

Héritage



Objectifs

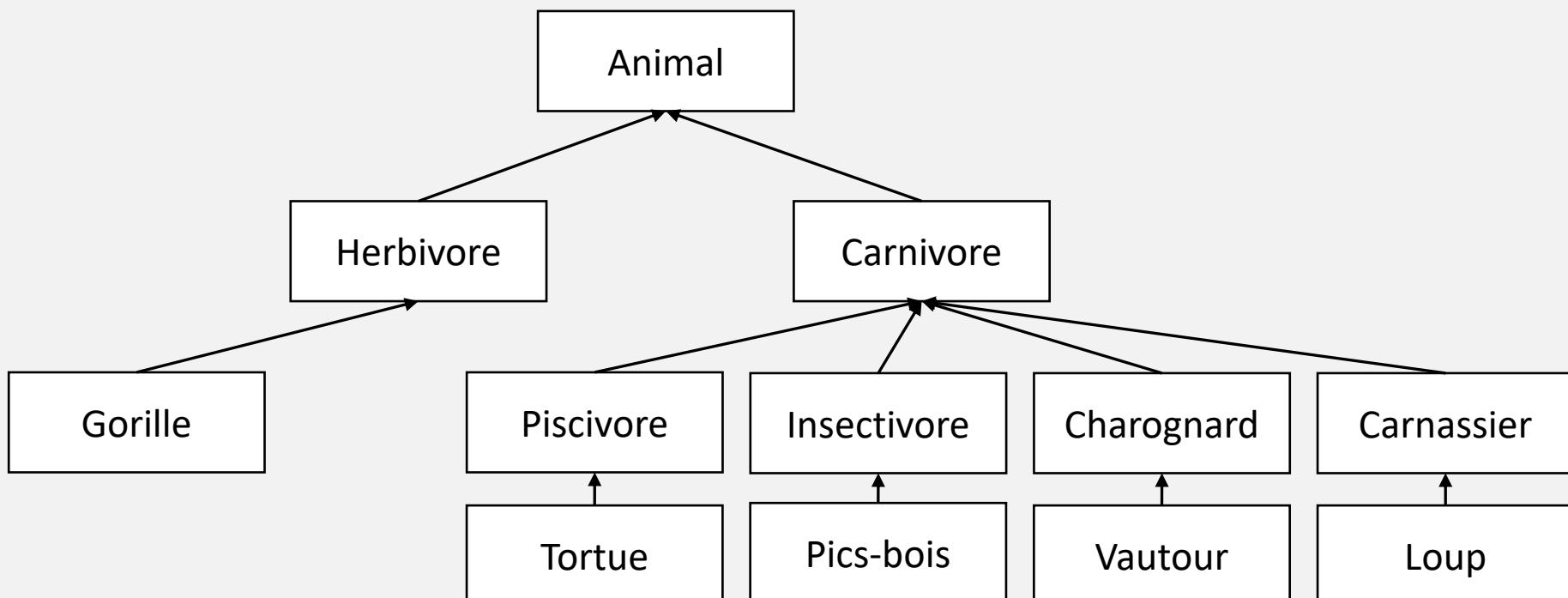
- Comprendre le principe d'héritage
- Apprendre le vocabulaire lié à l'héritage
- Revisiter les constructeurs

Héritage

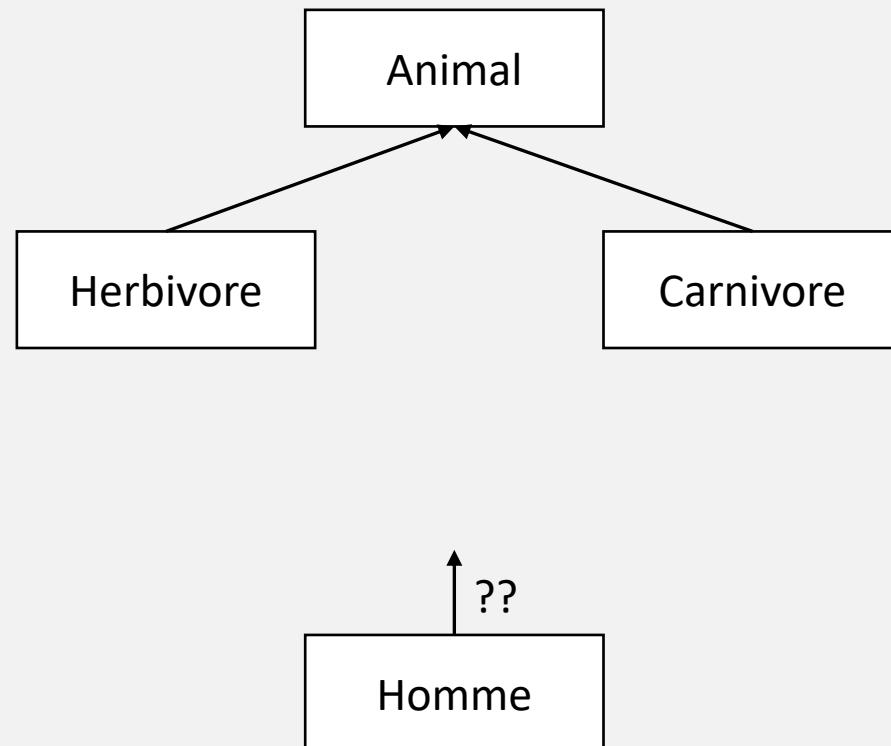
- En programmation orientée objet, l'héritage est un mécanisme qui permet, lors de la déclaration d'une nouvelle classe, d'y inclure les caractéristiques d'une autre classe. (Wikipédia)
- Les caractéristiques sont les données membres et les méthodes
- Certains langages permettent l'héritage de plusieurs classes comme en C++
- En C#, Java, etc., comme dans la plupart des langages modernes, l'héritage est simplifié en ne permettant que l'héritage simple
- Si vous n'explicitez aucun héritage pour une classe, elle hérite de la classe de base « Object »

Héritage

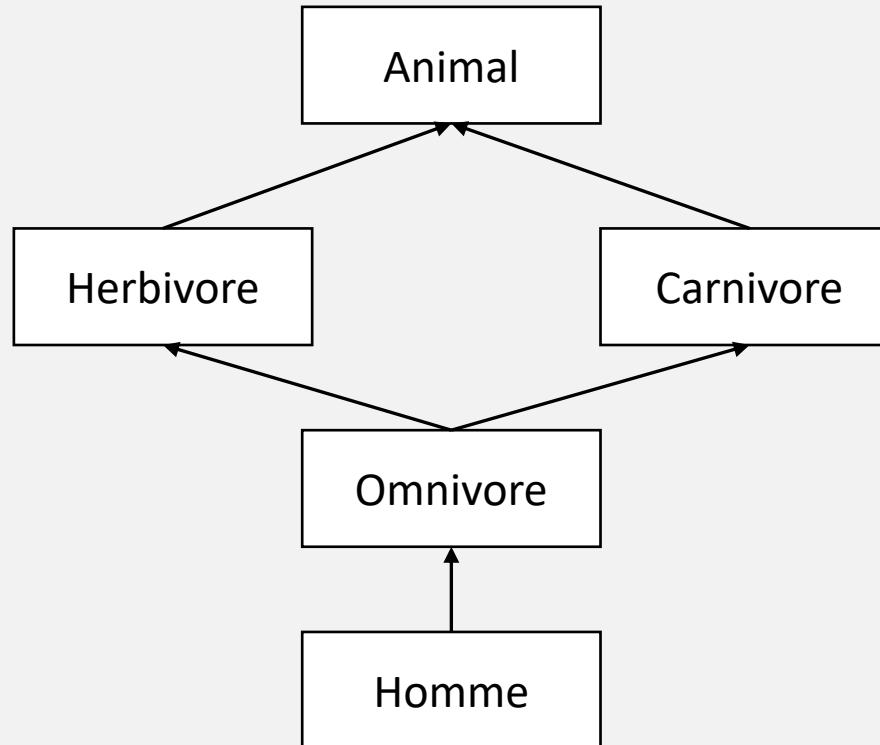
- L'association était caractérisée par la relation « a un / a des »
- L'héritage est caractérisé par la relation « est un » (dont il faut se méfier, cours POOII)



Héritage

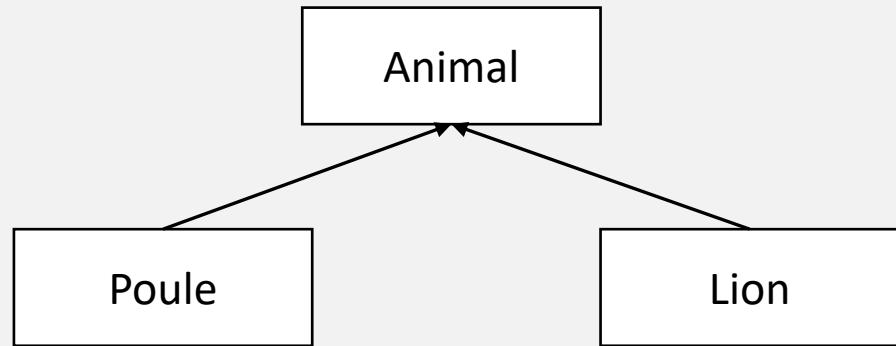


Héritage



=> Dans ce cours, nous ne traiterons pas l'héritage multiple (ici en diamant)

Héritage – Autre exemple avec les animaux



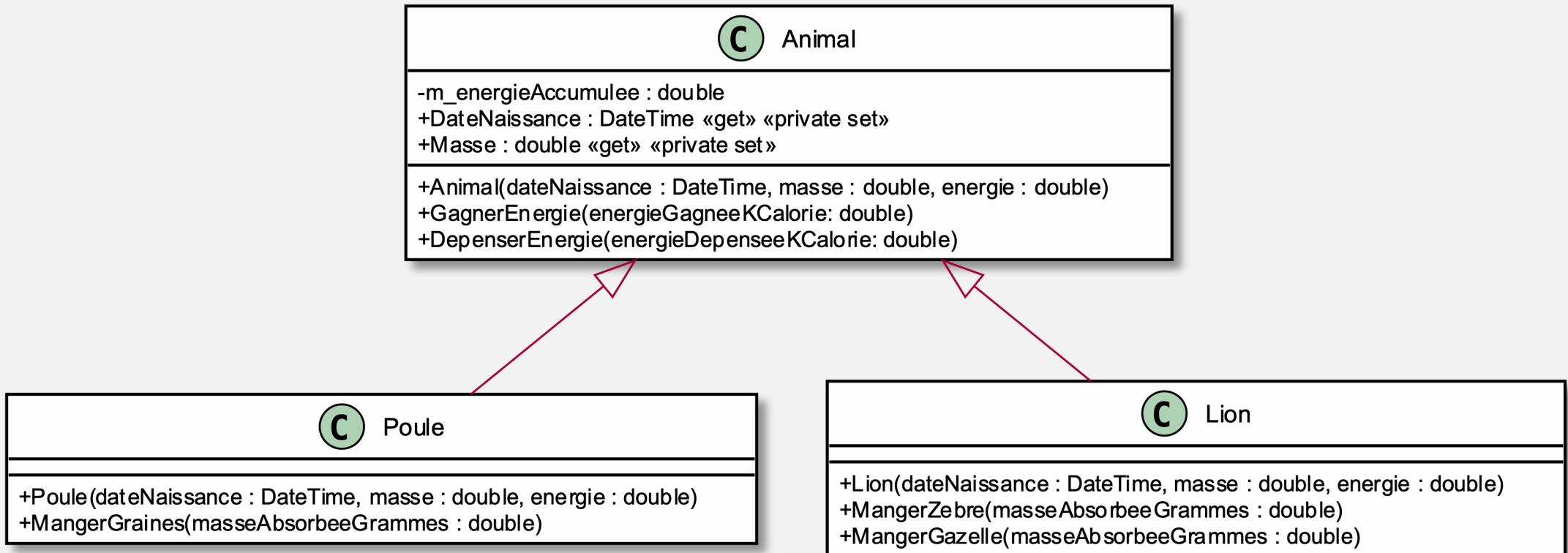
On veut représenter des animaux :

- Des poules
- Des lions

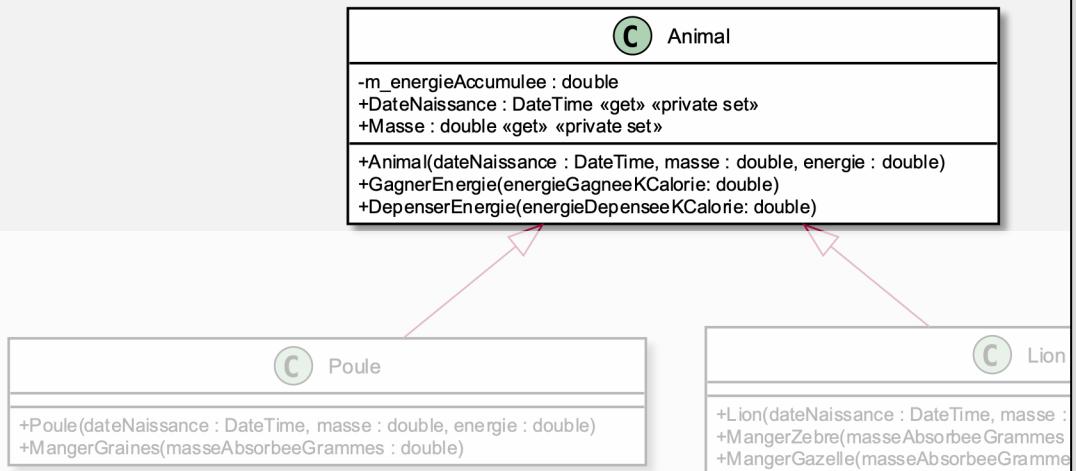
Les animaux ont une réserve d'énergie et une date de naissance. Ils peuvent gagner et dépenser de l'énergie.

Les poules mangent des graines de céréales.
Les lions mangent des zèbres et des gazelles

Héritage – Autre exemple avec les animaux – UML



Héritage – Autre exemple avec les animaux – C#



```
public class Animal
{
    private double m_energieAccumulee;
    public DateTime DateNaissance { get; private set; }
    public double Masse { get; private set; }

    public Animal(DateTime p_dateNaissance, double p_masse, double p_energie)
    {
        this.DateNaissance = p_dateNaissance;
        this.Masse = p_masse;
        this.m_energieAccumulee = p_energie;
    }

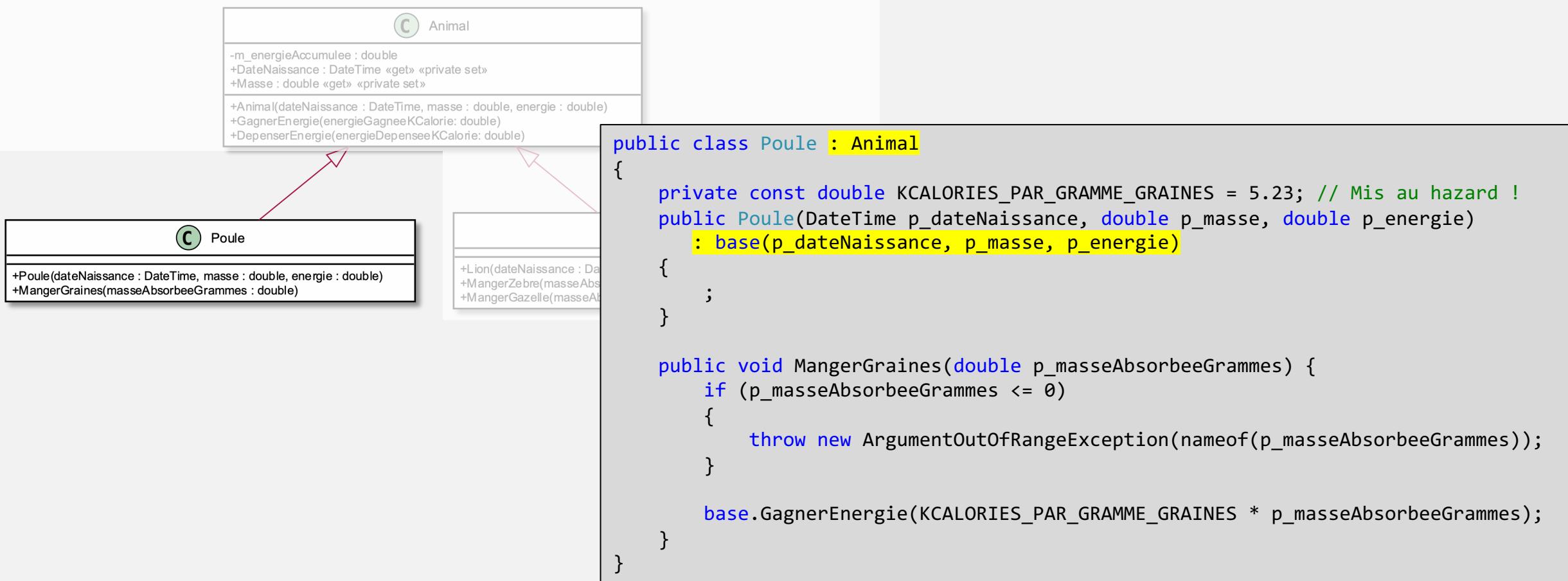
    public void GagnerEnergie(double p_energieGagneeKCalorie)
    {
        if (p_energieGagneeKCalorie <= 0)
        {
            throw new ArgumentOutOfRangeException(nameof(p_energieGagneeKCalorie));
        }

        this.m_energieAccumulee += p_energieGagneeKCalorie;
    }

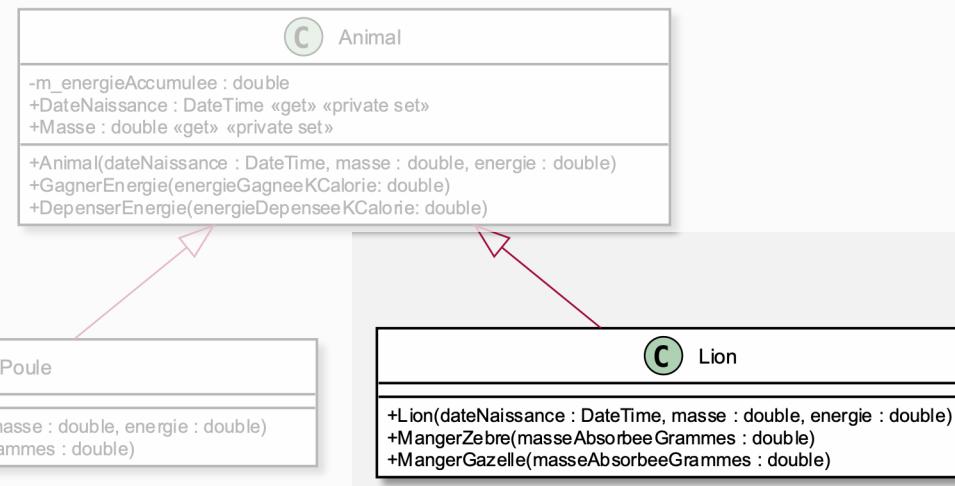
    public void DepenserEnergie(double p_energieDepenseeKCalorie)
    {
        if (p_energieDepenseeKCalorie <= 0)
        {
            throw new ArgumentOutOfRangeException(nameof(p_energieDepenseeKCalorie));
        }

        this.m_energieAccumulee -= p_energieDepenseeKCalorie;
    }
}
```

Héritage – Autre exemple avec les animaux – C#



Héritage – Autre exemple avec les animaux – C#



```
public class Lion : Animal
{
    public const double KCALORIES_PAR_GRAMME_GAZELLE = 2.5; // Mis au hazard !
    public const double KCALORIES_PAR_GRAMME_ZEBRE = 2.07; // Mis au hazard !
    public Lion(DateTime p_dateNaissance, double p_masse, double p_energie)
        : base(p_dateNaissance, p_masse, p_energie)
    {
        ;
    }

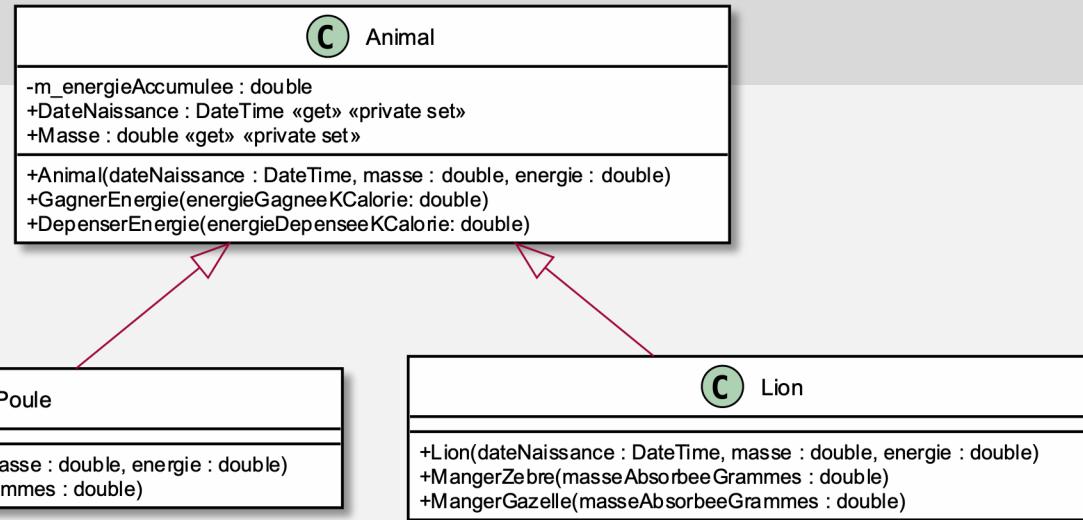
    public void MangerGazelle(double p_masseAbsorbeeGrammes)
    {
        if (p_masseAbsorbeeGrammes <= 0)
        {
            throw new ArgumentOutOfRangeException(nameof(p_masseAbsorbeeGrammes));
        }

        base.GagnerEnergie(KCALORIES_PAR_GRAMME_GAZELLE * p_masseAbsorbeeGrammes);
    }

    public void MangerZebre(double p_masseAbsorbeeGrammes)
    {
        if (p_masseAbsorbeeGrammes <= 0)
        {
            throw new ArgumentOutOfRangeException(nameof(p_masseAbsorbeeGrammes));
        }

        base.GagnerEnergie(KCALORIES_PAR_GRAMME_ZEBRE * p_masseAbsorbeeGrammes);
    }
}
```

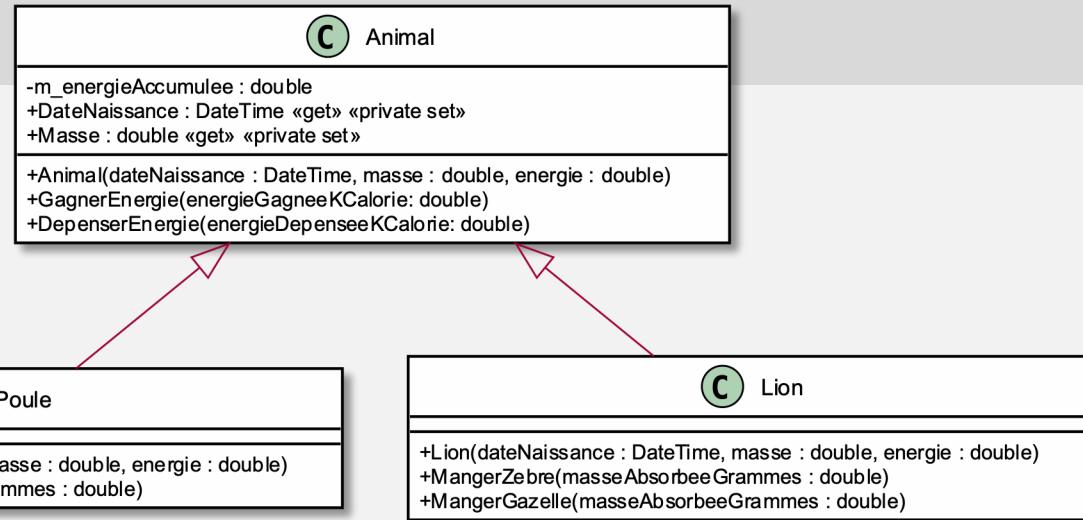
Héritage – Utilisation exemple avec les animaux – C#



```
Lion lion = new Lion(new DateTime(2010, 10, 1), 161, 12000);
lion.DepenserEnergie(1);
lion.GagnerEnergie(1);
lion.MangerGazelle(10000);
lion.MangerZebre(53000);
```

```
Poule poule = new Poule(new DateTime(2018, 3, 23), 3.3, 1600);
poule.DepenserEnergie(1);
poule.GagnerEnergie(1);
poule.MangerGraines(350);
```

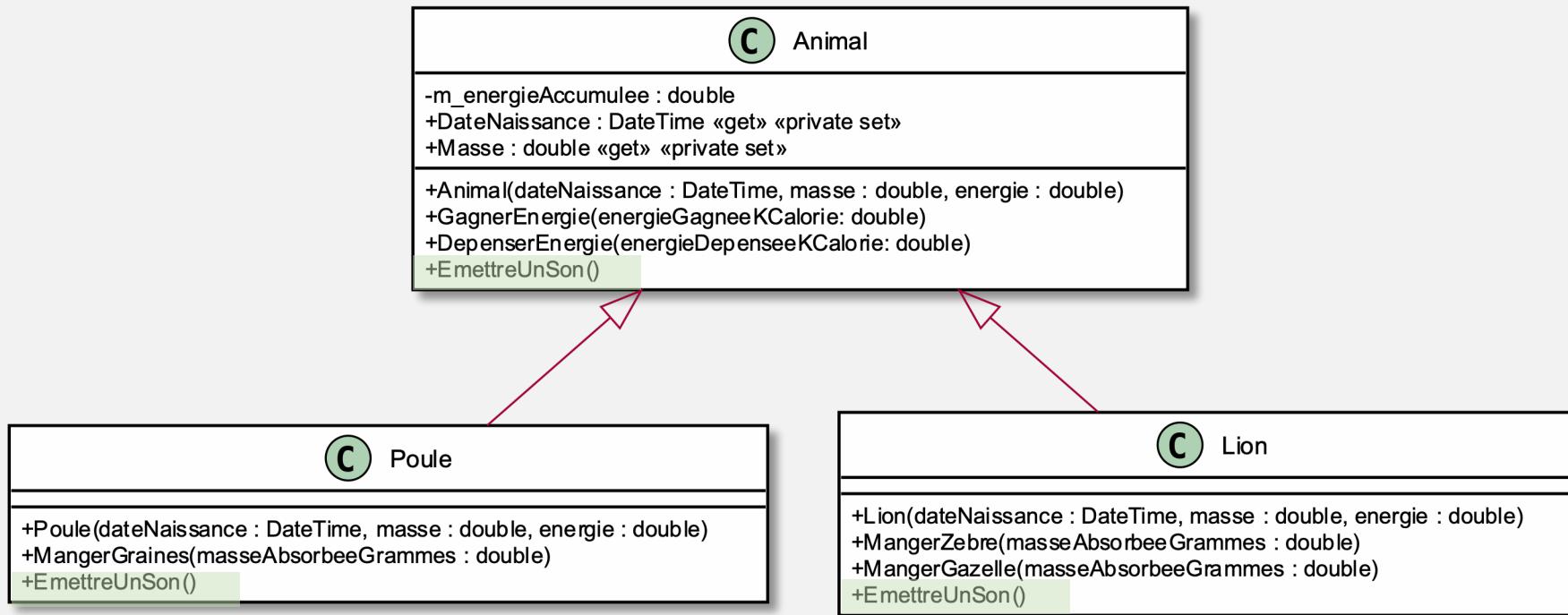
Héritage – Utilisation exemple avec les animaux – C#



```
Animal animal1 = lion;
Animal animal2 = poule;
|
|
List<Animal> listeAnimaux = new List<Animal>();
listeAnimaux.Add(lion);
listeAnimaux.Add(poule);
|
|
animal1.MangerGazelle(1223);
animal2.MangerGraines(12);
```

Héritage – Redéfinition de méthodes

- Comme nous l'avons fait pour les méthodes de la classe de base Objet, vous avez la possibilité de permettre la redéfinition de méthodes dans les classes filles



Héritage – Redéfinition de méthodes

- Pour rendre une méthode redéfinissable, il faut utiliser le modifier « `virtual` »

```
public class Animal
{
    // ...

    public virtual void EmettreUnSon()
    {
        // Nous verrons d'autres modificateurs pour éviter ce code
        ;
    }
}
```

Héritage – Redéfinition de méthodes

- Dans les classes filles, nous allons maintenant pouvoir utiliser le mot clef « `override` » afin de redéfinir des méthodes.

```
public class Lion : Animal
{
    // ...

    public override void EmettreUnSon()
    {
        // si besoin vous pouvez appeler la méthode de
        // la classe mère avec : base.EmettreUnSon();
        Console.Out.WriteLine("Grrrrrrr !");
    }
}
```

```
public class Poule : Animal
{
    // ...

    public override void EmettreUnSon()
    {
        // si besoin vous pouvez appeler la méthode de
        // la classe mère avec : base.EmettreUnSon();
        Console.Out.WriteLine("Cot cot !");
    }
}
```