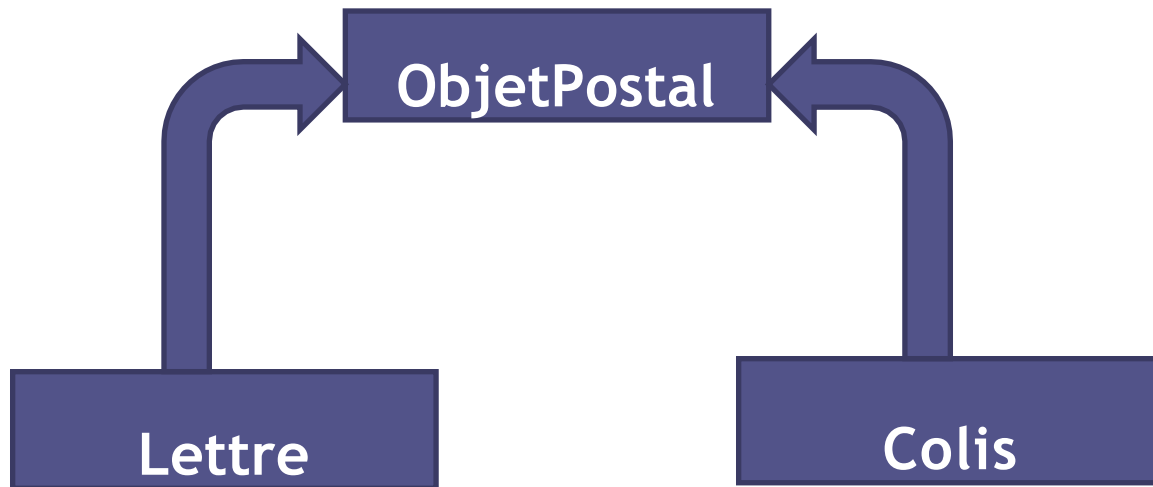




# **POO : Séance 6**

## **Héritage (Episode 3)**

# Rappel



# Classe ObjetPostal

```
public class ObjetPostal
{
    //attributs
    private int poids;
    private boolean recommande;
    private float tarif;
    //méthodes
    public ObjetPostal (int lePoids, float leTarif)
    {
        poids = lePoids;
        recommande = false;
        tarif = leTarif;
    }
    public int Get_poids()
    {
        return poids;
    }
    public float quelTarif()
    {
        return tarif;
    }
}
```

# Classe Lettre

```
public class Lettre extends ObjetPostal
{
    //attributs
    private boolean urgent;
    //méthodes
    public Lettre (int lePoids, float leTarif)
    {
        super(lePoids, leTarif);
        urgent = false;
    }
    public void Set_Urgent()
    {
        urgent = true;
    }

    public float quelTarif()
    {
        If (urgent) return (super. quelTarif() * 2.0);
        else return (super. quelTarif() );
    }
}
```

# Classe Colis

```
public class Colis extends ObjetPostal
{
    //attributs
    private int volume;
    //méthodes
    public Colis(int lePoids, float leTarif, int leVolume)
    {
        super(lePoids, leTarif);
        volume = leVolume;
    }
    public boolean grand()
    {
        if ( volume > 500 )
            return true;
        else
            return false;
    }
    public float quelTarif()
    {
        If (grand()) return (super. quelTarif() * 5.0);
        else return (super. quelTarif() * 2.0);
    }
}
```

# Polymorphisme des méthodes

- ❖ La surcharge (polymorphisme statique) consiste à proposer différentes signatures de la même méthode.
- ❖ La redéfinition (polymorphisme dynamique) ne se produit que dans l'héritage d'une classe par redéfinition de la méthode mère avec une méthode fille.

# Polymorphisme des objets

```
public class TestPostal  
{
```

```
    public static void main (String [] args)
```

```
    {
```

```
        ObjetPostal O1 = new ObjetPostal (400,2.0);
```

```
        ObjetPostal O2 = new Lettre (250,1.0);
```

```
        ObjetPostal O3 = new Colis(2000,5.0,600);
```

```
    }
```

```
}
```

Appel du constructeur  
de la classe ObjetPostal

Appel du constructeur  
de la classe Lettre

Appel du constructeur  
de la classe colis

O1 est un ObjetPostal

L'objet postal O2 est une Lettre : **Une lettre est un objet postal**

L'objet postal O3 est un Colis : **Un colis est un objet postal**

```
Lettre O4 = new objetPostal (250,1.0);
```




**Faux! Un objetPostal n'est pas obligatoirement une lettre**

# Polymorphisme des objets

```
System.out.println(" Le tarif de O1 est " + O1.quelTarif());
```


Invocation de la méthode  
quelTarif() de la classe  
ObjetPostal



```
O2. Set_Urgent();
```

```
System.out.println(" Le tarif de O2 est " + O2.quelTarif());
```

Invocation de la méthode  
quelTarif redéfinie dans la  
classe Lettre



```
System.out.println(" Le tarif de O3 est " + O3.quelTarif());
```

Invocation de la méthode  
quelTarif redéfinie dans la  
Classe Colis



```
}
```

**Le tarif de O1 est 2.0**

**Le tarif de O2 est 1.0**

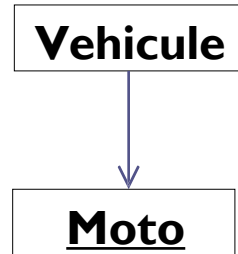
**Le tarif de O3 est 25.0**



# Polymorphisme des objets

- Le polymorphisme (poly-morph) est la capacité pour une entité de prendre plusieurs formes.
- Le polymorphisme est lié à une distinction entre la compilation et l'exécution. Lors de la compilation, une référence a le type de sa déclaration. A l'exécution, elle prend le type de l'objet créé et instancié.
- Ceci est réalisé via l'héritage : C'est au moment de l'exécution que l'on sélectionne la version de l'opération adaptée à une certaine occurrence de type.
- Le polymorphisme peut être vu comme la capacité de choisir dynamiquement la méthode qui correspond au type réel de l'objet.

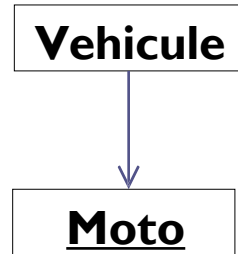
# Polymorphisme : Exemple I



- Honda, instance de la classe Moto aura comme type initial Moto.
- Une Moto possède par héritage le type Véhicule. Une moto est donc une véhicule.
- On peut donc écrire:

```
Moto Honda = new Moto();  
Vehicule honda = new Moto();
```

# Polymorphisme : Exemple I



- Il est possible de forcer le programme à voir un objet comme un type différent de son type initial, c'est le transtypage ou cast.
- Il y a transtypage implicite depuis la classe fille vers la classe mère :
  - Une Moto est implicitement de type Véhicule.

# Polymorphisme : Exemple 2

- On veut coder une classe Zoo qui aura plusieurs cages permettant d'accueillir différents types d'animaux .
- La Classe Animal est la classe mère des classes Lion et Singe

// Classe de base : Animal

```
public class Animal
{
    protected String nom;
    protected int age;
    public Animal(String nom, int age) {...}
    public String getNom() {return nom ;}
}
```

```
public class Lion extends Animal {

    public Lion(String nom, int age) {
        super(nom, age);
    }
}
```

```
public class Singe extends Animal {

    public Singe(String nom, int age) {
        super(nom, age);
    }
}
```

# Exemple 2 : Surcharger la méthode Accueillir

- Il faut implémenter une classe Cage permettant de contenir des animaux.
- On veut définir la méthode accueillir dans la classe cage permettant d'afficher l'animal accueilli.

```
// Classe Cage
public class Cage {

    // Méthode pour accueillir un Singe
    public void accueillir(Singe singe)
    {
        System.out.println("La cage accueille le singe : " + singe.getNom());
    }

    // Méthode pour accueillir un Lion
    public void accueillir(Lion lion)
    {
        System.out.println("La cage accueille le lion : " + lion.getNom());
    }
}
```

# Exemple 2 : Surcharger la méthode Accueillir

- La surcharge de la méthode accueillir est une très mauvaise solution.
- Si une nouvelle espèce animale doit être prise en compte, il faudra modifier le code de la classe Cage et ajouter une méthode accueillir avec un paramètre de type la nouvelle espèce à ajouter.
- Il faut implémenter autant de méthodes accueillir qu'il y a de races d'animaux.

## Exemple 2 : Définir une méthode polymorphe 'accueillir'

- La bonne solution consiste à utiliser le polymorphisme, en implémentant une méthode accueillir générique pour tous les animaux.
- Son paramètre étant de type Animal on pourra l'appeler avec une référence de Lion ou de Singe.

```
public class Cage
{
    public void accueillir(Animal animal)
    {
        System.out.println("La cage accueille l'animal : " + animal.getNom());
    }
}
```

## Exemple 2 : Définir une méthode polymorphe 'accueillir'

- La méthode accueillir prend un paramètre de type Animal. N'importe quel objet qui hérite de Animal peut être passé à la méthode et la méthode s'adapte au type de l'animal grâce au polymorphisme.
- Cela permet une grande flexibilité dans le code. Par exemple, si on ajoute la classe Elephant qui hérite de Animal, la méthode accueillir pourra également l'accepter sans modification.



## Exemple 2 : Classe ZOO

```
public class ZOO
{
    public static void main(String[] args)
    {
        // Création d'un Singe et d'un Lion

        Singe singe = new Singe("Titi");
        Lion lion = new Lion("Simba");

        // Création de la cage
        Cage cage = new Cage();

        // Utilisation de la même méthode accueillir pour différents types d'animaux
        cage.accueillir(singe); // La cage accueille le singe Titi
        cage.accueillir(lion);  // La cage accueille le lion Simba
    }
}
```

# Exemple 3 : Polymorphisme

```
class Animal {  
    void deplacer() {  
        System.out.println("Je bouge");  
    }  
  
class Chien extends Animal {  
    void deplacer() {  
        System.out.println("Je marche");  
    }  
}  
  
class Oiseau extends Animal {  
    void deplacer(){  
        System.out.println("Je vole");  
    }  
}  
  
class Pigeon extends Oiseau {  
    void deplacer() {  
        System.out.println("Je vole et en plus ... sur les passants");  
    }  
}
```

- Sur toutes ces classes, on peut appeler la méthode `deplacer()`.
- Le polymorphisme permet alors d'appeler la bonne méthode selon le type d'objet

# Exemple 3 : Polymorphisme

```
public static void main(String[] args) {  
  
    Animal a1 = new Animal();  
    Animal a2 = new Chien();  
    Animal a3 = new Pigeon();  
  
    a1.deplacer();  
    a2.deplacer();  
    a3.deplacer();  
}
```

## Exécution :

Je bouge

Je marche

Je vole et en plus ... sur les passants