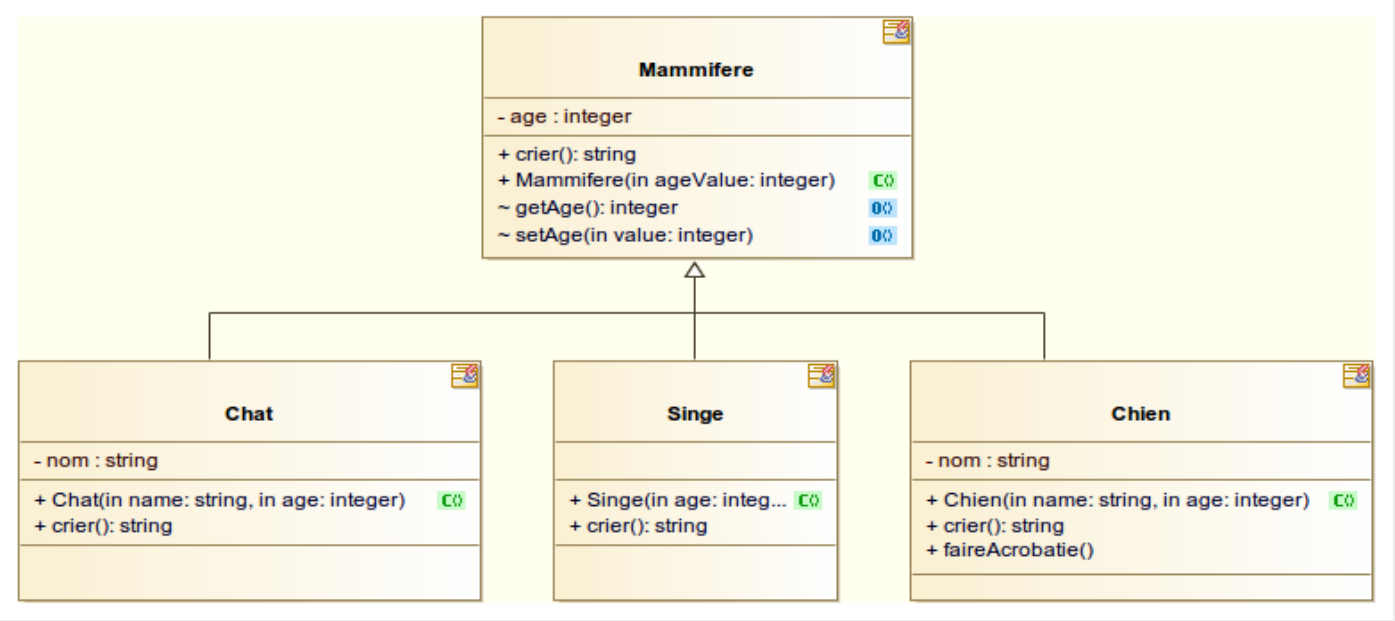
C# Synthèse

Les points clefs

Héritage

- Concepts POO
  - Permet de définir une famille d'objet qui ont un comportement semblable
  - Définit une relation : "est un« ou plutôt « se comporte comme un »
    - Un chat est un mammifère , une voiture est un véhicule



Héritage

- Filiation:
  - une classe dérivée est une spécialisation de la classe de base.
    - Héritage de l'état et des méthodes de l'ancêtre
    - Spécialisation de méthodes de la classe (réécriture)
    - Rajout de nouvelles méthodes (à éviter: voir SOLID LISKOV)
  - Attention: une classe ne peut dériver que d'une classe unique en C#
    - Pas d'héritage multiples (en C++ oui)
- Objectif visé
  - la « **réutilisabilité », la fiabilité du code, maintenance**
    - Mise en facteur de champs et de méthodes communes (invariantes dans le temps) pour un ensemble d'objet
      - => définition d'une classe de base et de classe dérivée (filiation)
    - **Manipulation indifférente des objets quelques que soit leur type**
      - **Concept polymorphe**

Héritage

- Exemple
  - Définition des classes

```
public class Mammifere
 {
// propriétés automatique
protected int Age {get;set;}
// constructeurs
public Mammifere()
{ Console.WriteLine("Ctor Mammifere 1"); }
public Mammifere(int age)
{
    Console.WriteLine("Ctor Mammifere 2");
    Age=age;
}
//methodes
public void Crier()
{Console.WriteLine("Raaaa");}
}
```

```
enum RaceChien { INCONNU, GOLDEN, CAIRN, DANDIE,
SHETLAND, DOBERMAN, LAB };


 class Chien : Mammifere
  {  // propiété auto
     public RACE Race { get; set; }
      // ctor 1
     public Chien(int age,RACE race):base(age)
     { Console.WriteLine("Ctor Chien 2");
    Race = race; ; }

      // specialisation
      // new est optionnel => evite warning
 compilateur
      //new précise l'intention du concepteur de
 spécialiser la méthode
     public new void Crier() {
 Console.WriteLine("Ouarfff"); }
  };
```

Héritage

- Exemple
  - Le code client

```
Mammifere unMamifere = new Mammifere(10);
 unMamifere.Crier();
Chien unChien = new Chien(RaceChien.GOLDEN, 5);
unChien.Crier();
```

1-Le ctor de Chien

2-Le ctor de mammifère

```
public Chien(int age,RACE race):base(age)
    { Console.WriteLine("Ctor Chien 2");
   Race = race; ; }
```

3-Le ctor de Chien Suite

Héritage

- La classe Object
  - Classe particulière
    - Elle constitue la racine de la hiérarchie des types
  - Toute vos classes héritent automatiquement de Object

Méthode de Object que vous pouvez spécialiser

	Nom	Description
	Equals(Object)	Détermine si l'objet spécifié est identique à l'objet actuel.
	Equals(Object, Object)	Détermine si les instances d'objet spécifiées sont considérées comme égales.
	Finalize	Autorise un objet à tenter de libérer des ressources et d'exécuter d'autres opérations de nettoyage avant qu'il ne soit récupéré par l'opération garbage collection.
	GetHashCode	Sert de fonction de hachage par défaut.
	GetType	Obtient le Type de l'instance actuelle.
	MemberwiseClone	Crée une copie superficielle de l'objet Object actuel.
	ReferenceEquals	Détermine si les instances de Object spécifiées sont identiques.
	ToString	Retourne une chaîne qui représente l'objet actif.

Héritage

```
using System;

// The Point class is derived from System.Object.
class Point
{
    public int x, y;

    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public override bool Equals(object obj)
    {
        // If this and obj do not refer to the same type, then they are not equal.
        if (obj.GetType() != this.GetType()) return false;

        // Return true if  x and y fields match.
        Point other = (Point) obj;
        return (this.x == other.x) && (this.y == other.y);
    }

    // Return the XOR of the x and y fields.
    public override int GetHashCode()
    {
        return x ^ y;
    }

    // Return the point's value as a string.
    public override String ToString()
    {
        return String.Format("({0}, {1})", x, y);
    }
```

Héritage

- Lecture de doc

Button, classe

.NET Framework 4.5 | Autres versions | Ce sujet n'a pas encore été évalué - Évaluez ce su

Représente un contrôle bouton Windows.

Hiérarchie d'héritage

System.Object
  System.MarshalByRefObject
    System.ComponentModel.Component
      System.Windows.Forms.Control
        System.Windows.Forms.ButtonBase
          System.Windows.Forms.Button

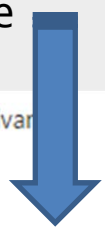
Espace de noms : System.Windows.Forms
Assembly : System.Windows.Forms (dans System.Windows.Forms.dll)

Syntaxe

C# | C++ | F# | VB

```
[ComVisibleAttribute(true)]
[ClassInterfaceAttribute(ClassInterfaceType.AutoDispatch)]
public class Button : ButtonBase, IButtonControl
```

Détails et exemples en ligne



Le type Button expose les membres suivan

- Constructeurs
- Propriétés
- Méthodes
- Événements
- Implémentations d'interface explicite

Polymorphisme

- Convertion de type UpCast
  - Objet manipulé au travers d'une référence type de base

```
Chien unChien2 =new Chien(5, RACE.GOLDEN);
Mammifere autreMammifere2= unChien2;  //Upcast
```

  - Cette convertion réussit toujours à condition de respecter la filiation des classes
    - Mot clé is et as pour valider la conversion sans exception

- Spécialisation sans mot clé virtual
  - Quelle est l'action produite par? `autreMammifere2.Crier()`
    - `Resultat: “Raaaa” et non “Ouarfff”`
  - Pourquoi considérer un chien comme un mammifère alors?
    - Réponse: il est utile de regrouper au sein d'un conteneur (une liste par exemple) des objets différents et de leur appliquer un traitement commun
      - Exemple: une liste de différents mammifères, la liste est balayée pour les faire crier

Polymorphisme

- Le couple virtual/override
  - L'objectif est de pouvoir appeler la méthode de substitution adéquate à partir d'une référence *de base pointant vers des objets de type classe dérivée*

```
public class Mammifere
{
//methodes
public virtual void Crier() { Console.WriteLine("Raaaa"); }
public virtual void Sous_Ordre_Especes() {
Console.WriteLine("Mammifere"); }
public void NbreDePattes() { Console.WriteLine("Inconnu"); }
}
class Chien : Mammifere
{
   public override void Crier() { Console.WriteLine("Ouarfff"); }
   public void Sous_Ordre_Especes() { Console.WriteLine("Canidés"); }
   public void NbreDePattes() { Console.WriteLine("Chien: 4 pattes"); }
};
```

```
Mammifere mam1;
mam1 = new Chien();
// mam1 pointe vers un chien Upcast OK
mam1.Crier();
mam1.Sous_Ordre_Especes();
mam1.NbreDePattes();
```

Ouarfff

Mammifere
Inconnu

```
 public class Mammifere
 {
//methodes
public virtual void Crier() { Console.WriteLine("Raaaa"); }
public virtual void Sous_Ordre_Especes() { Console.WriteLine("Mammifere"); }
public void NbreDePattes() { Console.WriteLine("Inconnu"); }
}
class Chat : Mammifere
{
    public override void Crier() { Console.WriteLine("Miaouu"); }
    public override void Sous_Ordre_Especes() { Console.WriteLine("Félidés"); }
    public  void NbreDePattes() { Console.WriteLine("Chat: 4 pattes"); }
};
```

```
Mammifere mam1 = new Chat(); // Upcast OK
mam1.Crier();
mam1.Sous_Ordre_Especes();
mam1.NbreDePattes();
```

Miaouu

Félidés

Inconnu

Polymorphisme

- Permet d'effectuer une action sera sélectionnée contextuellement en fonction de la classe d'objets à laquelle elle s'applique
  - Elle s'appuie sur la convertion UpCast
  - Sur le couple virtual/override
- *L'intérêt de cette approche est de pouvoir unifier un traitement appliqué à des objets de nature différente*

```
// On regroupe tout le monde sous le type de base
   List<Mammifere> maMenagerie=new  List<Mammifere>() ;
   maMenagerie.Add(new Chat());
   maMenagerie.Add(new Chien());
   maMenagerie.Add(new Mammifere()
   // Je fais crier tous le monde

   foreach (Mammifere mam in maMenagerie)
         mam.Crier();
```

⇦ **TRAITEMENT DE BASE APPLIQUE A DES OBJETS DIFFERENTS(MAIS APPARTENANT A LA MÊME CLASSE DE BASE)**

Miaouu
Ouarfff
Raaaa