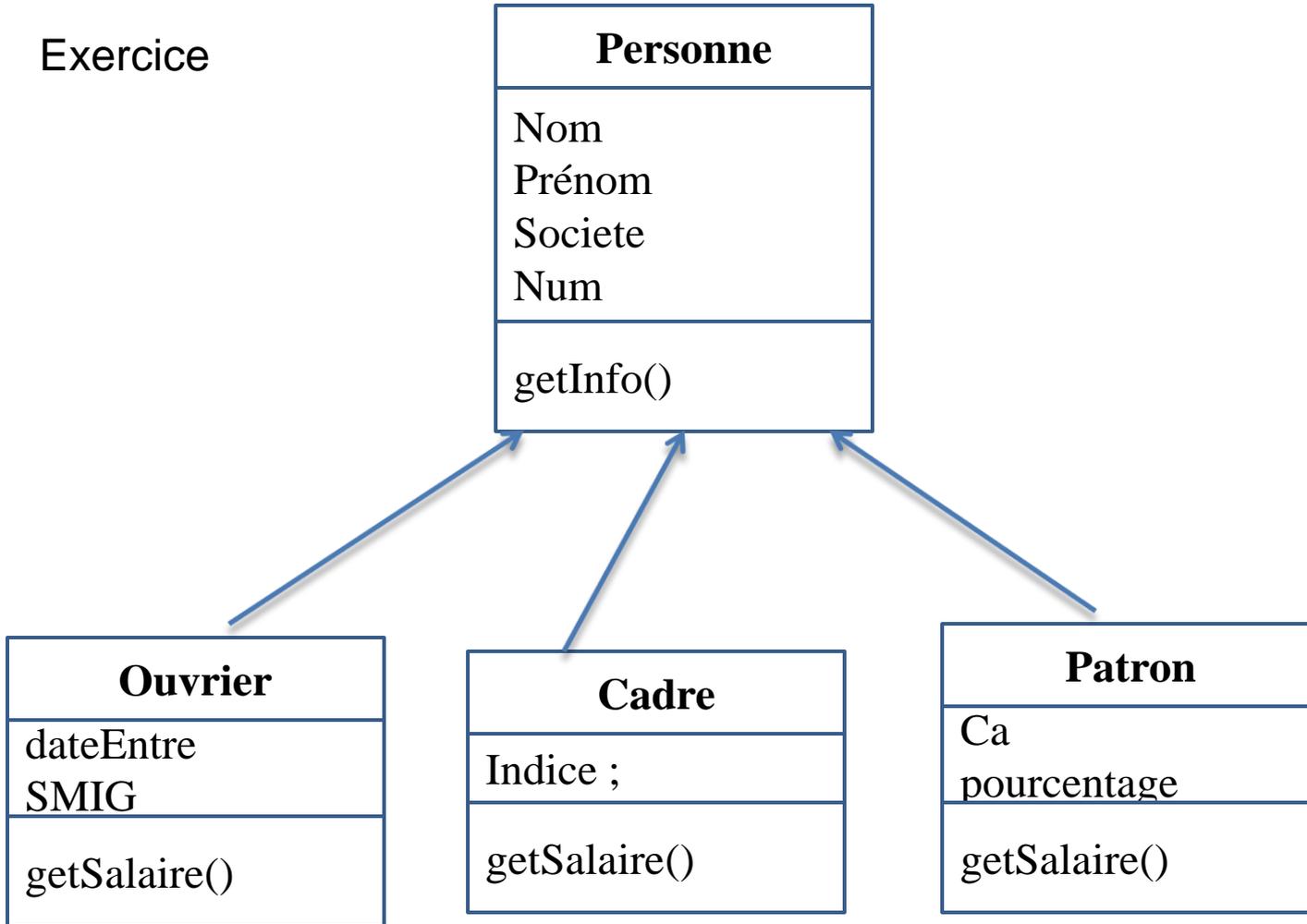
Le polymorphisme par héritage

- ▶ C'est un mot qui signifie plusieurs formes ,la notion de polymorphisme permet à un objet
- ▶ de référencer différents types d'objets ;on peut dire une variable O_i de classe C_i peut référencer non seulement un objet de la classe C_i mais tout objet dérivé de la classe C_i .

Remarques

- ▶ création d'un véhicule `v` de type avion ; `v` peut être considéré comme un avion
- ▶ l'objet `v` invoque la méthode `afficher()`, le compilateur accède à la classe `véhicule`, cherche la méthode `afficher`, trouve le mot clé « `virtual` » Qui signifie que la méthode est virtuelle, donc remplaçable, et il faut chercher sa redéfinition dans la classe `avion`, il accède à la classe `avion` plus précisément la méthode `afficher()`, s'il trouve le mot clé « `new` » il se contente de l'exécution de la méthode `afficher()` de la classe `véhicule`, s'il trouve « `override` » il exécute la méthode `afficher()` de la classe `avion`.
- ▶ « `new` » permet de redéfinir par contre « `override` » permet de remplacer .

Exercice



Exercice

Créer la classe Ouvrier, la classe Cadre et la classe Patron qui héritent de la classe Personne. et prévoir les constructeurs (à 3 et à 0 arguments) de chacune des 3 classes,

Le patron a un salaire qui est égal à x% du chiffre d'affaire : $\text{salaire} = \text{cA} * \text{pourcentage} / 100$

Le cadre a un salaire qui dépend de son indice :

E1 : salaire annuel brut 130.000,00 F,

E2 : salaire annuel brut 150.000,00 F,

E3 : salaire annuel brut 170.000,00 F,

E4 : salaire annuel brut 200.000,00 F,

L'ouvrier a un salaire qui est $\text{salaire} = \text{SMIG} + (\text{age} - 18) * 100 + (\text{dateCourante} - \text{dateEntree}) * 150$. De plus, le salaire ne doit pas dépasser $\text{SMIG} * 2$.

Ecrire la méthode float *getSalaire()* qui permet de calculer le salaire pour chacune des classes.

Ecrire la méthode String *getInfo()* qui renvoie le nom, le prenom, la société, le salaire et le poste occupé (On pourra utiliser la méthode *getInfo()* de classe mère et rajouter l'information du poste occupé).

Le programme principal devra créer 1 patron et 2 cadres aux indices E1 et E3 et 5 ouvriers. Tester les différentes fonctions.

Afficher les informations concernant les différents employés. Peut-on utiliser une boucle for ?

dans le programme de test taper les informations suivantes :

```
Personne e [ ] = new Personne[8];
e[0]=new Patron("boss", "boss", "ADW",1560206789086,1457000,2);
e[1]=new Cadre("Arthur", "Arthur", "ADW",1600206099086,E2);
e[2]=new Cadre("Roger", "Roger", "ADW", 2700206092389, E1);
e[3]=new Ouvrier("bob", "bob", "ADW", 1501206099086,80);
e[4]=new Ouvrier("bobe", "bob", "ADW", 2591206099086,80);
e[5]=new Ouvrier("bobi", "bob", "ADW", 1701105609086,90);
e[6]=new Ouvrier("bobo", "bob", "ADW", 1551208909086,80);
e[7]=new Ouvrier("bobob", "bob", "ADW",1501206099086,99);
for(int i=0;i<8;i++) Console.WriteLine(e[i].getInfo());
// attention e[i].getSalaire() entraine une erreur de compilation puisque cette fonction
n'existe pas dans la classe Personne
```

Si l'on essaie de comprendre ce qui s'est passé : à l'appel de la méthode *getInfo()* de la classe mère, *c#* cherche la référence de l'objet *e[i]*. Si cette référence correspond à une classe fille (les classes *Ouvrier*, *Cadre* ou *Patron*) alors il y a appel de la méthode de la classe fille. On dit que l'édition de lien est dynamique (polymorphisme) puisque le choix de la méthode n'est pas fait lors de la compilation mais au moment de l'appel de la méthode.

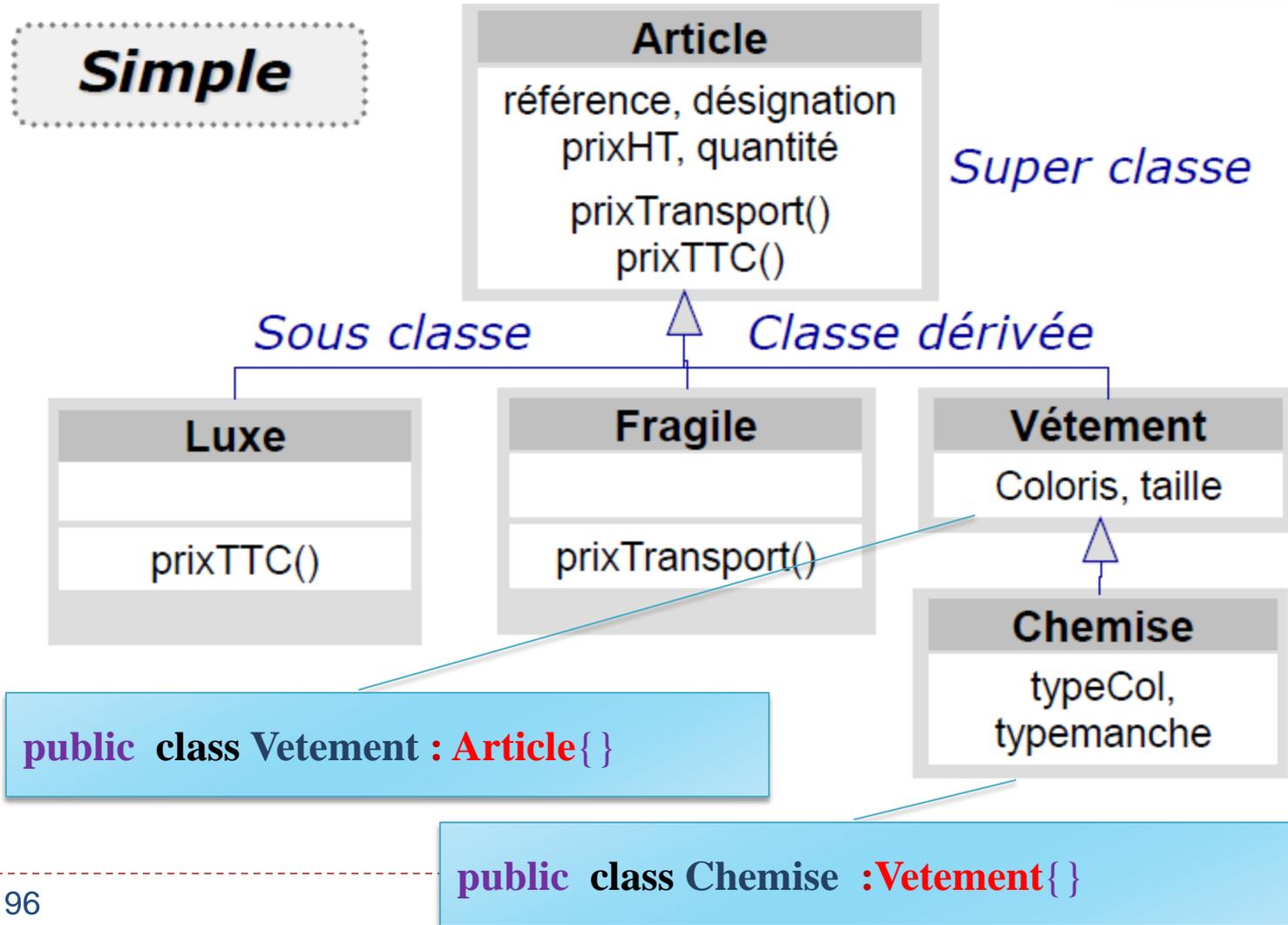
Récapitulatif d'héritage

- ❑ Une classe ne peut hériter que d'une seule et unique classe !
- ❑ Si aucun constructeur n'est défini dans une classe fille, le compilateur en créera un et appellera automatiquement le constructeur de la classe mère.
- ❑ La classe fille hérite de toutes les propriétés et méthodes **public** et **protected** de la classe mère.
- ❑ Les méthodes et propriétés **private** d'une classe mère ne sont pas accessibles dans la classe fille.

Récapitulatif d'héritage

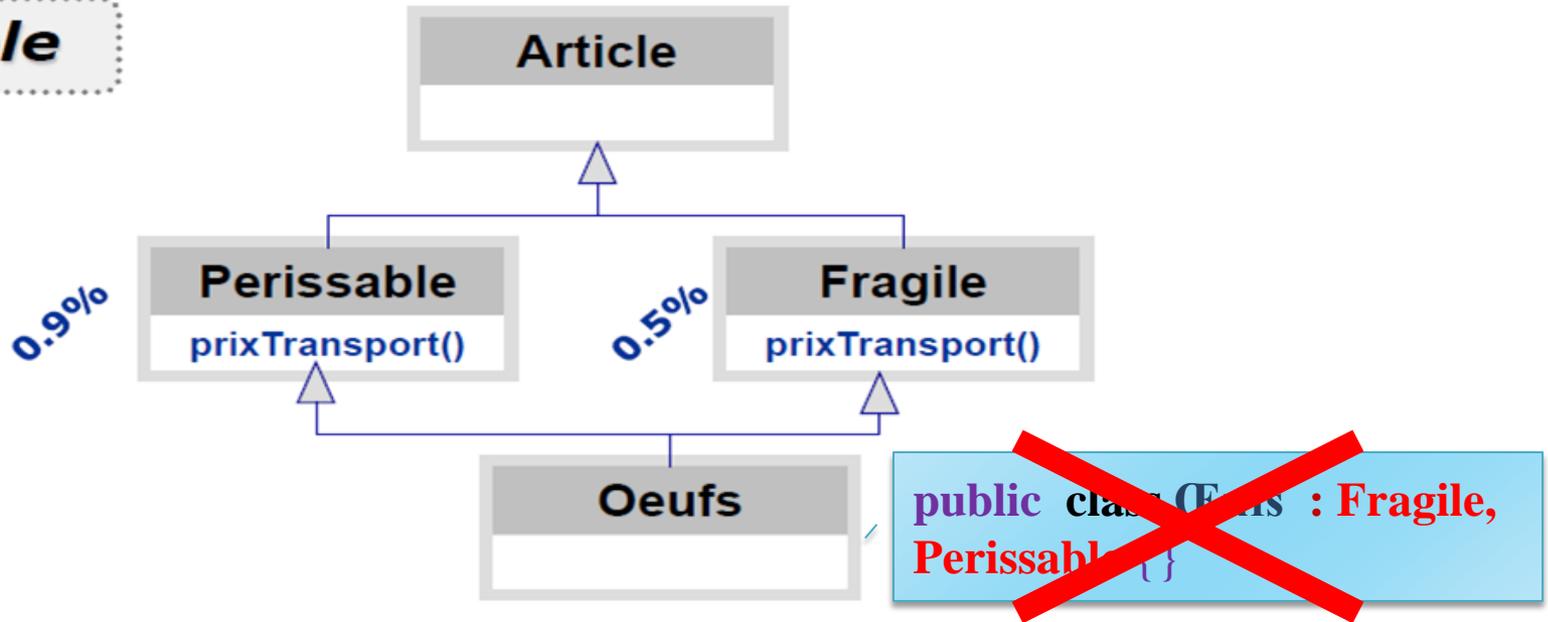
- ❑ On peut redéfinir (changer tout le code) d'une méthode héritée.
- ❑ On peut utiliser **le polymorphisme** sur une méthode par le biais du mot clé **base**.
- ❑ **Le polymorphisme** correspond à la possibilité pour un opérateur ou une fonction d'être utilisable dans des contextes différents (différenciables par le nombre et le types des paramètres) et d'avoir un comportement adapté à ces paramètres.
- ❑ Si une méthode d'une classe mère n'est pas redéfinie ou polymorphée, à l'appel de cette méthode par le biais d'un objet enfant, c'est la méthode de la classe mère qui sera appelée !
- ❑ Vous ne pouvez pas hériter d'une classe déclarée **const**.
- ❑ Une méthode déclarée **const** est non redéfinissable.

Récapitulatif d'héritage



Récapitulatif d'héritage

Multiple



Problème 1 *Conflit de nommage*

→ Quel est le **prix de transport des œufs** ?

Les classes abstraites

- ▶ une classe **abstraite** est comme une classe normale. Ceci dit, elle a tout de même une particularité :
vous ne pouvez pas l'instancier !

```
public class Test{  
public static void main(String[] args){  
  A obj = new A(); //Erreur de compilation !!  
}  
}
```

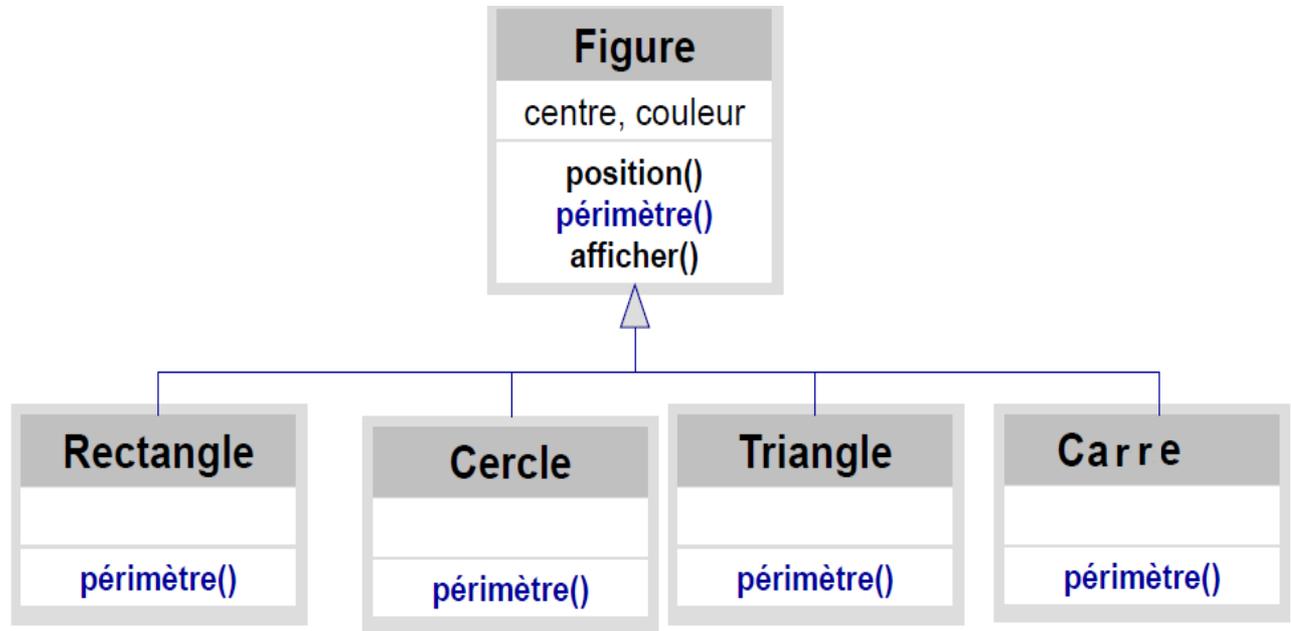
- ▶ À quoi ça sert, alors ?

Notion de classe abstraite

- ▶ Imaginez que vous êtes en train de réaliser un programme qui gère différents types des figures :

Dans ce programme, vous avez :

- ▶ Les cercles ;
- ▶ Les rectangles ;
- ▶ Les triangles ;
- ▶ Et les carrés.



Une classe Figure très abstraite

- ▶ Une classe considérée comme abstraite. Elle doit être déclarée avec le mot clé **abstract**.

```
abstract class Figure{  
}
```

- ▶ Une telle classe peut avoir le même contenu qu'une classe normale (attributs et méthodes). Cependant, ce type de classe permet de définir des méthodes abstraites. Ces méthodes ont une particularité ; elle n'ont pas de corps !

```
public abstract class Figure{  
public abstract Double perimetre(); // une méthode abstraite  
}
```

- ▶ Une méthode abstraite ne peut exister que dans une classe abstraite 😊

Une classe Figure très abstraite

```
abstract public class figure
{
    protected String Centre;
    protected String Couleur;

    //le Constructeur
    public figure(String Cen, String Cou)
    {
        Centre = Cen; Couleur = Cou;
    }
    // la méthode abstraite
    public abstract double perimetre();

    //les méthodes normales
    public String afficher(){return " le centre est :"+ Centre+" la couleur est :
"+Couleur; }
    public String position() { return " le centre est " + Centre; }
}
```

Une classe Cercle

```
public class cercle : figure{
    private double r;
    private const double p =3.14;
//le Constructeur
public cercle(String C,String Co,double ra) : base (C,Co){

    this.r=ra;
}
// Implémentation de la méthode abstraite
public override double perimetre()
{
    return 2 * p * r;
}

// le polymorphisme de la méthode afficher();
public String afficher() {return base.afficher() + " le rayon est : "+r;
}
}
```

Une classe Rectangle

```
public class Rectangle : figure
{
    private double Lar;
    private double Lon;
    //le Constructeur
    public Rectangle(String C, String Co, double L, double La)
        : base(C, Co)
    {
        Lar = L;
        Lon = La;
    }
    // Implémentation de la méthode abstraite
    public override double perimetre() { return 2 * (Lar + Lon); }
    // le polymorphisme de la méthode afficher();
    public String afficher()
    {
        return base.afficher() + " la largeur est : " + Lar + " la longueur est : " + Lon;
    }
}
```

Une classe test

```
public class test
{
    public static void Main(string[] args)
    {
        cercle f = new cercle("C", "red", 12);

        Console.WriteLine(f.afficher());
        Console.WriteLine(f.perimetre());
        Console.WriteLine(f.position());

        Rectangle r = new Rectangle("O", "red", 12, 14);

        Console.WriteLine(r.afficher());
        Console.WriteLine(r.perimetre());
        Console.WriteLine(r.position());
    }
}
```



Les interfaces et l'héritage multiple

- ▶ Avec l'héritage multiple, une classe peut hériter en même temps de plusieurs super classes. Ce mécanisme n'existe pas en c#. Les interfaces permettent de mettre en oeuvre un mécanisme de remplacement.
- ▶ Une interface est un ensemble de déclarations de méthodes abstraites.
- ▶ Tous les objets qui implémentent cette interface possèdent les méthodes déclarées dans celle-ci.
- ▶ Plusieurs interfaces peuvent être implémentées dans une même classe.
- ▶ Les interfaces se déclarent avec le mot clé **interface**.
- ▶ Une interface est implicite déclarée avec le modificateur **abstract**.

Les interfaces et l'héritage multiple

- ▶ Déclaration d'une interface :

```
Public interface nomInterface {  
    // insérer ici des méthodes abstraites et public.
```

- ▶ Implémentation d'une interface :

```
Modificateurs class nomClasse : nomInterface1, nomInterface 2, ...{  
    //insérer ici des méthodes et des champs  
}
```

Les interfaces et l'héritage multiple

Exemple:

```
Public interface A{  
double Somme(); }
```

```
public class B : A {  
    double b;  
    double a;  
    public double somme() {  
        return a+b;  
    }  
}
```

Les Collections

Les collections

- 1. La classe ArrayList**
- 2. La classe générique List<T>**
- 3. La classe générique HashSet<T>**
- 4. La classe générique LinkedList<T>**
- 5. La classe Dictionary<TKey,TValue>**
- 6. La classe Hashtable <TKey,TValue>**
- 7. La classe Hashset <TKey,TValue>**

La classe ArrayList

▶ Définition

- ▶ La classe ArrayList implémente un « tableau dynamique d'objets ».
- ▶ La taille d'un tel Tableau est automatiquement ajustée, si nécessaire, lorsque des éléments y sont ajoutés, ce qui n'est pas le cas pour les tableaux traditionnels.
- ▶ Un tableau dynamique, parfois appelé vecteur, peut contenir n'importe quel objet puisqu'il contient des objets d'une classe dérivée de la classe Object.
- ▶ Dans la pratique, un tableau dynamique contient généralement des objets de même type La classe ArrayList implémente l'interface IList qui, elle-même, implémente les interfaces ICollection et IEnumerable.
- ▶ Cela signifie que la classe ArrayList implémente les propriétés et méthodes mentionnées dans l'interface IList : objs, Add, Clear, etc.

La classe ArrayList

Cette classe implémente un tableau d'objets de taille dynamique (qui peut augmenter ou diminuer à l'exécution)

Les methodes de la classe ArrayList:

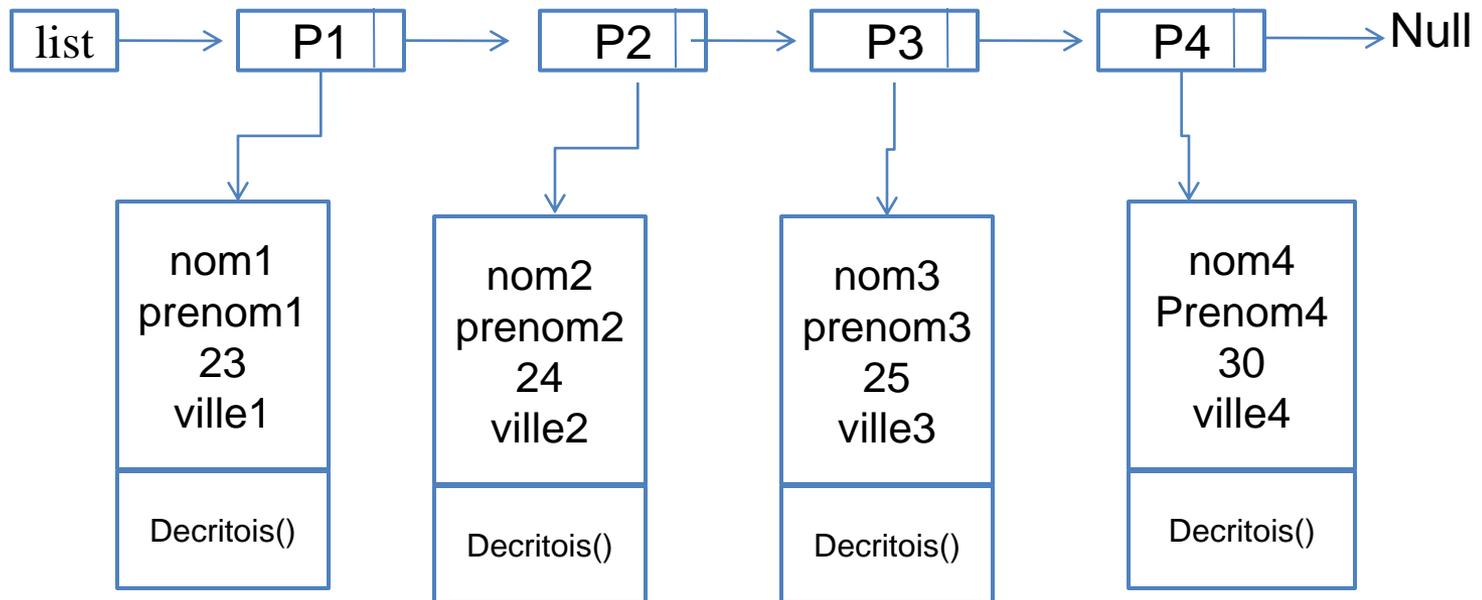
Méthode	Rôle
public void add (Object obj)	permet d'ajouter un objet à la fin du tableau
<i>public int Count</i>	nombre d'éléments de la liste
public void Clear ()	supprime tous les éléments de la liste
Void removeAt (int index)	permet de supprimer l'élément de position index. Elle retourne l'élément supprimé.
Void remove (object o)	Supprime l'objet o

La classe ArrayList

Méthode	Rôle
public bool Contains (object obj)	rend True si obj est dans la liste, False sinon
public void Insert (object obj, int index)	insère obj à la position index de la liste

Exercice

Créer une liste :



Exemple

```
class personne
{
    String nom;
    String prenom;
    int age;
    String ville;
// Constructeur paramétrique
    public personne() { }
    public personne(String nom, String
prenom, int age, String ville)
    {
        this.nom= nom;
        this.prenom= prenom;
        this.age= age;
        this.ville= ville;
    }
//méthode d'affichage
    public String decrisToi()
    {
        String s = " Je m'appelle " +
nom + " " + prenom + " J'ai " +
age + " an J'habite à " + ville;
        return s;
    }
}
```

Exemple

```
using System.Collections;
...
public static void Main(string[] args)
{
    // creer 4 personnes
    personne p1 = new personne("nom1", "prenom1", 23, "Ville1");
    personne p2 = new personne("nom2", "prenom2", 24, "Ville2");
    personne p3 = new personne("nom3", "prenom3", 25, "Ville3");
    personne p4 = new personne("nom4", "prenom4", 30, "Ville4");

    // creer la liste
    ArrayList list = new ArrayList();

    // remplir la liste
    list.Add(p1);
    list.Add(p2);
    list.Add(p3);
    list.Add(p4);
}
```

```
// parcourir la liste 1 ere méthode

    for (int i = 0; i < list.Count; i++)
    {
        //convertir l'objet list[i] en personne.
        personne p = (personne) list[i];
        Console.WriteLine(p.decrisToi());
    }

// parcourir la liste 2eme méthode
    foreach (personne p in list)
    {
        Console.WriteLine(p.decrisToi());
    }

    Console.ReadLine();
}
```

Exemple 2

- ▶ Créer la classe stagiaire qui contient le nom le prenom et la note.
- ▶ Créer les constructeurs
- ▶ Ecrire les propriétés
- ▶ Ecrire la méthode afficher()
- ▶ Ecrire une classe de test pour ajouter 4 stagiaires à une liste « ArrayList »
- ▶ Afficher les stagiaires
- ▶ Afficher les stagiaires qui ont une note > 10;
- ▶ Afficher la moyenne des notes des stagiaires.
- ▶ Chercher un stagiaire dont le nom est entré par l'utilisateur.
- ▶ Modifier la note d'un stagiaire dont le nom est entré par l'utilisateur.
- ▶ Supprimer un stagiaire dont le nom est entré par l'utilisateur.
- ▶ Afficher le nombre des stagiaires
- ▶ Fin 😊

La classe générique `List<T>`

- ▶ La classe `System.Collections.Generic.List<T>` permet d'implémenter des collections d'objets de type `T` dont la taille varie au cours de l'exécution du programme.
- ▶ Un objet de type `List<T>` se manipule presque comme un tableau. Ainsi l'élément `i` d'une liste `l` est-il noté `l[i]`.
- ▶ Pour un objet `List<T>` ou `T` est une classe, la liste stocke là encore les références des objets de type `T`

Exemple

```
using System.Collections;
...
public static void Main(string[] args)
{
    // creer 4 personnes
    personne p1 = new personne("nom1", "prenom1", 23, "Ville1");
    personne p2 = new personne("nom2", "prenom2", 24, "Ville2");
    personne p3 = new personne("nom3", "prenom3", 25, "Ville3");
    personne p4 = new personne("nom4", "prenom4", 30, "Ville4");

    // creer la liste
    List<personne> list = new List<personne>();
    // remplir la liste
    list.Add(p1);
    list.Add(p2);
    list.Add(p3);
    list.Add(p4);
}
```

```
// parcourir la liste 1 ere méthode

    for (int i = 0; i < list.Count; i++)
    {
        Console.WriteLine(list[i].decrisToi());
    }

// parcourir la liste 2eme méthode

    foreach (personne p in list)
    {
        Console.WriteLine(p.decrisToi());
    }

    Console.ReadLine();
}
```

La classe générique HashSet<T>

```
using System.Collections;
...
public static void Main(string[] args)
{
    // creer 4 personnes
    personne p1 = new personne("nom1", "prenom1", 23, "Ville1");
    personne p2 = new personne("nom2", "prenom2", 24, "Ville2");
    personne p3 = new personne("nom3", "prenom3", 25, "Ville3");
    personne p4 = new personne("nom4", "prenom4", 30, "Ville4");

    // creer la liste
    HashSet <personne> list =
        new Hashset <personne>();

    // remplir la liste
    list.Add(p1);
    list.Add(p2);
    list.Add(p3);
    list.Add(p4);
}
```

```
// parcourir la liste 2eme methode

    foreach (personne p in list)
    {
        Console.WriteLine(p.decrisToi());
    }

    Console.ReadLine();
}
```

La classe générique LinkedList<T>

```
using System.Collections;
...
public static void Main(string[] args)
{
    // creer 4 personnes
    personne p1 = new personne("nom1", "prenom1", 23, "Ville1");
    personne p2 = new personne("nom2", "prenom2", 24, "Ville2");
    personne p3 = new personne("nom3", "prenom3", 25, "Ville3");
    personne p4 = new personne("nom4", "prenom4", 30, "Ville4");

    // creer la liste
    LinkedList <personne> list =
        new LinkedList <personne>();

    // remplir la liste
    list.AddLast(p1);
    list.AddLast(p2);
    list.AddLast(p3);
    list.AddLast(p4);

    // parcourir la liste

    foreach (personne p in list)
    {
        Console.WriteLine(p.decrisToi());
    }

    Console.ReadLine();
}
```

Exercice 3

- ▶ Créer la classe personne qui contient la code, le nom , le prenom , et l'age.
- ▶ Créer les constructeurs et les propriétés
- ▶ Ecrire la methode getInfo();
- ▶ Créer une liste qui stocke les personnes
- ▶ Créer la méthode ajouter(personne) qui ajoute une personne à la liste.
- ▶ Créer la méthode modifier_nom (code, nom) qui modifie le nom d'une personne.
- ▶ Créer la méthode supprimer (nom) qui supprime une personne de la liste.
- ▶ Créer la méthode chercher(code) qui cherche une personne connaissant son code.
- ▶ Créer la méthode affichage() qui liste tous les personnes.
- ▶ Créer la méthode compter() qui return le nombre des personnes stockées.
- ▶ créer une fonction catégories() qui affiche la catégorie de chaque personne (« p » si son age <18 , « j » si son age est entre 19 est 35 , et « g » si son age > 35.
- ▶ Ecrire un menu pour appeler ces fonctions.
- ▶ ☺

La classe Dictionary<TKey,TValue>

- ▶ La classe

`System.Collections.Generic.Dictionary<TKey,TValue>`

permet d'implémenter un dictionnaire. On peut voir un dictionnaire comme un tableau à deux colonnes :

Clé	valeur
clé1	valeur1

- ▶ Dans la classe Dictionary<TKey,TValue> les clés sont de type TKey, les valeurs de type TValue.
- ▶ Les clés sont uniques, c.a.d. qu'il ne peut y avoir deux clés identiques.

La classe Dictionary<TKey,TValue>

Méthode	Rôle
public void Add (TKey key, TValue value)	ajoute le couple (key, value) au dictionnaire
<i>public int Count</i>	nombre d'éléments
public void Clear ()	supprime tous les éléments
public bool ContainsKey (TKey key)	rend True si key est une clé du dictionnaire, False si non
public bool ContainsValue (TValue value)	rend True si value est une valeur du dictionnaire, False sinon
public bool Remove (TKey key)	supprime du dictionnaire le couple de clé key. Rend True l'opération réussit, False sinon.

La classe Dictionary<TKey, TValue>

```
using System.Collections;
...
public static void Main(string[] args)
{
    // creer 4 personnes
    personne p1 = new personne("nom1", "prenom1", 23, "Ville1");
    personne p2 = new personne("nom2", "prenom2", 24, "Ville2");
    personne p3 = new personne("nom3", "prenom3", 25, "Ville3");
    personne p4 = new personne("nom4", "prenom4", 30, "Ville4");

    // creer le dictionnaire
    Dictionary<string, personne> dic = new
Dictionary<String, personne>();

    // remplir le dictionnaire
    dic.Add(p1.Nom, p1);
    dic.Add(p2.Nom, p2);
    dic.Add(p3.Nom, p3);
    dic.Add(p4.Nom, p4);
```

```
// pour afficher les clés et les valeurs
foreach (String s in dic.Keys)
{
    Console.WriteLine("la clé est : " + s
    + " la valeur est : " +
    dic[s].decrisToi());
}

// pour afficher les valeurs
foreach (personne p in dic.Values)
{
    Console.WriteLine(p.decrisToi());
}
```

La classe Hashtable <TKey,TValue>

```
using System.Collections;
...
public static void Main(string[] args)
{
    // creer 4 personnes
    personne p1 = new personne("nom1", "prenom1", 23, "Ville1");
    personne p2 = new personne("nom2", "prenom2", 24, "Ville2");
    personne p3 = new personne("nom3", "prenom3", 25, "Ville3");
    personne p4 = new personne("nom4", "prenom4", 30, "Ville4");

    // creer le Hashtable

    Hashtable    dic = new Hashtable();

    // remplir    le dictionnaire
    dic.Add(p1.Nom,p1);
    dic.Add(p2.Nom, p2);
    dic.Add(p3.Nom, p3);
    dic.Add(p4.Nom, p4);
```

```
// pour afficher les clés
    foreach (String s in dic.Keys)
    {
        Console.WriteLine("la clé est : " +s) ;
    }

// pour afficher les valeurs
    foreach (personne p in dic.Values)
    {
        Console.WriteLine(p.decrisToi());
    }
```



Solution de l'exercice 3

```
class personne
```

```
{  
    int code;  
    String nom;  
    String prenom;  
    int age;
```

```
// les propriétés
```

```
public int Code
```

```
{  
    get { return code; }  
    set { code = value; }  
}
```

```
public String Prenom
```

```
{  
    get { return prenom; }  
    set { prenom = value; }  
}
```

```
public int Age
```

```
{  
    get { return age; }  
    set { age = value; }  
}
```

```
public String Nom
```

```
{  
    get { return nom; }  
    set { nom = value; }  
}
```

```
// Constructeur paramétrique
```

```
public personne() { }  
    public personne(String nom, String prenom, int  
age, int code)  
    {  
        this.nom = nom;  
        this.prenom = prenom;  
        this.age = age;  
        this.code = code;  
    }
```

Solution de l'exercice 3

```
public String getinfo()
    { return " Je m'appelle " + nom + " -- " + Prenom + " J'ai " + Age
+ " an j'ai le code " + code;
    }
// déclaration de la liste des personne
static List<personne> lis = new List<personne>();

static public void ajouter(personne p)
    {    lis.Add(p);
    }

static public void modifier_nom(int c ,String n)
    {
    foreach (personne p in lis)
        {
            if (p.code==c)
            {
                p.nom = n;
            }
        }
    }
}
```

Solution de l'exercice 3

```
static public void chercher(int code)
{
    foreach (personne p in lis)
    {
        if (p.code == code)
        {
            Console.WriteLine("la personne cherche est : "+p.getinfo());
        }
    }
}

static public void supprimer( String n)
{
    foreach (personne p in lis)
    {
        if (p.nom.Equals(n))
        {
            lis.Remove(p);
            break;
        }
    }
}
```

Solution de l'exercice 3

```
static public void affichage()  
{  
  
    foreach (personne p in lis)  
    {  
        Console.WriteLine(p.getinfo());  
    }  
}  
  
static public int compter()  
{  
    return lis.Count;  
}
```

Solution de l'exercice 3

```
static public void categories()  
{  
    foreach (personne p in lis)  
    {  
        if(p.age<= 18)  
            Console.WriteLine("la catégorie de "+p.nom+ " est :P" );  
  
        if (p.age >= 19 && p.age<=35)  
            Console.WriteLine("la catégorie de " + p.nom + " est :J");  
  
        if (p.age >= 36)  
            Console.WriteLine("la catégorie de " + p.nom + " est :G");  
    }  
}  
}
```

Solution de l'exercice 3

```
public class test
{
    public static void Main(string[] args)
    {

        int c;

        do
        {
            Console.WriteLine("1- pour ajouter");
            Console.WriteLine("2- pour afficher");
            Console.WriteLine("3- pour modifier");
            Console.WriteLine("4- pour supprimer");
            Console.WriteLine("5- pour rechercher");
            Console.WriteLine("6- pour afficher la gategorie");
            Console.WriteLine("7- pour compter");
            Console.WriteLine("8- pour quitter");

            c = int.Parse(Console.ReadLine());
        }
    }
}
```

Solution de l'exercice 3

```
switch (c)
{
    case 1:
    {
        Console.WriteLine("donner le nom , le prenom, l'age
et le code");

        String n = Console.ReadLine();
        String pr = Console.ReadLine();
        int a = int.Parse(Console.ReadLine());
        int co = int.Parse(Console.ReadLine());
        personne p = new personne(n, pr, a, co);
        personne.ajouter(p);
        break;
    }
    case 2: personne.affichage(); break;
}
```

Solution de l'exercice 3

```
case 3:
{
    Console.WriteLine("entrer le code à modifier");
    int co = int.Parse(Console.ReadLine());
    Console.WriteLine("donner le nouveau nom");
    String n = Console.ReadLine();
    personne.modifier_nom(co, n); break;
}
case 4:
{
    Console.WriteLine("entrer le nom à supprimer");
    String n = Console.ReadLine();
    personne.supprimer(n); break;
}
case 5:
{
    Console.WriteLine("entrer le code à chercher");
    int n = int.Parse(Console.ReadLine());
    personne.chercher(n); break;
}
```

Solution de l'exercice 3

```
        case 6: personne.categories(); break;

        case 7: Console.WriteLine("le nombre des personnes est :" +
                                   personne.compter()); break;
    }
} while (c != 8);

Console.ReadLine();
}
}
```

Exercice

- ▶ Dans une entreprise chaque service se compose d'un chef de service et des employés
- ▶ Ecrire un programme c# qui donne tous les services d'une entreprise ,les chefs de chaque service ainsi ses employés

Par exemples le programme doit m'afficher :

L'entreprise du nom :« e1 » contient les services suivants:

S1 sont chef est MS1(nom,prenom,age...) ses employés sont

Em1 (nom,prenom,salaire...);

Em2 (nom,prenom,salaire...); ...

S2 sont chef est MS2(nom,prenom,age...) ses employés sont

Em3 (nom,prenom,salaire...);

Em4 (nom,prenom,salaire...); ...

Les Exceptions

Problème

Essayons dans ce premier exemple de voir l'effet des erreurs courantes sur le fonctionnement du programme

```
class Program
{
    static void Main(string[] args)
    {
        int a=10, b=0, c;
        Console.WriteLine("Avant division");
        c = a/b;
        Console.WriteLine("Après division, c vaut " + c);
    }
}
```

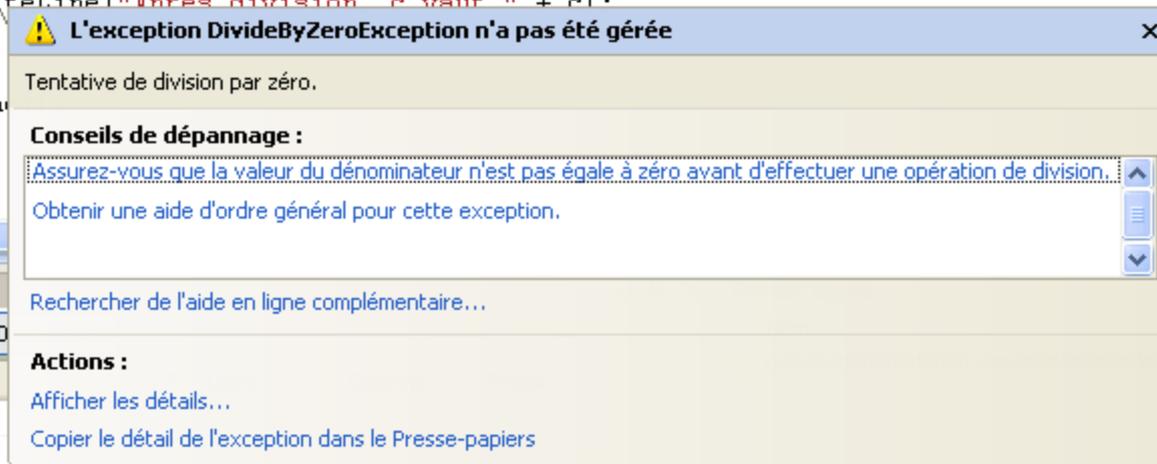
Problème

Essayons dans ce premier exemple de voir l'effet des erreurs courantes sur le fonctionnement du programme

```
public class Program
{
    public static void Main(string[] args)
    {

        int a = 10, b = 0, c;
        Console.WriteLine("Avant division");
        c = a / b;
        Console.WriteLine("Après division c vaut " + c);

        Console.ReadLine();
    }
}
```



Solution1

Pour ne pas avoir de problème il suffit de tester les valeurs entrées :

```
public class Program
{
    public static void Main(string[] args)
    {
int a = 10, b = 0, c;
        Console.WriteLine("Avant division");
        if (b == 0)
            Console.WriteLine("Division par zéro");
        else
        {
            c = a / b;
            Console.WriteLine("Après division, c vaut " + c);
        }
        Console.ReadLine();
    }
}
```

Le problème posé par cette solution est une augmentation des lignes de code et une complication du programme.

De plus, si l'on oublie de tester une valeur, il y aura arrêt brutal du programme (bugs de programmes).

Solution2

La solution : gestion des exceptions bloc try..catch

Le système essaye d'exécuter le code dans les blocs try et si une exception est levée seul le code dans les blocs catch est exécuté.

```
static void Main(string[] args)
{
    try
    {
        int a = 10, b = 0, c;
        Console.WriteLine("Avant division");
        c = a / b;
        Console.WriteLine("Après division, c vaut " + c);
    }
    catch (DivideByZeroException ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

Gestion des EXCEPTIONS

- ▶ une exception est un objet qui est instancié lors d'un incident : on dit qu'une exception est levée.
- ▶ **La capture est effectuée avec les clauses try et catch**
- ▶ la clause try définit un bloc d'instructions pour lequel on désire *capturer* les exceptions éventuellement levées.
- ▶ La clause catch définit l'exception à capturer, en référençant l'objet de cette exception par un paramètre puis le bloc à exécuter en cas de capture.

Gestion des EXCEPTIONS

- ▶ **Syntaxe try catch**
- ▶ La syntaxe est la suivante :

```
try{  
    //code qui peut générer des exception  
}  
catch( TypeException1 e){  
    //code exécuté si une Exception1 est lancée  
}  
catch( TypeException2 e){  
    //code exécuté si une Exception2 est lancée  
}  
finally{  
    //code tout le temps exécuté à la fin  
}
```



Les Exceptions : des classes

Message() retourne le message d'erreur s'il existe.

Exemple des exceptions prédéfinies :

IndexOutOfRangeException indice du tableau est en dehors de ces bornes

DivideByZeroException division par zéro

Créer sa propre exception

- ▶ Créer une classe employé (nom et âge)
- ▶ Si l'utilisateur saisit une valeur d'âge ≤ 18 ou ≥ 60 , le système doit lever une exception
- ▶ La solution consiste à créer une classe ageexception.

```
public class ageexception : Exception
{
    public ageexception(string msg) : base(msg)
    {
    }
}
```

Créer sa propre exception

► Implémenter la classe Employe

```
// le constructeur de la classe employé
public employe(String nom, int age)
{
    //teste age valide ?
    if (age < 18 || age > 60) throw new ageexception("l'age doit etre entre 18 et 60");
    else
    {
        this.nom = nom;
        this.age = age;
    }
}
```

throw : action de lever une exception.

Créer sa propre exception

► Implémenter la classe Employe

```
// la propriété age
public int Age
{
    get { return age; }
    set
    {
        //teste age valide ?
        if (value < 18 || value > 60) throw new ageexception("l'age doit etre entre 18 et 60");
        else
        {
            age = value;
        }
    }
}
```

throw : action de lever une exception.

Créer sa propre exception

- ▶ Implémenter la classe Main

```
public static void Main(string[] args) {  
  
    employe p=null;  
    try  
    {  
        p=new employe ("ee",15);  
        Console.WriteLine(p.getinfo());  
  
    } catch (ageexception e) {  
        Console.WriteLine(e.Message);  
    }  
    finally {  
        Console.WriteLine(p.getinfo());  
    }  
    Console.ReadLine();  
  
}
```

Créer sa propre exception

- ▶ Implémenter la classe Main

```
public static void Main(string[] args) {
    employe p=null;
    try
    {
        p=new employe ("ee",35);
        p.Age = 13;
        Console.WriteLine(p.getinfo());

    } catch (ageexception e) {
        Console.WriteLine(e.Message);
    }
    finally {
        Console.WriteLine(p.getinfo());
    }
    Console.ReadLine();
}
```

La sérialisation d'objet

La sérialisation d'objet

- ▶ La sérialisation (en anglais *serialization*) est un procédé qui consiste à sauver l'état d'un objet sur le disque ou le réseau plutôt que de le garder en mémoire.
- ▶ On peut dire que l'objet est "aplatit" pour pouvoir le convertir en un flux de données, que l'on peut transmettre à un autre objet via la désérialisation
- ▶ La sérialisation et la désérialisation se fait via des fichiers (binaire, XML,...).

Sérialiser un objet (XML)

- ▶ On crée la classe `personne`, pour pouvoir stocker l'objet dans un fichier il faut que celui-ci soit sérialisable.

```
using System.IO;
using System.Xml.Serialization;
namespace test
{
    [Serializable]
    public class personne
    {
        public personne() { }
        string nom;
        public string Nom
        {
            get { return nom; }
            set { nom = value; }
        }
        string prenom;
        ...
        public string Prenom
        {
            get { return prenom; }
            set { prenom = value; }
        }
        string telephone;
        public string Telephone
        {
            get { return telephone; }
            set { telephone = value; }
        }
    }
}
```

Sérialiser un objet

- ▶ On crée la classe personne, pour pouvoir stocker l'objet dans un fichier il faut que celui-ci soit sérialisable.

```
// list des personne
public static List<personne> carnet = new List<personne>();

// serialiser l'objet carnet :

public static void Sauvegarder(string nomfichier)
{
    FileStream f = File.Open(nomfichier, FileMode.OpenOrCreate);
    XmlSerializer s = new XmlSerializer(typeof(List<personne>));
    s.Serialize(f, carnet);

    f.Close();
}
```

Sérialiser un objet

- ▶ On crée la classe `personne`, pour pouvoir stocker l'objet dans un fichier il faut que celui-ci soit sérialisable.

// désérialiser l'objet carnet :

```
public static List<personne> Charger(string nomfichier)
{
    FileStream f = File.Open(nomfichier, FileMode.Open);
    XmlSerializer s = new XmlSerializer(typeof(List<personne>));
    List<personne> lis = (List<personne>)s.Deserialize(f);
    f.Close();
    return lis;
}
```

Sérialiser un objet

```
public static void Main(string[] args)
{
    personne p = new personne();

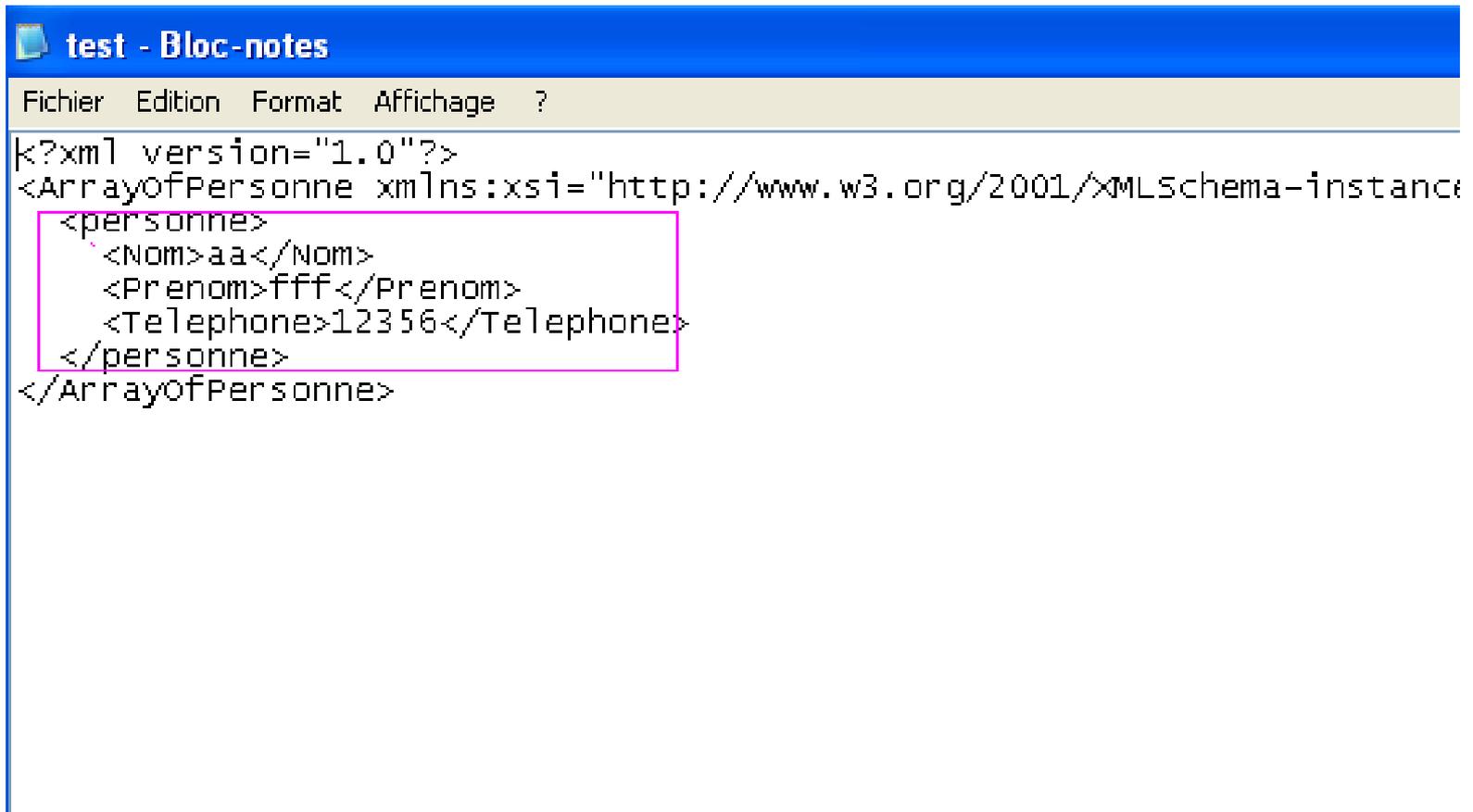
    p.Nom = "aa";
    p.Prenom = "fff";
    p.Telephone = "12356";

    // ajouter la personne à la liste
    personne.carnet.Add(p);

    // sauvgarder dans le fichier nommé test
    personne.Sauvegarder("test");
}
```

Sérialiser un objet

- ▶ Résultat



The screenshot shows a Notepad window titled "test - Bloc-notes". The menu bar includes "Fichier", "Edition", "Format", "Affichage", and "?". The text content is XML code. A pink box highlights the following XML element:

```
<personne>  
  <Nom>aa</Nom>  
  <Prenom>fff</Prenom>  
  <Telephone>12356</Telephone>  
</personne>
```

DéSérialiser un objet

```
▶ public static void Main(string[] args)
    {

    // creer une liste et la remplir à partir du fichier
    List<personne> k = personne.Charger("test");

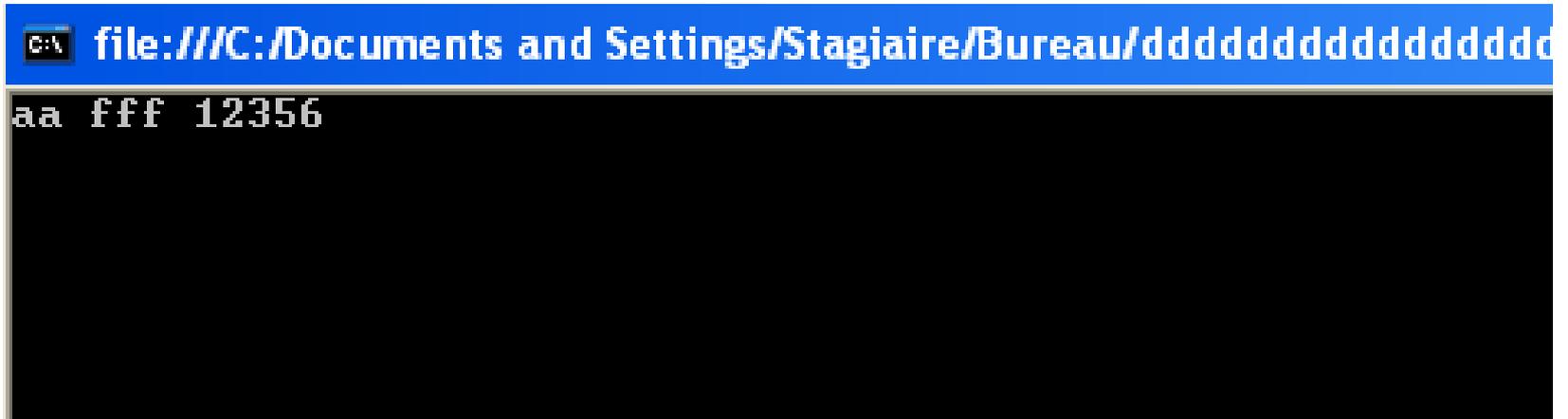
    // parcourir la liste

    foreach (personne s in k)
    {
        System.Console.WriteLine(s.Nom + " " +
            s.Prenom + " " + s.Telephone);
    }
}
```

DéSérialiser un objet

▶ Résultat

▶



```
C:\> file:///C:/Documents and Settings/Stagiaire/Bureau/dddddddddddddddd  
aa fff 12356
```

Sérialiser un objet (BIN)

- ▶ On crée la classe `personne`, pour pouvoir stocker l'objet dans un fichier il faut que celui-ci soit sérialisable.

```
using System.IO;
using System.Xml.Serialization;
namespace test
{
    [Serializable]
    public class personne
    {
        public personne() { }
        string nom;
        public string Nom
        {
            get { return nom; }
            set { nom = value; }
        }
        string prenom;
        ...
        public string Prenom
        {
            get { return prenom; }
            set { prenom = value; }
        }
        string telephone;
        public string Telephone
        {
            get { return telephone; }
            set { telephone = value; }
        }
    }
}
```

Sérialiser et désérialiser un objet (BIN)

// serialiser l'objet carnet en binaire :

```
public static void SauvegarderBin(string nomfichier)
{
    FileStream f = File.Open(nomfichier, FileMode.OpenOrCreate);
    IFormatter s = new BinaryFormatter();
    s.Serialize(f, carnet);
    f.Close();
}
```

// désérialiser l'objet carnet en binaire :

```
public static List<personne> chargerBin(string nomfichier)
{
    FileStream f = File.Open(nomfichier, FileMode.Open);
    IFormatter s = new BinaryFormatter();
    List<personne> lo = (List<personne>)s.Deserialize(f);
    f.Close();
    return lo;
}
```

Sérialiser et désérialiser un objet (BIN)

```
public static void Main(string[] args) {  
  
    personne p = new personne();  
    p.Nom = "aa";      p.Prenom = "fff";      p.Telephone = "12356";  
  
    // ajouter la personne à la liste  
    personne.carnet.Add(p);  
  
    // sauvgarder dans le fichier nommé test  
    personne.SauvegarderBin("test");  
  
    // creer une liste et la remplir à partir du fichier  
    List<personne> k = personne.chargerBin("test");  
  
    // parcourir la liste  
    foreach (personne s in k)  
    {  
        System.Console.WriteLine(s.Nom + " " + s.Prenom + " " + s.Telephone);  
    }  
}
```

Fin



Merci pour votre attention