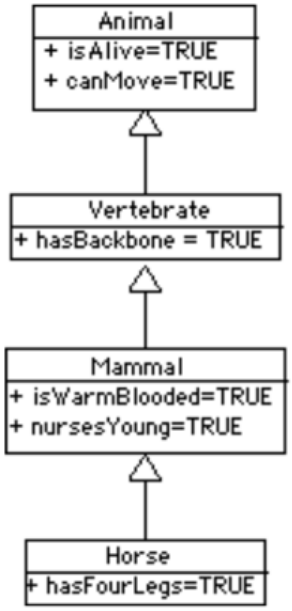
L'Héritage dans Java

L'héritage est un concept fondamental de la programmation orientée objet (POO) en Java. Il permet de créer de nouvelles classes basées sur des classes existantes, favorisant ainsi la réutilisabilité et l'extensibilité du code. Voici une vue d'ensemble de l'héritage en Java :

1. Concepts de Base



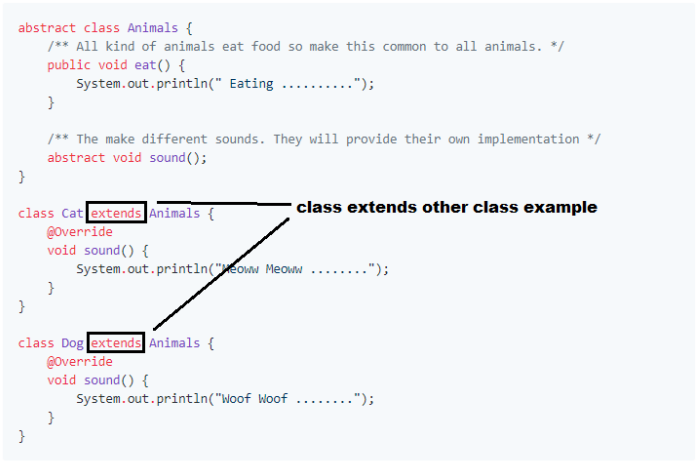
Classe Parent et Classe Enfant

- **Classe Parent (ou Superclasse)** : La classe dont les propriétés et les méthodes sont héritées.
- **Classe Enfant (ou Sous-classe)** : La classe qui hérite de la superclasse.

Mots-Clés

- **extends** : Utilisé pour indiquer qu'une classe hérite d'une autre.

2. Syntaxe de Base



Pour définir une classe enfant qui hérite d'une classe parent, on utilise le mot-clé `extends` :

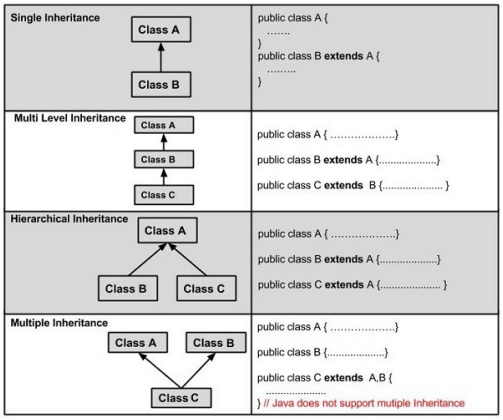
```
class Parent {
    // Propriétés et méthodes de la classe Parent
    public void afficherMessage() {
        System.out.println("Message de la classe Parent");
    }
}

class Enfant extends Parent {
    // Propriétés et méthodes de la classe Enfant
    public void afficherMessageEnfant() {
        System.out.println("Message de la classe Enfant");
    }
}

public class Main {
    public static void main(String[] args) {
        Enfant enfant = new Enfant();
        enfant.afficherMessage(); // Héritée de la classe Parent
```

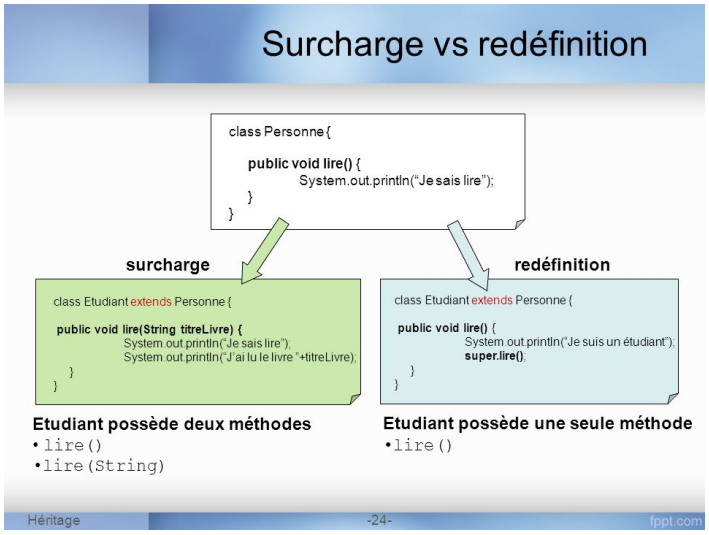
```
        enfant.afficherMessageEnfant(); // Méthode propre à la classe Enfant
    }
}
```

3. Types d'Héritage



Java supporte principalement l'héritage simple (une classe enfant hérite d'une seule classe parent). L'héritage multiple (une classe héritant de plusieurs classes) n'est pas directement supporté, mais peut être simulé en utilisant les interfaces.

4. Surcharge et Redéfinition

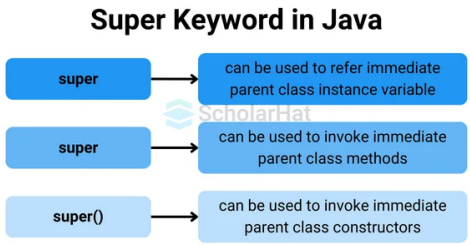


- **Surcharge (Overloading)** : Permet d'avoir plusieurs méthodes avec le même nom mais avec des paramètres différents.
- **Redéfinition (Overriding)** : Permet à une classe enfant de fournir une implémentation spécifique d'une méthode qui est déjà définie dans sa superclasse.

```
class Parent {
    public void afficherMessage() {
        System.out.println("Message de la classe Parent");
    }
}

class Enfant extends Parent {
    @Override
    public void afficherMessage() {
        System.out.println("Message de la classe Enfant");
    }
}
```

5. Utilisation du mot-clé `super`



Le mot-clé `super` est utilisé pour appeler le constructeur ou les méthodes de la superclasse :

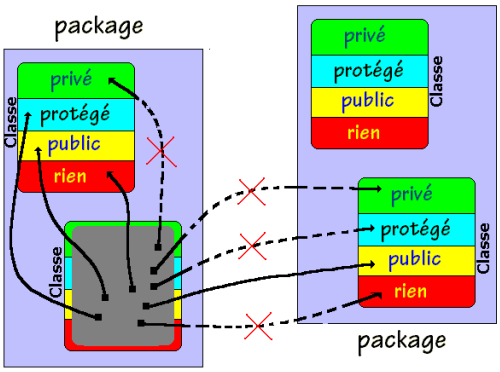
```
class Parent {
    public Parent() {
        System.out.println("Constructeur de la classe Parent");
    }

    public void afficherMessage() {
        System.out.println("Message de la classe Parent");
    }
}
```

```java
class Enfant extends Parent {
    public Enfant() {
        super(); // Appel au constructeur de la classe Parent
        System.out.println("Constructeur de la classe Enfant");
    }

    @Override
    public void afficherMessage() {
        super.afficherMessage(); // Appel à la méthode afficherMessage() de la classe
        System.out.println("Message de la classe Enfant");
    }
}
```

6. Héritage et Visibilité



Les membres d'une classe (variables et méthodes) ont différents niveaux d'accès :

- **public** : Accessible de partout.
- **protected** : Accessible dans la même classe, les sous-classes et les classes du même package.
- **private** : Accessible uniquement dans la classe où ils sont définis.
- **default** (pas de modificateur) : Accessible dans le même package.

7. Héritage et Constructeurs

```java
class Super {
    String s;
    public Super(String s) {
        this.s = s;
        System.out.println("Super s");
    }
}

public class Sub extends Super {
    int x = 200;
    public Sub(String s) {
        super(s);
    }


    public static void main(String[] args){
        Sub s = new Sub("a");
    }
}
```

Les constructeurs ne sont pas hérités, mais la classe enfant peut appeler le constructeur de la classe parent à l'aide du mot-clé `super`.

Conclusion

L'héritage en Java est un outil puissant qui permet de créer des hiérarchies de classes et de réutiliser du code. Il est essentiel de comprendre comment l'utiliser efficacement pour écrire un code propre, maintenable et extensible.