Programmation objet – Cours 5
Héritage (1ère partie)

1. Un exemple d'héritage
2. Créer et initialiser un objet d'une classe étendue/héritée
3. Invoquer des méthodes d'une classe étendue
4. Accès protégé (protected)

Exemple de code redondant (répété)

```
public class Animal
{
    private float poids ;

    public Animal(float p)
    {
        this.poids = p ;
    }

    public float getPoids( )
    {
        return this.poids ;
    }

    public void affiche( )
    {
        System.out.println (this.poids) ;
    }
} // fin classe Animal
```

```
public class Mammifere
{
    private float poids ;
    private int nbMamelles ;
    public Mammifere(float p, int m)
    {
        this.poids = y ;
        this.nbMamelles = m ;
    }
    public float getPoids( )
    {
        return this.poids ;
    }
    public int getNbMamelles( )
    {
        return this.nbMamelles ;
    }
    public void affiche()
    {
        System.out.println (this.poids) ;
        System.out.println (this.mamelles) ;
    }
} // fin classe Mammifere
```

Exemple de code non redondant grâce à l'héritage

La classe *Mammifere* est une extension de la classe *Animal,*

```
public class Animal
{
    private float poids ;
    public Animal(float p)
    {
        this.poids = p ;
    }
    public float getPoids( )
    {
      return this.poids ;
    }
public void affiche( )
{
    System.out.println (this.poids) ;
}
}
```

```
public class Mammifere extends Animal
{
    private int nbMamelles ;
    public Mammifere(float p, int m)
    {
        super(p) ;
        this.nbMamelles = m ;
    }
    public int getNbMamelles( )
    {
      return this.nbMamelles ;
    }
    public void affiche( )
    {
        super.affiche( ) ;
        System.out.println (this.mamelles) ;
    }
} // fin classe Mammifere
```

Un classe étendue contient une partie implicite héritée et une partie explicite

```
private float poids ;

public Animal(float p)
{
    this.poids = p;
}

public float getPoids( )
{
  return this.poids ;
}

public void affiche( )
{
    System.out.println (this.poids) ;
}
```

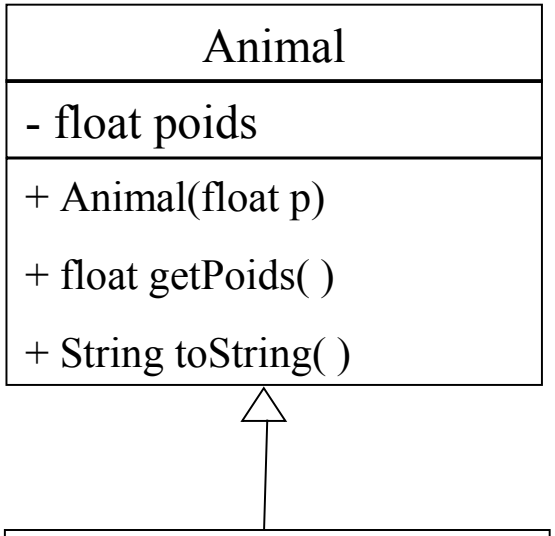
partie implicite héritée de *Animal*
(n'apparaît pas dans *Mammifere*)

```
public class Mammifere extends Animal
{
    private int nbMamelles ;

    public Mamifere(float p, int m)
    {
        super(p) ;
        this.nbMamelles = m ;
    }

    public int getNbMamelles( )
    {
      return this.nbMamelles ;
    }

    public void affiche( )
    {
        super.affiche( ) ;
        System.out.println (this.mamelles) ;
    }
}   // fin classe Mammifere
```

partie explicite spécifique à *Mammifere*

Caractéristiques de l'héritage

L'héritage permet de définir une nouvelle classe à partir d'une classe existante en ne spécifiant que ce qu'elle ajoute (on ne peut rien retirer)



(notation UML)

(partie héritée implicite)

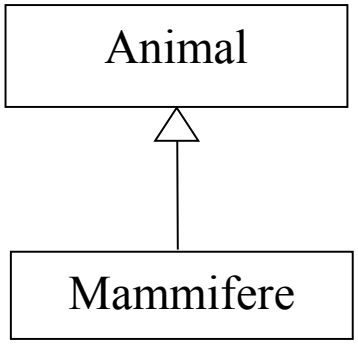
public class Mammifere **extends** Animal
{

(partie spécifique explicite)

}

Vocabulaire

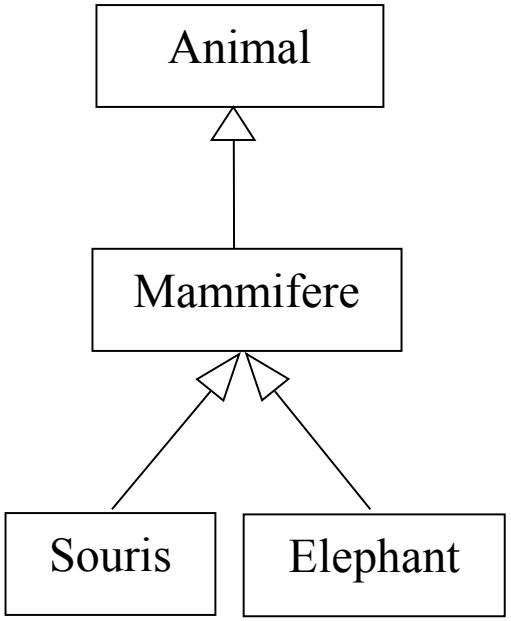
On dit de façon équivalente que :



- la classe *Mammifere* hérite de la classe *Animal*
- la classe *Mammifere* étend la classe *Animal*
- la classe *Mammifere* est une classe fille de la classe *Animal*
- la classe *Animal* est la classe mère de la classe *Mammifere*
- la classe *Mammifere* est une sous classe (ou classe dérivée) de la classe *Animal*
- la classe *Animal* est la sur-classe de la classe *Mammifere*
- la classe *Mamifere* est plus spécifique que la classe *Animale*
- la classe *Animal* est plus générale que la classe *Mammifere*

Intérêts de l'héritage

Réutilisation et factorisation



1 – Réutilisation : concevoir de nouvelles classes à partir de classes déjà correctement définies

⇩

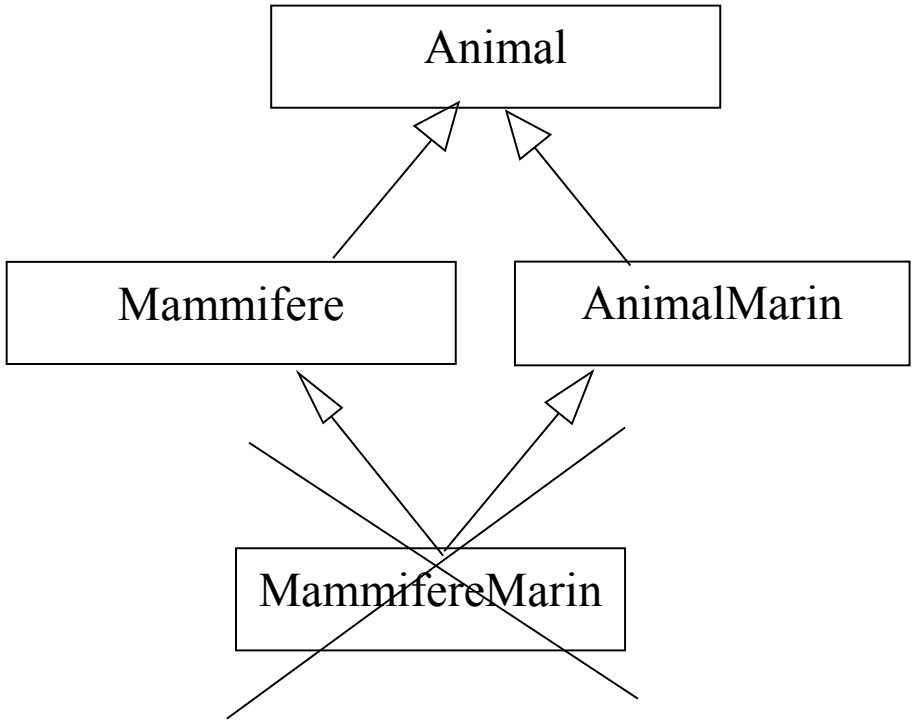
gain important de temps de développement + minimisation des risques d'erreurs

2 - Factorisation :mise en commun des variables et méthodes

⇩

gain de place

Règles d'héritage en java



Plusieurs classes peuvent hériter d'une même classe

mais

une même classe ne peut hériter de plusieurs classes

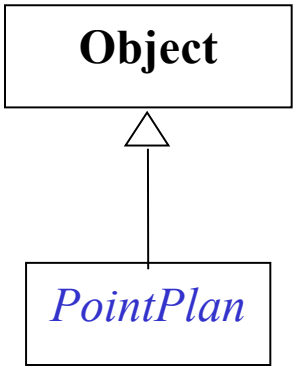
(pas d'héritage multiple)

Classe non extensible

Une classe précédée du mot-clé **final** ne peut être étendue

```
public final class PointPlan
{
    ...
}
// fin classe PointPlan
```

extends *Object* est implicite




```
public class PointColore extends PointPlan
{
    private int couleur ;
    ...
}
// fin classe Souris
```

rejet du compilateur : *PointPlan* est une classe finale

Objet d'une classe étendue

Un objet (instance) d'une classe étendue contient :

les variables d'instance héritées + les variables d'instance spécifiques

```
public class Animal
{
   private float poids ;
   …
} // fin classe Animal
```

```
public class Mammifere extends Animal
{
      private int nbMamelles ;
      …
} // fin classe Mammifere
```

```
{   Mammifere m1 = new Mammifere(30f, 6) ;
}   Mammifere m2 = new Mammifere(50f, 4) ;
```

Les variables d'instances héritées sont initialisées par un constructeur de la classe mère

```
public class Mammifere extends Animal
{
        private int nbMamelles ;

         public Mammifere(float p, int m)
         {
              super(p) ;
                        this.nbMamelles = m ;
          }
      ...
```

```
public class TestMammifere
 {
     public static void main(String[ ] args)
    {
   Mammifere m = new Mammifere(30, 6) ;

    } // fin main

 } // fin classe
```

super(p) ⇔*Animal(p)* :
appelle du constructeur de la classe mère :
obligatoirement en 1ère instruction du constructeur.
En cas d'omission il y a un appel implicite *super( )*
(provoque une erreur de compilation si la classe mère n'a pas de constructeur sans argument)

:Mammifere	
- poids	30f
- nbMamelles	6

} // fin classe Mammifere

Le constructeur de la classe mère invoqué est choisi en fonction de sa signature

```
public class Animal
{
    private float poids ;

    public Animal(float p)
    {
        this.poids = p ;
    }

    public Animal ( )
    {
        this.poids = 50f ;
    }

} // fin classe Animal
```

```
public class Mammifere extends Animal
{
    private int nbMamelles ;

    public Mamifere(float p, int m)
    {
        super(p) ;
        this.nbMamelles = m ;
    }

    public Mamifere(int m)
    {
        super( ) ;
        this.nbMamelles = m ;
    }
} // fin classe Mammifere
```

:Mammifere
- poids 30f
- nbMamelles 6

:Mammifere
- poids 50f
- nbMamelles 4

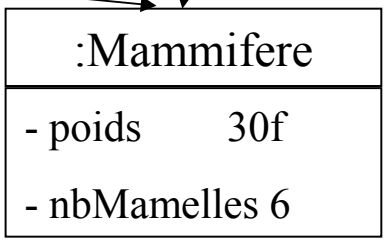
```
{   Mammifere m1 = new Mammifere(30f, 6) ;
}   Mammifere m2 = new Mammifere(4) ;
```

Invoquer une méthode spécifique

(Une méthode spécifique n'existe pas dans la classe mère)

```
public class Mammifere extends Animal
{
        private int nbMamelles ;

        public Mamifere(float p, int m)
        {
             super(p) ;
             this.nbMamelles = m ;
        }

       public int getNbMamelles( )
       {
          return this.nbMamelles ;
       }


 } // fin classe Mammifere
```

```
public class TestMammifere
{
     public static void main(String[] args)
     {
          Mammifere m1 = new Mammifere(30f, 6) ;

          int n = m1.getNbMamelles( ) ;
     }
 } // fin classe TestMammifere
```



Invoquer une méthode héritée

```
public class Animal
{
   private float poids ;
     ...
   public float getPoids( )
   {
     return this.poids ;
   }
} // fin classe Animal
```

```
public class Mammifere extends Animal
{
      private int nbMamelles ;

      public Mamifere(float p, int m)
       {
            super(p) ;
            this.nbMamelles = m ;
       }

} // fin classe Mammifere
```

```
public class TestMammifere
{
     public static void main(String[] args)
     {
          Mammifere m1 = new Mammifere(30f, 6) ;

           int p = m1.getPoids( ) ;
     }
} // fin classe TestMammifere
```

méthode héritée de Animal

:Mammifere
- poids 30f
- nbMamelles 6

Redéfinir une méthode

Écrire une méthode spécifique de même signature qu'une méthode héritée

```
public class Animal
{
    public void quiSuisJe( )
    {
        System.out.println("je suis un animal") ;
    }
    …

  }   // fin classe Animal
```

```
public class Mammifere extends Animal
{
    …
    public void quiSuisJe( )
    {
        System.out.println ("je suis un mammifère") ;
    }

  }   // fin classe Mammifere
```

La classe *Mammifere* possède 2 méthodes *quiSuisJe( )* de même signature :

on dit que *quiSuisJe( )* a été redéfinie

Redéfinir ≠ surcharger

(surcharger = écrire une méthode de même nom mais de signature différente)

Invoquer une méthode redéfinie

C'est toujours la méthode appartenant à la classe fille qui est invoquée

```
public class Animal
{
    public void quiSuisJe( )
    {
        System.out.println("Je suis un animal") ;
    }
}   // fin classe Animal
```

```
public class Mammifere extends Animal
 {  …
   public void quiSuisJe( )
   {
       System.out.println ("Je suis un mammifère") ;
   }
} // fin classe Mammifere
```

```
public class TestMammifere
{
    public static void main(String[] args)
    {
        Mammifere m1 = new Mammifere(30f, 6) ;

        m1.quiSuisJe( ) ;
    }

} // fin classe TestMammifere
```

c'est la méthode *quiSuisJe( )* de la classe *Mammifere* qui est invoquée.

Affichage : "je suis un mammifère"

La méthode héritée *quiSuisJe( )* est masquée par sa redéfinition

Invoquer une méthode masquée

```
public class Animal
{
      public String toString( )
      {
         return ("poids = " + this.poids) ;
      }
 } // fin classe Animal
```

```
public class Mammifere extends Animal
{    …
    public String toString( )
    {
        return (super.toString( ) +


              ", nb mamelles = " +
                  this.nbMammelles) ;
     }
} // fin classe Mammifere
```

```
public class TestMammifere
{
    public static void main(String[] args)
    {
       Mammifere m1 = new Mammifere(30f, 6) ;

       System.out.println(m1) ;
    }
} // fin classe TestMammifere
```

super.toString() appelle la méthode *toString( )* héritée de la classe mère (Animal).

Affichage : "poids = 30, nb mamelles = 6"

Etendre une classe pour redéfinir des méthodes

Il n'y a aucune obligation à ajouter de nouvelles variables et/ou méthodes dans une classe étendue
(on pourrait même ne rien changer)

On étend parfois une classe pour simplement redéfinir certaines de ses méthodes (et ainsi changer le comportement de ces méthodes dans la classe étendue)

Les variables privées héritées sont inaccessibles dans la classe fille

```
public class Animal
{
    private float poids ;

    …

} // fin classe Animal
```

```
public class Mammifere extends Animal
{
    private int nbMamelles ;

    public void affiche ()
    {
        System.out.println (this.poids + " this.nbMamelles) ;
    }

    …

} // fin classe Mammifere
```

rejet du compilateur : *poids* est une variable privée (*private*)

protected : un accès intermédiaire entre private et public

Tout membre (variable ou méthode) d'une classe **X** ayant l'accès **protected** est considéré :

- publique (**public**) :
  - Depuis toute classe descendant de **X**
  - depuis toute classe située dans le même paquetage que **X**
- privé (**private**) dans tous les autres cas

```
public class Animal
{
    protected float poids ;

  // fin classe Animal
}
```

```
public class Mammifere extends Animal
{
    private int nbMamelles ;
    public void affiche ()
    {
        System.out.println (this.poids + " this.nbMamelles) ;
    }
}  // fin classe Mammifere
```

ok