

SI4 – Bases de la programmation

POO

Avec le langage c#

POO

- * Objectifs à atteindre :
 - * Héritage
 - * Encapsulation
 - * Polymorphisme
- * Ces notions conditionnent un langage en tant que langage objet.
- * Ce cours utilise le C#

POO

- * Encapsulation : faire qu'une valeur ne soit accessible que par une méthode.
- * Héritage : réutiliser les propriétés et méthodes d'autres classes plus génériques, les redéfinir, ...
- * Polymorphisme : capacité d'une méthode à s'adapter à l'objet auquel elle s'applique.

POO

- * On pourrait s'arrêter là, ... tout est dit !
- * Le reste est connu : méthodes = fonctions.
- * Dans le cas de l'encapsulation, on parle d'accesseurs.
- * Accesseurs = getters, setters, fonctions `get()`, `set()`

P00

- * On touche inévitablement à la notion de code
- * `Id Classe::Get() { // TODO ... }`
- * `Id Classe::Set() { // TODO ... }`
- * Et puis, il y a les constructeurs, triviaux, par défaut, paramétrés, de recopies, les destructeurs, ...
- * Nous ajouterons dans ce dossier , les listes, ...
- * Il reste donc des notions à voir ...

POO - Objet

* C'est quoi un objet ?

Un objet informatique est équivalent à un objet de la vie quotidienne.

Il possède des propriétés (attributs) qui permettent de le distinguer des autres.

Il est possible d'effectuer des actions avec.

POO - Objet

- * Un objet possède des méthodes :
 - * Constructeur
 - * Destructeur
 - * Accesseurs
 - * Méthodes propres (affichage, déplacement, ...)

POO - Classe

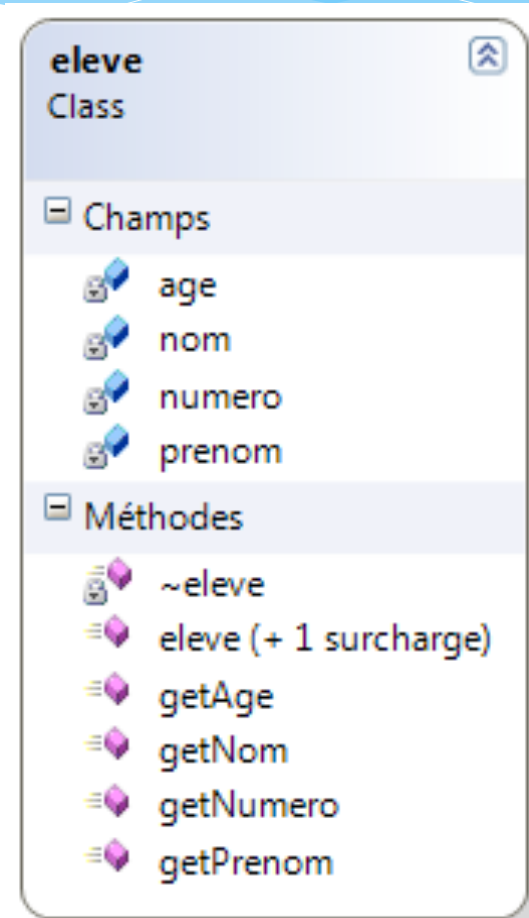
- * Un objet est une instance d'une classe (classe = collection d'objets)
- * La classe décrit deux aspects de l'objets :
 - * Ses attributs
 - * Ses méthodes
- * Au travers de :
 - * Son interface
 - * Son implémentation

POO - Classe

- * Une classe d'objet peut utiliser les propriétés et méthodes d'autres classes d'objets : héritage
- * L'héritage peut provenir de plusieurs hiérarchies différentes pour composer un héritage multiple.
- * Le polymorphisme est une des conséquences de l'héritage.

POO – classe UML

- * Les attributs sont privés et les méthodes publiques



POO - Classe

- * Il est possible de décomposer une classe en deux :
 - * Les attributs
 - * Les méthodes
- * Il est possible de décomposer le code d'une classe en deux :
 - * Interface
 - * Implémentation
- * Ce dernier point tend à disparaître dans les langages modernes.
- * ObjC, C++ conservent ses mécanismes.

PОО – classe interface

```
public partial class eleve
{
    // attributs
    private int numero; //id
    private string nom;
    private string prenom;
    private int age;

    |
    // méthodes
    /*
    public eleve();
    public eleve(int numero, string nom, string prenom, int age);
    public int getNumero();
    public string getNom();
    public string getPrenom();
    public int getAge();
    ~eleve();*/
}
```

PОО – classe implémentation

```
public partial class eleve
{

    public eleve()
    {
        this.nom = "";
        this.prenom = "";
        this.age = 0;
    }

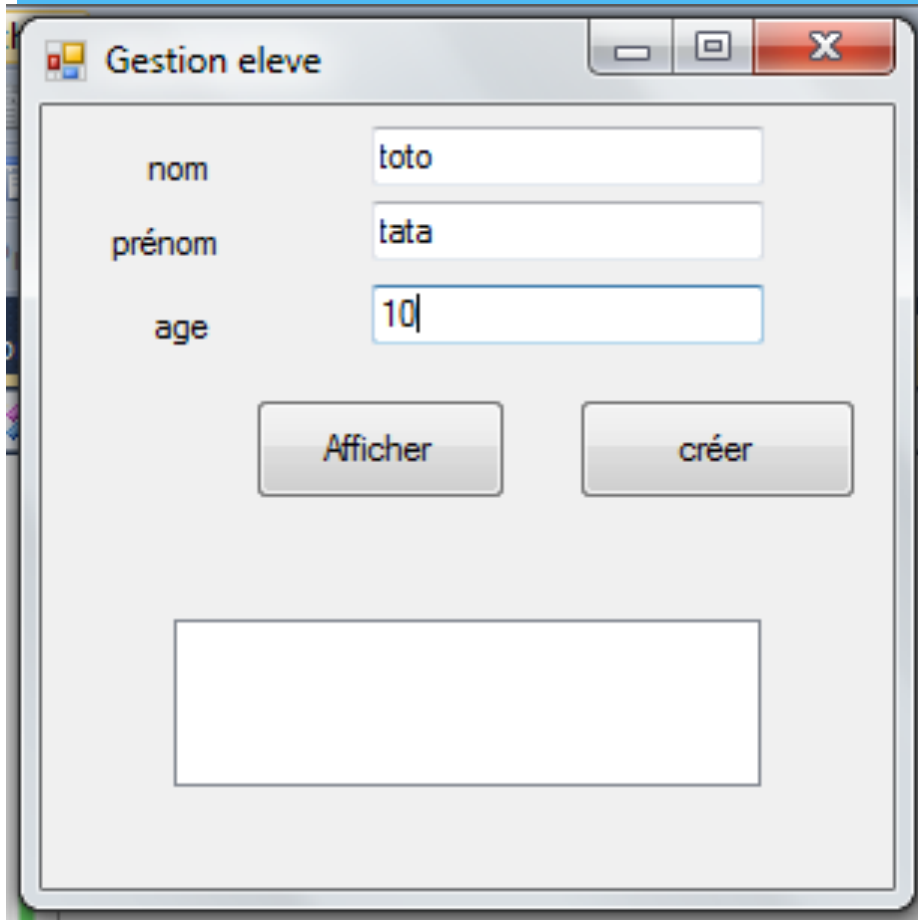
    public eleve(int numero = 0, string nom = "", string prenom = "", int age= 0)
    {
        this.numero = numero;
        this.nom = nom;
        this.prenom = prenom;
        this.age = age;
    }

    public int getNumero()
    {
        return this.numero;
    }
}
```

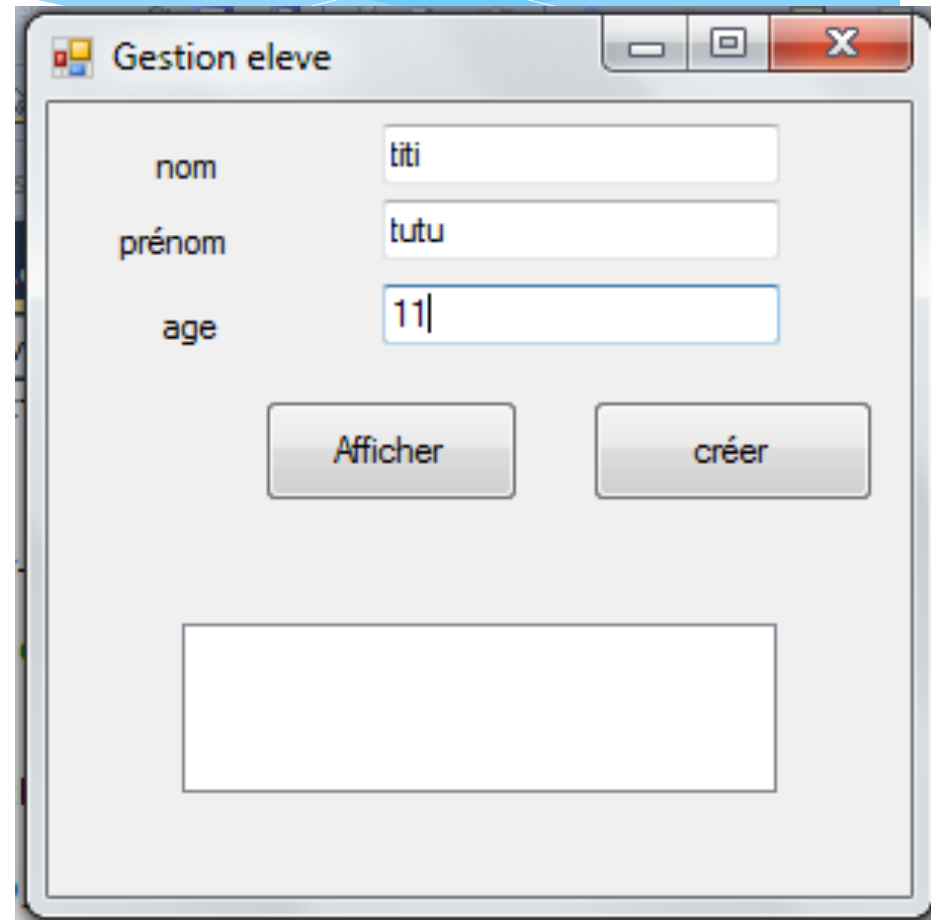
POO - Utilisation

- * Eleve unEleve;
- * // constructeur par défaut
- * unEleve = new eleve();
- * // constructeur paramétré par défaut
- * unEleve = new eleve(1,"toto","tata",10);
- * // accès aux membres par accesseurs
- * string nom = unEleve.getNom();
- * // accès direct impossible, encapsulation !
- * ~~string nom = unEleve.nom;~~

POO - Application



The screenshot shows a window titled "Gestion eleve" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there are three text input fields labeled "nom", "prénom", and "age". The "nom" field contains the text "toto", the "prénom" field contains "tata", and the "age" field contains "10". Below these fields are two buttons: "Afficher" and "créer". At the bottom of the window is a large, empty rectangular box, likely for displaying a list of students.



This screenshot shows the same "Gestion eleve" window, but with different data entered. The "nom" field now contains "titi", the "prénom" field contains "tutu", and the "age" field contains "11". The "Afficher" and "créer" buttons remain in the same positions. The large empty box at the bottom is still present.

* Saisie de 2 élèves

POO - Application

- * Création de 2 objets
- * Affichage de 2 objets

The screenshot shows a Java Swing window titled "Gestion eleve". It contains three text input fields labeled "nom", "prénom", and "age". Below these fields are two buttons: "Afficher" (highlighted with a dashed border) and "créer". At the bottom, there is a text area displaying the output of the application:

```
1 toto tata 10  
2 titi tutu 11
```


POO - héritage

- * Dans le cadre de l'héritage, il faut trouver au moins une classe fille.
- * Un étudiant est un élève, c'est fait !
- * La classe étudiant va hériter de celle d'élève, et ajouter des attributs spécifiques :
 - * Baccalauréat obtenu oui/non, avec série et mention.
 - * Boursier : oui/non

PОО - héritage

- * Il faut ajouter une nouvelle classe au projet
- * Rédiger l'interface et l'implémentation
- * Modifier l'application et l'interface.
- * Penser aux attributs de la classe mère à passer en « protected »

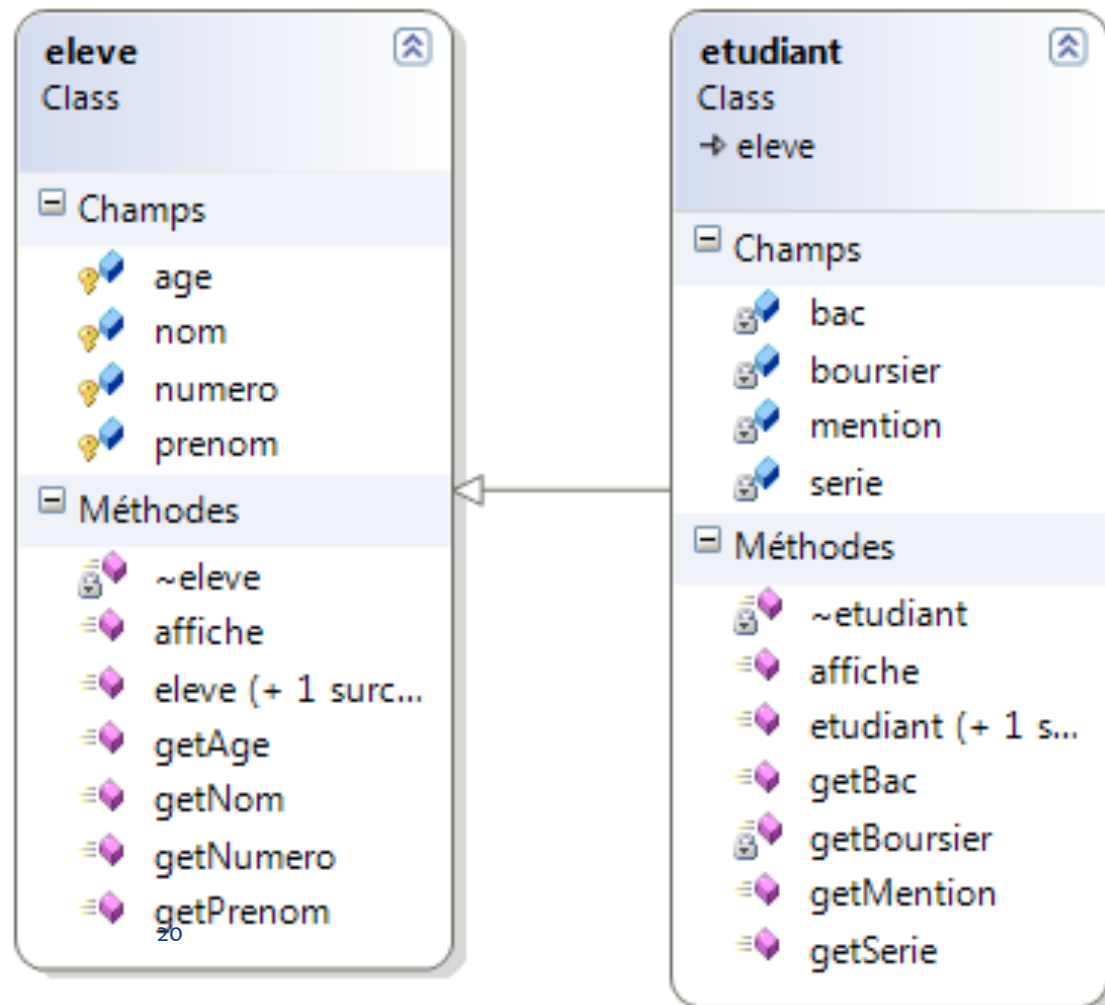
POO - héritage

```
public partial class eleve
{
    // attributs
    protected int numero; //id
    protected string nom;
    protected string prenom;
    protected int age;

    // méthodes
    /*
    public eleve();
    public eleve(int numero, string nom, string prenom, int age);
    public int getNumero();
    public string getNom();
    public string getPrenom();
    public int getAge();
    public int affiche();
    ~eleve();*/
}
```

POO - héritage

Etudiant hérite d'élève
Les attributs d'élève sont
maintenant protégés



PОО – héritage

```
public partial class etudiant : eleve
{

    //attributs
    private bool bac;
    private string serie;
    private string mention;
    private bool boursier;

    //méthodes

    public etudiant();
    bool getBac();
    string getSerie();
    string getMention();
    bool getBoursier();
}
```

PОО - héritage

```
public etudiant()  
{  
    this.bac = false;  
    this.serie = "";  
    this.mention = "";  
    this.boursier = false;  
}
```

this est le pointeur sur l'objet courant
Le mot self est utilisé dans d'autres
langages

```
public etudiant(int numero, string nom, string prenom, int age, bool bac = false, string  
{  
    this.bac = bac;  
    this.serie = serie;  
    this.mention = mention;  
    this.boursier = boursier;  
}
```

```
public bool getBac()  
{  
    return this.bac;  
}
```

POO - héritage

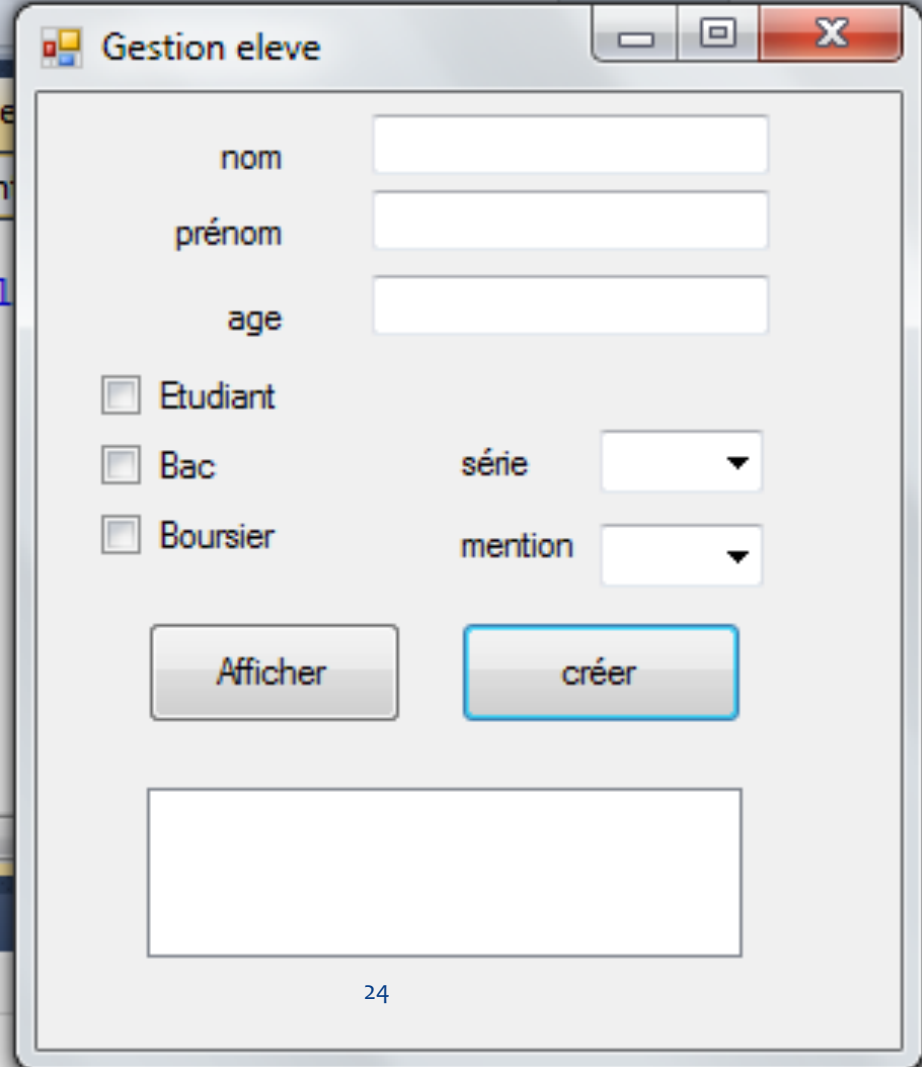
- * // appel du constructeur de la super-classe dans // étudiant
- * :base(numero, nom , prenom, age)
- * Super est utilisé dans d'autres langages.
- * On fabrique un étudiant ou un élève, suivant la saisie, et la case à cocher



Etudiant

POO - héritage

* Nouvelle interface



The screenshot shows a Java Swing window titled "Gestion eleve". The window contains a form with the following fields and controls:

- Text input fields for "nom", "prénom", and "age".
- Three checkboxes: "Etudiant", "Bac", and "Boursier".
- Two dropdown menus: "série" and "mention".
- Two buttons: "Afficher" and "créer".
- A large empty rectangular box at the bottom.

The "créer" button is highlighted with a blue border. The window has standard Windows-style title bar controls (minimize, maximize, close).

Interface

- * Noter au passage que l'on traite de l'événementiel
- * On utilise également des composants :

- * textBox, label,

- * checkBox

- * comboBox

- * listBox

- * Boutons

- * Forms

The form contains the following elements:

- Text input fields for "nom", "prénom", and "age".
- Three checkboxes: "Etudiant", "Bac", and "Boursier".
- Two dropdown menus: "série" and "mention".
- Two buttons: "Afficher" and "créer".
- A large empty rectangular box at the bottom.

Arrows from the text on the left point to the following elements:

- "textBox, label," points to the "nom" input field.
- "checkBox" points to the "Etudiant" checkbox.
- "comboBox" points to the "série" dropdown menu.
- "listBox" points to the "Bac" checkbox.
- "Boutons" points to the "Afficher" button.
- "Forms" points to the large empty box at the bottom.

POO - héritage

- * L'héritage permet à un étudiant de bénéficier des attributs de l'élève. On réutilise.
- * Il peut aussi utiliser ses méthodes.
- * L'étudiant a ses caractéristiques propres, non partagées avec l'élève.
- * Il est possible pour l'étudiant de modifier une méthode de l'élève afin qu'elle s'applique parfaitement à son cas.

POO - polymorphisme

- * Dans notre cas nous allons ajouter une méthode « affiche » à l'élève.
- * Cette méthode permettra de remplir la listBox à l'affichage.
- * Cette méthode sera utilisée par l'étudiant.
- * Nous sommes dans le cas du masquage ou surcharge hérité, la signature de la méthode reste la même dans étudiant.
- * La gestion de cette méthode est dynamique et transparente !

POO - polymorphisme

* Dans la classe élève :

// signature d'affiche

```
public virtual string affiche()  
{  
    return (this.getNumero().ToString() + " " + this.getNom() + " " +  
            this.getPrenom() + " " + this.getAge().ToString());  
}
```

PОО - polymorphisme

* Dans la classe étudiant :

// signature d'affiche

```
public override string affiche()  
{  
    return (this.getNumero().ToString() + " " + this.getNom() + " " +  
        this.getPrenom() + " " + this.getAge().ToString() + " " +  
        this.getBac().ToString() + " "+ this.getSerie() + " "+  
        this.getMention() + " " + this.getBoursier().ToString());  
}
```

* Une autre solution est possible ?

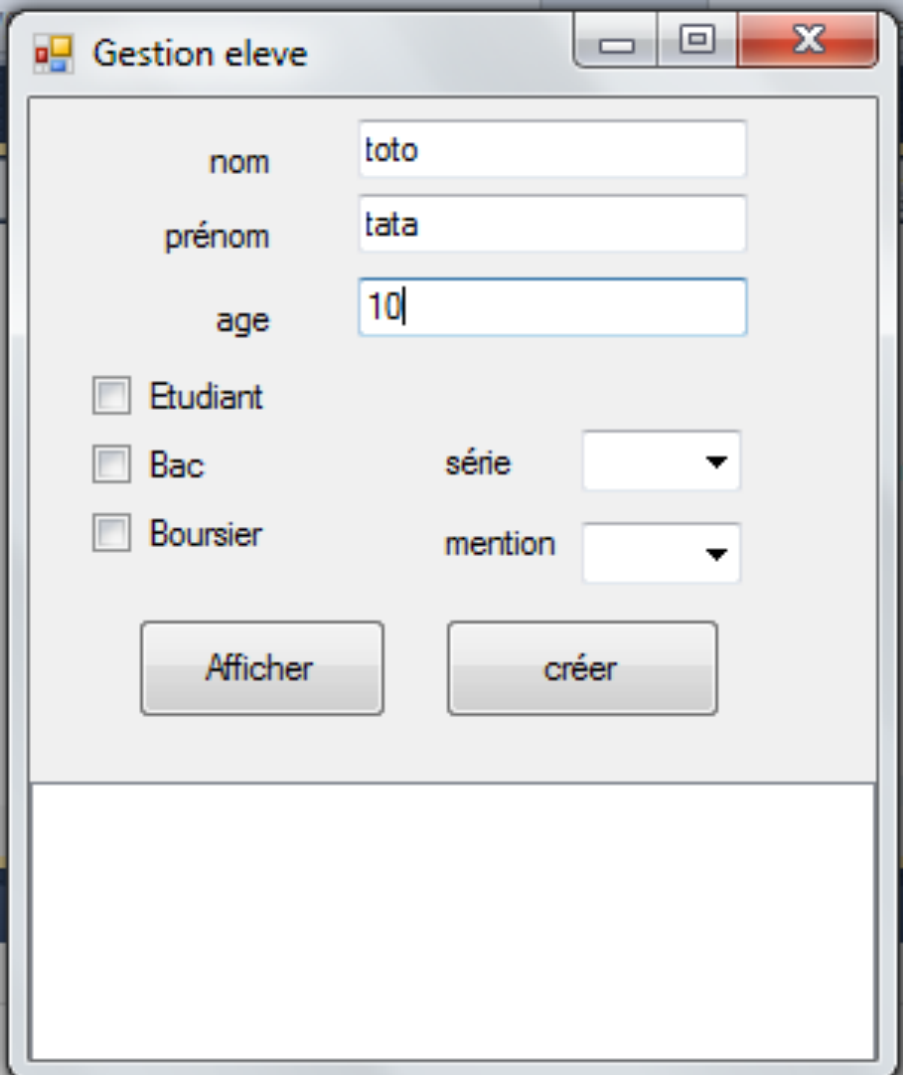
PОО - polymorphisme

* Appel d'affiche() dans l'affichage de la listBox

```
for (int i = 0; i < listEleve.Count(); i++)  
{  
    // adaptation de la méthode à l'objet à afficher suivant  
    élève ou étudiant !  
    listBoxEleve.Items.Add(listEleve[i].affiche());  
}
```

POO - polymorphisme

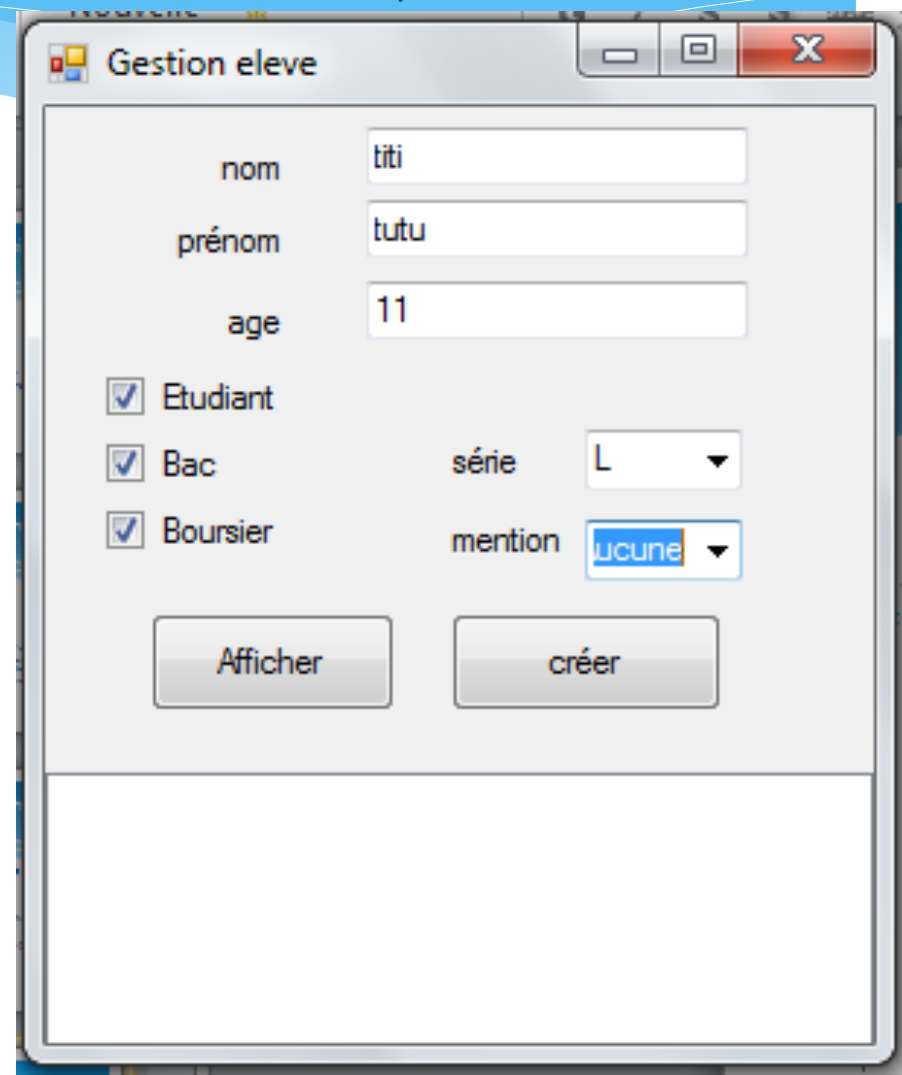
* Ajout de 2 objets (un élève et un étudiant !)



The screenshot shows a window titled "Gestion eleve" with a standard Windows-style title bar (minimize, maximize, close buttons). The form contains the following fields and controls:

- nom: toto
- prénom: tata
- age: 10
- Etudiant: ☐
- Bac: ☐ série: [dropdown menu]
- Boursier: ☐ mention: [dropdown menu]
- Afficher: [button]
- créer: [button]

Below the form is a large empty rectangular area.



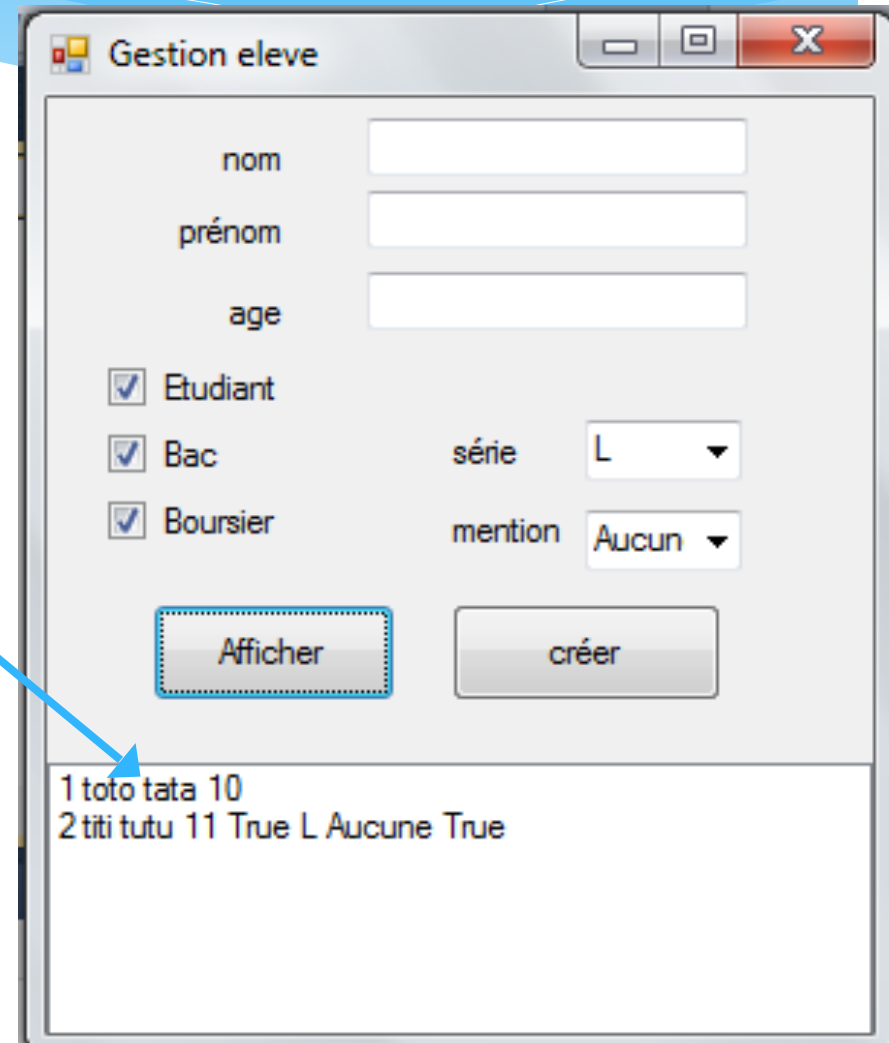
The screenshot shows a window titled "Gestion eleve" with a standard Windows-style title bar (minimize, maximize, close buttons). The form contains the following fields and controls:

- nom: titi
- prénom: tutu
- age: 11
- Etudiant: ☒
- Bac: ☒ série: L [dropdown menu]
- Boursier: ☒ mention: lucune [dropdown menu]
- Afficher: [button]
- créer: [button]

Below the form is a large empty rectangular area.

P00 - polymorphisme

- * Affichage suivant le type d'objet !



The screenshot shows a window titled 'Gestion eleve' with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains a form with the following fields:

- nom: text input field
- prénom: text input field
- age: text input field
- Etudiant: checkbox (checked)
- Bac: checkbox (checked)
- Boursier: checkbox (checked)
- série: dropdown menu (selected: L)
- mention: dropdown menu (selected: Aucun)

Below the form are two buttons: 'Afficher' (highlighted with a blue dashed border) and 'créer'.

At the bottom of the window, there is a list of objects displayed in a text area:

```
1 toto tata 10
2 titi tutu 11 True L Aucune True
```

A blue arrow points from the text '* Affichage suivant le type d'objet !' to the 'Afficher' button.

POO

- * C'est fini !
- * On touche à la fin du cours SI4 de S1
- * Questions ?