

Module M15

La programmation orientée objet JAVA

Chapitre 3

Programmation Orientée Objets avec JAVA

Encapsulation

Héritage

Polymorphisme

Programmation Orientée Objets avec JAVA

Plan du chapitre

1. Concept N2: encapsulation
2. Concept N3: héritage
3. Concept N°4 : polymorphisme

La programmation orientée objet

Les principaux concepts de la Programmation Orientée Objet:

1. Classe et Objet
2. Encapsulation (Accessibilité)
3. Héritage
4. Polymorphisme

Concept N° 2

Encapsulation

Encapsulation

Dans cette étape nous allons traiter les concepts suivants:

- Visibilité des membres: privé et protected
- Getters et Setters

NB: Durant cette partie, nous allons utiliser la visibilité *private* des attributs

Encapsulation

Dans cette étape nous allons traiter les concepts suivants:

- Visibilité des membres: privé et protected
- Getters et Setters

NB: Durant cette partie, nous allons utiliser la visibilité *private* des attributs

Visibilité des membres: Tests et conclusions

- Dans la classe point , modifier le mot clé public par private de tous ses éléments puis compiler et exécuter le programme principal

```
Program.java  Point.java
1 package com.setp1;
2
3 public class Point {
4     private double abscisse;
5     private double ordonnee;
6     private String couleur;
7     private Point(){
8     }
9     private Point(double abscisse, double ordonnee){
10         this.abscisse=abscisse;
11         this.ordonnee=ordonnee;
12     }
13     private Point(double abscisse, double ordonnee,String couleur){
14         this.abscisse=abscisse;
15         this.ordonnee=ordonnee;
16         this.couleur=couleur;
17     }
18     private void mon_etat()
19     {
20         System.out.println ("mon état:");
21         System.out.println("(" +abscisse+ ", "+ordonnee+") "+couleur );
22     }
23     private void maj_etat(){
24         abscisse=0;
25         ordonnee=0;
26         couleur="noir";
27     }
28     private void maj_etat(double abscisse,double ordonnee){
29         this.abscisse=abscisse;
30         this.ordonnee=ordonnee;
31         this.couleur="noir"; //faites this --> this.abscisse=
```

```
Program.java  Point.java
1 package com.setp1;
2
3 public class Program {
4     public static void main(String[] args) {
5         Point p;
6         p = new Point();
7         Point p2=new Point();
8         Point p3=new Point(3,3);
9         Point p4=new Point(4,4,"Bleu");
10        p.ordonnee = 0;
11        p.couleur="verte";
12        System.out.println("avant changement de l'état");
13        p.mon_etat();//l'objet sur lequel a été appelé
14        p.maj_etat();//l'objet sur lequel a été appelé
15        System.out.println("après maj_etat()");
16        p.mon_etat();
17        p.maj_etat(8,8);//l'objet sur lequel a été appelé
18        System.out.println("après maj_etat(8,8)");
19        p.mon_etat();
20        p.maj_etat(9,9,"rouge");//l'objet sur lequel a été appelé
21        System.out.println("après maj_etat(9,9,'rouge)");
22        p.mon_etat();
23    }
24 }
```

Conclusion: Les éléments de la classe Point ne sont plus connus à partir d'une autre classe

Visibilité des membres: Tests et conclusions

- Dans la classe point , les attributs ont une visibilité privée et les **méthodes ont une visibilité public**
- Dans la classe Program, faire les modifications comme montrées dans la figure ci-dessous:

```
Program.java Point.java
1 package com.setp1;
2
3 public class Point {
4     private double abscisse;
5     private double ordonnee;
6     private String couleur;
7     public Point(){
8     }
9     public Point(double abscisse, double ordonnee){
10         this.abscisse=abscisse;
11         this.ordonnee=ordonnee;
12     }
13     public Point(double abscisse, double ordonnee,String couleur){
14         this.abscisse=abscisse;
15         this.ordonnee=ordonnee;
16         this.couleur=couleur;
17     }
18     public void mon_etat()
19     {
20         System.out.println ("mon état:");
21         System.out.println("(" +abscisse+ ", "+ordonnee+" )"+couleur );
22     }
23     public void maj_etat(){
24         abscisse=0;
25         ordonnee=0;
26         couleur="noir";
27     }
28     public void maj_etat(double abscisse,double ordonnee){
29         this.abscisse=abscisse;
30         this.ordonnee=ordonnee;
31         this.couleur="rouge"; //modification de l'état
```

```
Program.java Point.java
1 package com.setp1;
2
3 public class Program {
4     public static void main(String[] args) {
5         Point p;
6         p = new Point();
7         Point p2=new Point();
8         Point p3=new Point(3,3);
9         Point p4=new Point(4,4,"Bleu");
10        p.abscisse = 3;
11        p.ordonnee = 3;
12        p.couleur="verte";
13        System.out.println("avant changement de l'état");
14        p.mon_etat();//l'objet sur lequel a été appelé la méthode
15        p.maj_etat();//l'objet sur lequel a été appelé la méthode
16        System.out.println("après maj_etat()");
17        p.mon_etat();
18        p.maj_etat(8,8);//l'objet sur lequel a été appelé la méthode
19        System.out.println("après maj_etat(8,8)");
20        p.mon_etat();
21        p.maj_etat(9,9,"rouge");//l'objet sur lequel a été appelé la méthode
22        System.out.println("après maj_etat(9,9,'rouge')");
23        p.mon_etat();
24    }
25 }
```

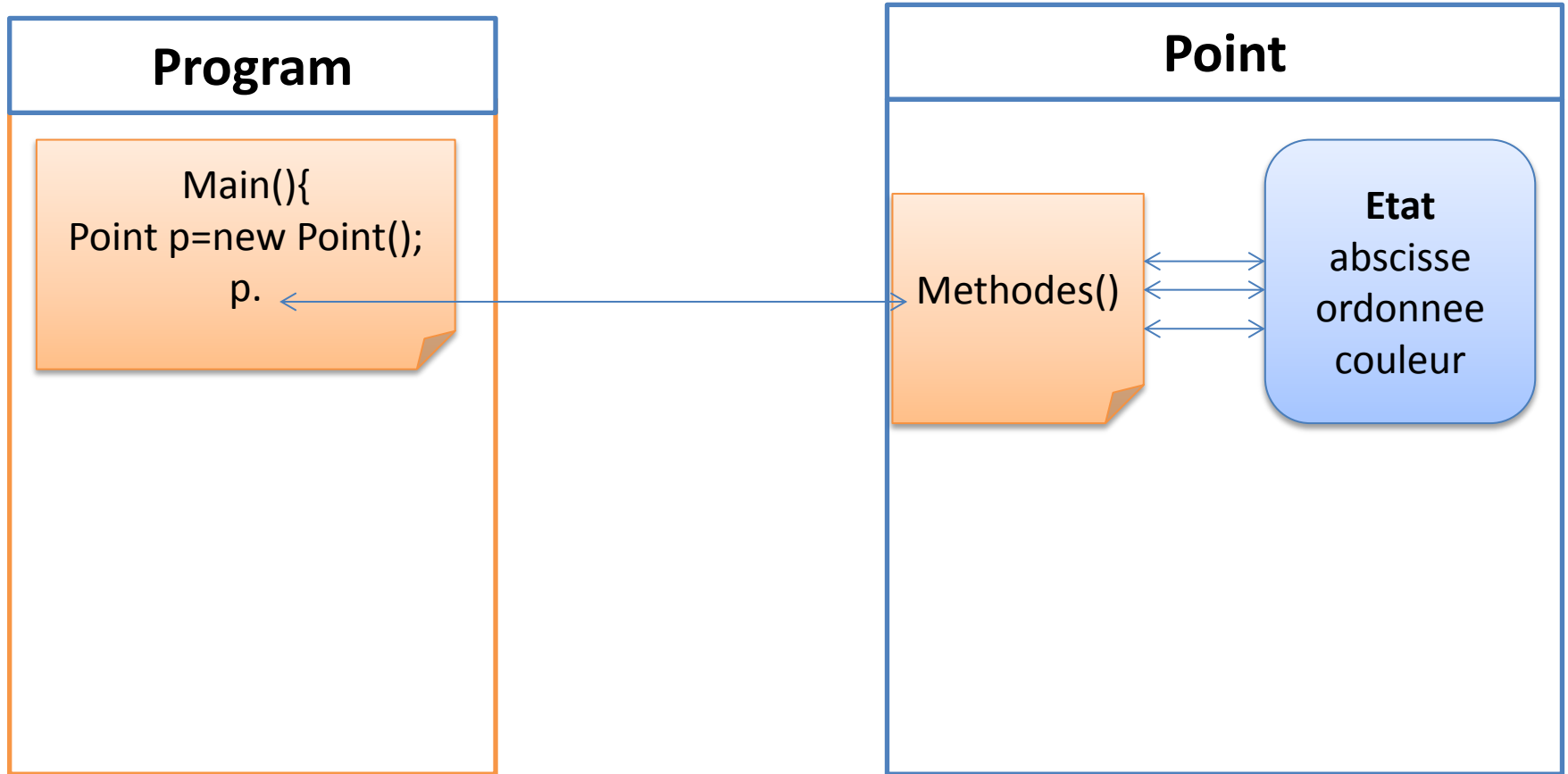
Conclusion: l'état de l'objet n'est plus accessible depuis l'extérieur **directement** ni en modification ni en consultation mais à travers des méthodes de la classe Point.

Encapsulation: rôle et utilisation

- En programmation orientée objet, l'encapsulation est une idée qui permet de protéger les attributs (l'état) de l'objet
- Pour assurer l'encapsulation, en général, les attributs ont une visibilité privée ou protégée
 - Pour les attributs privés, seulement les méthodes de la même classe qui peuvent accéder à ces attributs.
 - Pour les attributs protégés, seulement les méthodes de la même classe, les classes du **même namespace** ou **les classes filles** qui peuvent accéder à ses attributs.

NB: « les classes filles » CF Héritage

Encapsulation: rôle et utilisation



Encapsulation

Dans cette étape nous allons traiter les concepts suivants:

- Visibilité des membres: privé et protected
- Getters et Setters

NB: Durant cette partie, nous allons utiliser la visibilité *private* des attributs

Encapsulation

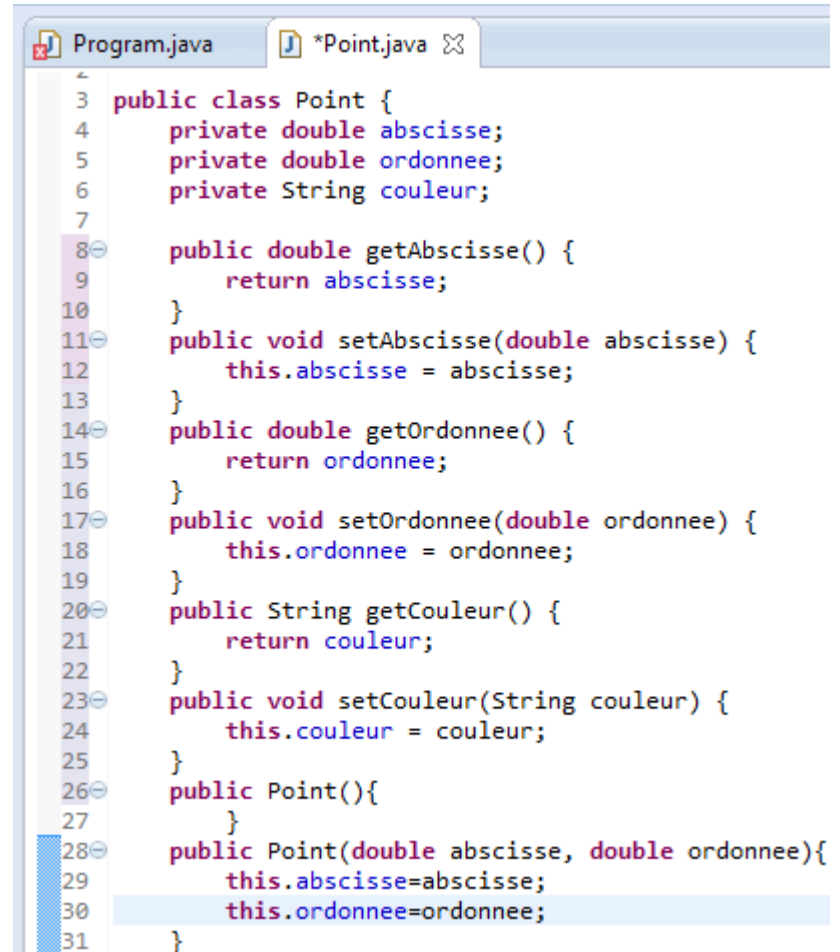
Les méthodes particulières de la classe: Getters et Setters

- Ce type de méthodes assure **l'accès aux différents attributs privés**
 - **Accès pour modification: les mutateurs ou setters**
 - Les setters commencent toujours par le mot **set** et finissent par le nom de l'attribut en écrivant en majuscule la lettre qui vient juste après le Set.
 - Les setters sont toujours de type void et reçoivent un paramètre ayant le même type que l'attribut.
 - **Accès pour consultations: les accesseurs ou getters**
 - Les getters commencent toujours par le mot **get** et finissent par le nom de l'attribut en écrivant en majuscule la lettre qui vient juste après le **get**.
 - Les getters retourne une valeur ayant toujours le même type que l'attribut correspondant.
- **En cas de besoin, un attribut est relié à son getter et à son setter**

Getters et Setters

Les getters et les setters de la classe Point

```
public double getAbscisse() {  
    return abscisse;  
}  
public void setAbscisse(double abscisse) {  
    this.abscisse = abscisse;  
}  
public double getOrdonnee() {  
    return ordonnee;  
}  
public void setOrdonnee(double ordonnee) {  
    this.ordonnee = ordonnee;  
}  
public String getCouleur() {  
    return couleur;  
}  
public void setCouleur(String couleur) {  
    this.couleur = couleur;  
}
```

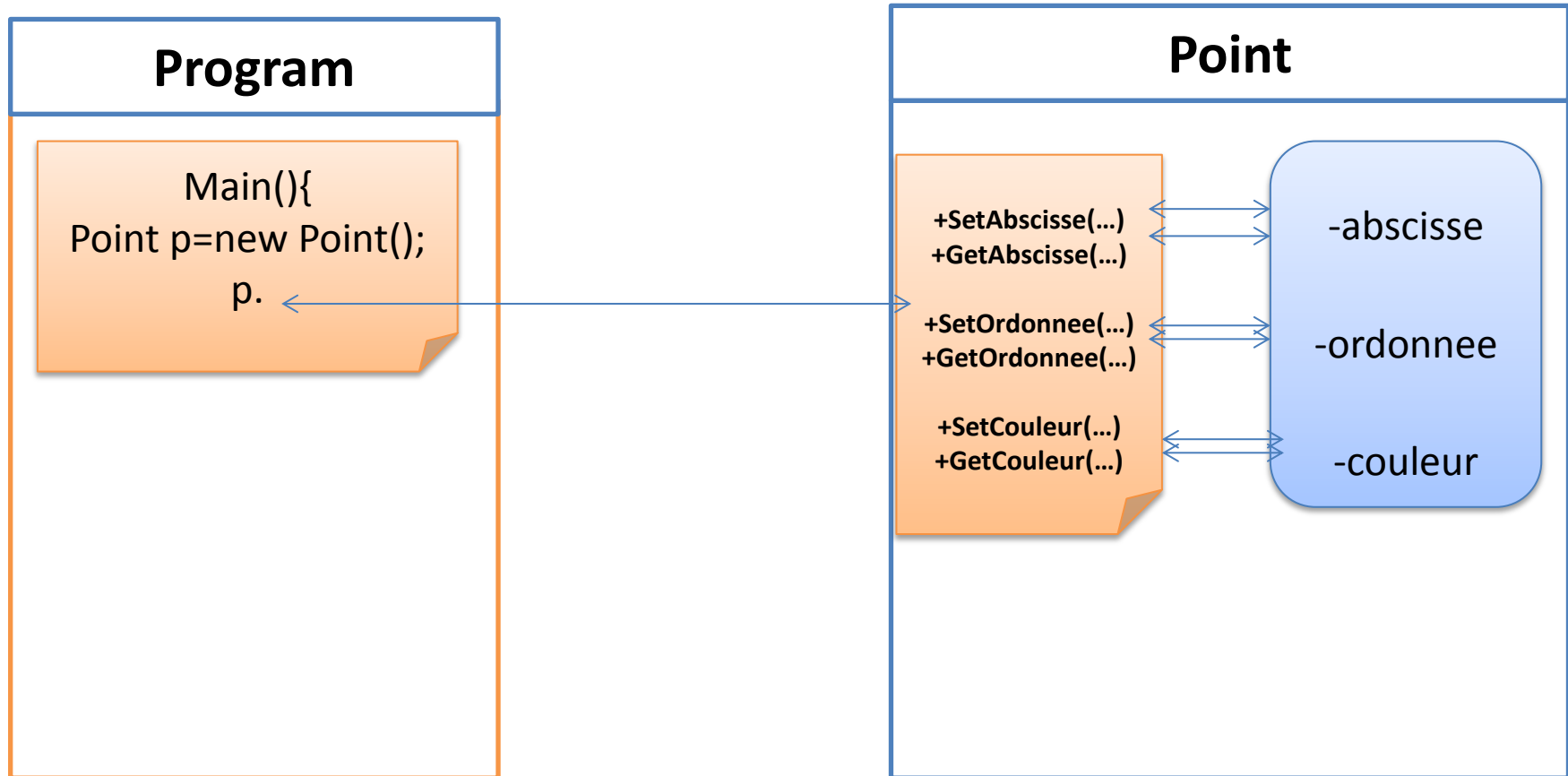


```
Program.java  *Point.java  
3  public class Point {  
4      private double abscisse;  
5      private double ordonnee;  
6      private String couleur;  
7  
8      public double getAbscisse() {  
9          return abscisse;  
10     }  
11     public void setAbscisse(double abscisse) {  
12         this.abscisse = abscisse;  
13     }  
14     public double getOrdonnee() {  
15         return ordonnee;  
16     }  
17     public void setOrdonnee(double ordonnee) {  
18         this.ordonnee = ordonnee;  
19     }  
20     public String getCouleur() {  
21         return couleur;  
22     }  
23     public void setCouleur(String couleur) {  
24         this.couleur = couleur;  
25     }  
26     public Point(){  
27     }  
28     public Point(double abscisse, double ordonnee){  
29         this.abscisse=abscisse;  
30         this.ordonnee=ordonnee;  
31     }  
}
```

Les getters et les setters peuvent être générés facilement à l'aide de l'IDE Eclipse

Encapsulation

Getters et Setters



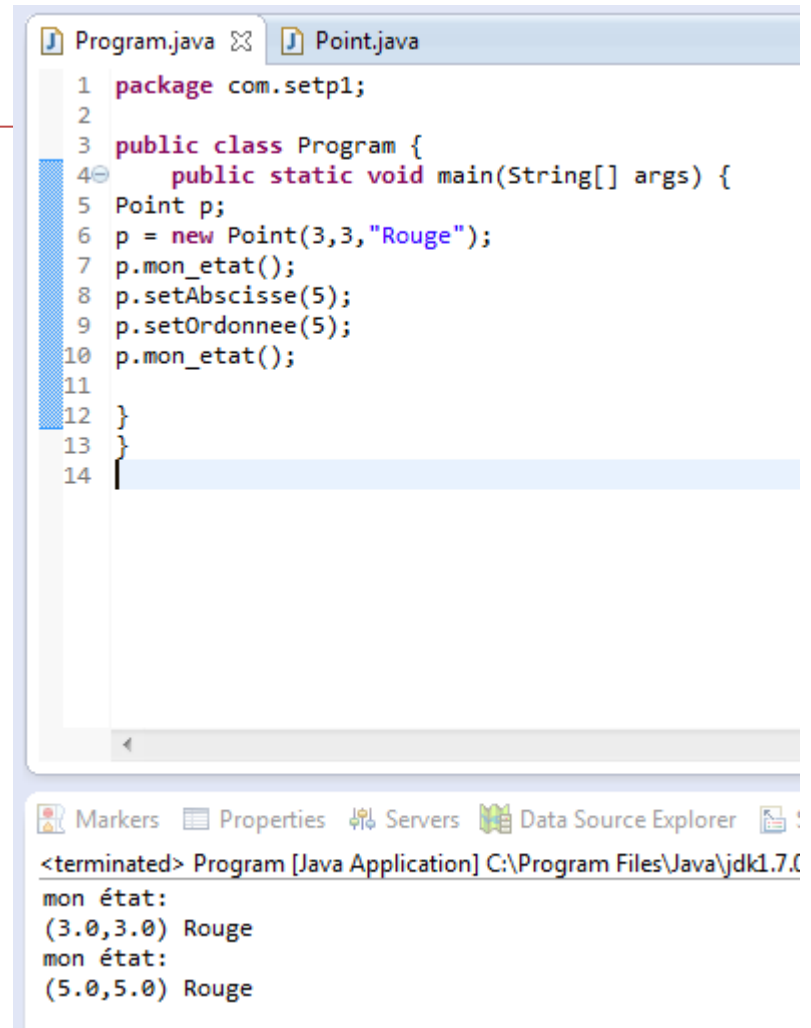
Généralement, l'état d'un objet est privé ou protégé, et son comportement est public

Encapsulation

Utilisation des getters et setters

```
package com.setp1;
```

```
public class Program {  
    public static void main(String[] args) {  
        Point p;  
        p = new Point(3,3,"Rouge");  
        p.mon_etat();  
        p.setAbscisse(5);  
        p.setOrdonnee(5);  
        p.mon_etat();  
    }  
}
```



```
Program.java Point.java  
1 package com.setp1;  
2  
3 public class Program {  
4     public static void main(String[] args) {  
5         Point p;  
6         p = new Point(3,3,"Rouge");  
7         p.mon_etat();  
8         p.setAbscisse(5);  
9         p.setOrdonnee(5);  
10        p.mon_etat();  
11    }  
12 }  
13 }  
14 |  
  
Markers Properties Servers Data Source Explorer  
<terminated> Program [Java Application] C:\Program Files\Java\jdk1.7.0_75\bin\java.exe  
mon état:  
(3.0,3.0) Rouge  
mon état:  
(5.0,5.0) Rouge
```


Concept N° 3

Héritage

Héritage

Définition

Dans la programmation orientée objet (que ce soit en C++, Java, C#, ...), le concept d'héritage est une technique très puissante et extrêmement pratique.

- Dans l'héritage on parle de:
 - une classe de base (classe mère)
 - Une classe dérivée (classe fille)
- L'héritage permet de représenter la relation: est un

Héritage

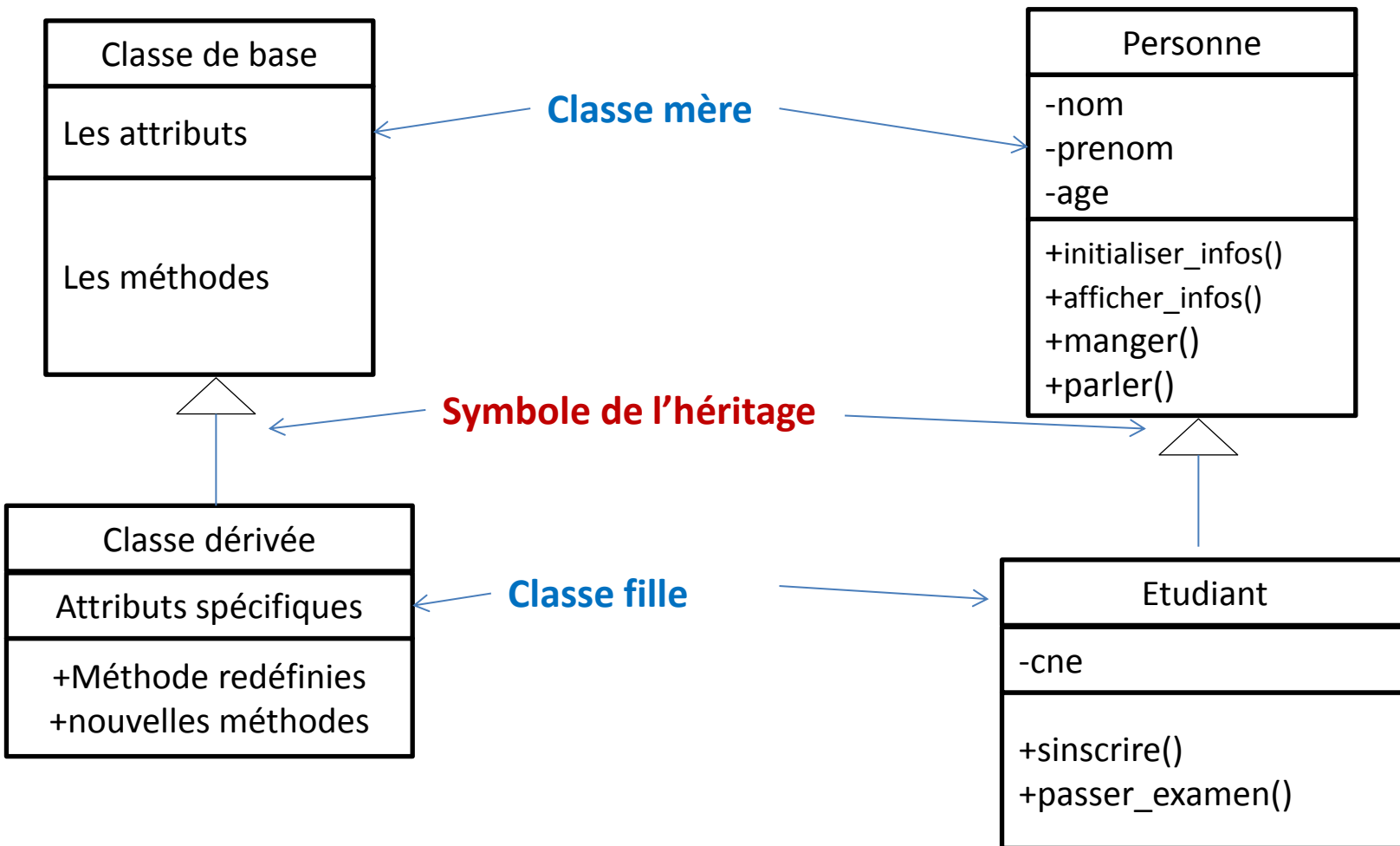
Exemple d'introduction

L'exemple que nous allons traiter contient les éléments suivants:

- une classe « **Personne** » qui va servir comme une classe de base (classe mère). Le code source de cette classe se trouve dans `Personne.java`
- Une classe « **Etudiant** » qui va servir comme une classe dérivée de la classe `Personne` et dont le code source est dans le fichier `Etudiant.java`
- Une classe « **Program** » qui va servir comme programme principal et dont le code source est dans le fichier `Program.java`

Héritage

Représentation UML de l'héritage



Héritage: une relation de type « est un »
Un étudiant est une personne

Héritage

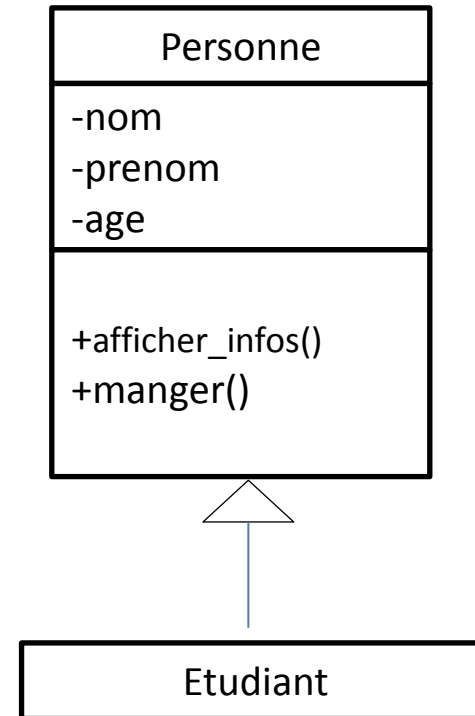
Relation entre la classe mère et la classe dérivée

- Le concept de l'héritage permet de créer une nouvelle classe (classe dérivée) à partir d'une classe existante (classe mère).
 - La classe dérivée **contient tous les éléments de la classe mère** (attributs et méthodes) **sauf ses constructeurs**.
 - La classe dérivée **peut posséder autres éléments** (attributs et/ou méthodes) spécifiques
 - La classe dérivée **peut redéfinir des méthodes** de la classe mère.
- On dit aussi que la classe dérivée étend la classe de base.

Héritage

Implémentation du diagramme de classe

```
package com.scolarite;
public class Personne {
    private String nom;
    private String prenom;
    private int age;
    public void afficher_infos(){
        System.out.println("nom:"+nom);
        System.out.println("prénom:"+prenom);
        System.out.println("age:"+age);
    }
    public void manger(){
        System.out.println("je mange, ham ham");
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
}
```



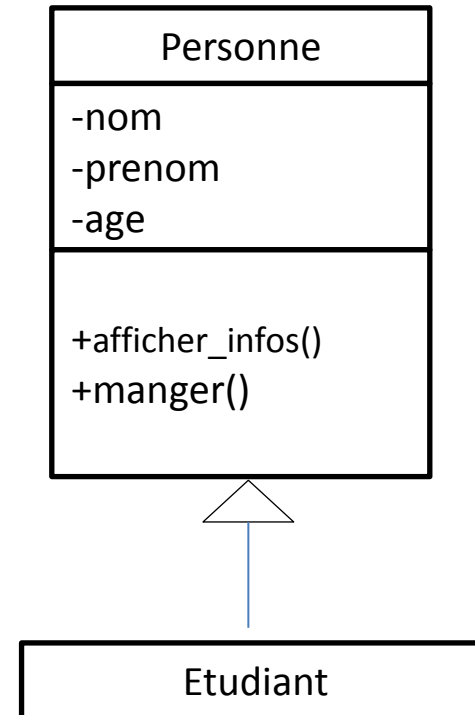
The screenshot shows a code editor with two tabs: 'Personne.java' and 'Etudiant.java'. The 'Etudiant.java' tab is active, displaying the following code:

```
1 package com.scolarite;
2
3 public class Etudiant extends Personne{
4
5 }
```

Héritage

Implémentation du diagramme de classe

```
Personne.java  Etudiant.java
1 package com.scolarite;
2 public class Personne {
3     private String nom;
4     private String prenom;
5     private int age;
6     public void afficher_infos(){
7         System.out.println("nom:"+nom);
8         System.out.println("prénom:"+prenom);
9         System.out.println("age:"+age);
10    }
11    public void manger(){
12        System.out.println("je mange, ham ham");
13    }
14    public String getNom() {
15        return nom;
16    }
17    public void setNom(String nom) {
18        this.nom = nom;
19    }
20    public String getPrenom() {
21        return prenom;
22    }
23    public void setPrenom(String prenom) {
24        this.prenom = prenom;
25    }
}
```



Pour appliquer l'héritage en java, on utilise « **extends** »

```
Personne.java  Etudiant.java
1 package com.scolarite;
2
3 public class Etudiant extends Personne{
4
5 }
```

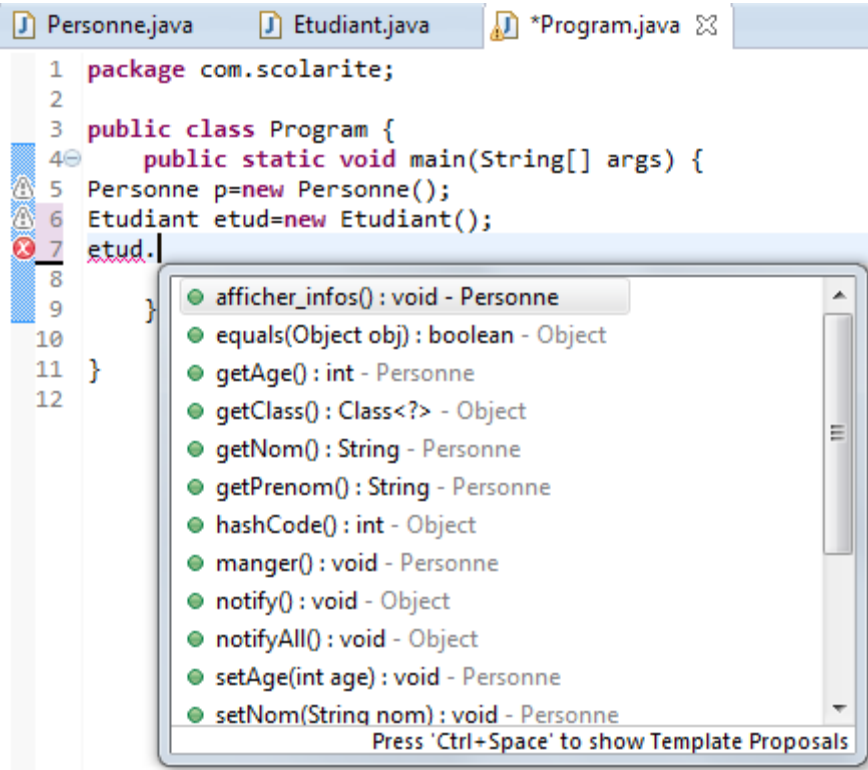
Héritage

Dans cette étape nous allons traiter les concepts suivants:

- **Création des objets de la classe dérivée**
- Visibilité des attributs
- Le mot clé super
- Personnaliser la classe dérivée
- Traitement des membres spécifiques à la classe dérivée
- Compatibilité de type entre la classe de base et la classe dérivée
- Exemples de classes de base prédéfinies

Héritage

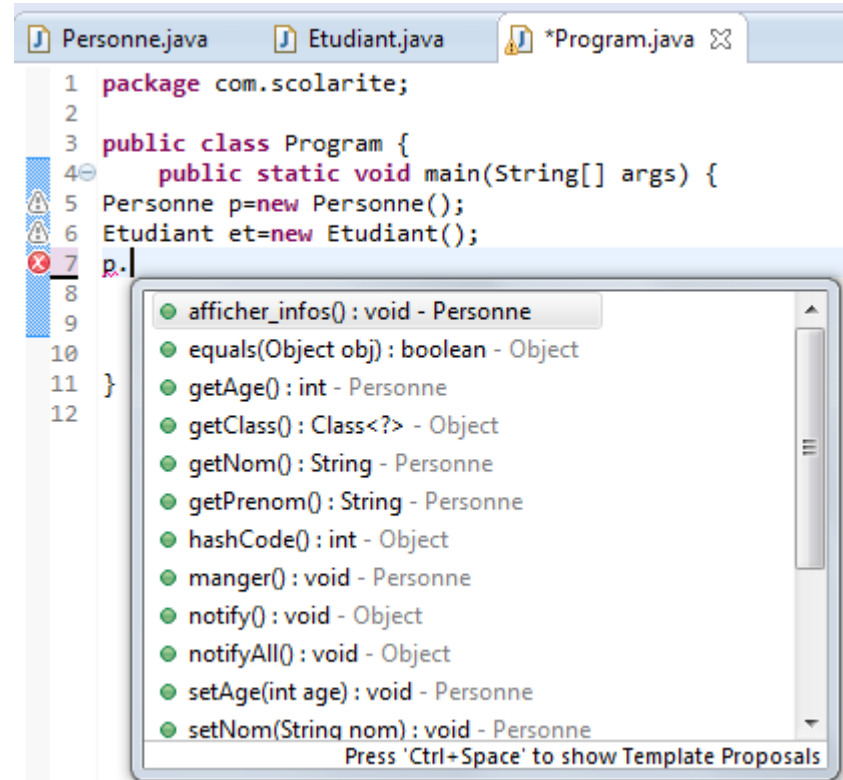
Création des objets de la classe de base et la classe dérivée



```
1 package com.scolarite;
2
3 public class Program {
4     public static void main(String[] args) {
5         Personne p=new Personne();
6         Etudiant etud=new Etudiant();
7         etud.
8     }
9 }
10
11 }
12
```

- afficher_infos() : void - Personne
- equals(Object obj) : boolean - Object
- getAge() : int - Personne
- getClass() : Class<?> - Object
- getNom() : String - Personne
- getPrenom() : String - Personne
- hashCode() : int - Object
- manger() : void - Personne
- notify() : void - Object
- notifyAll() : void - Object
- setAge(int age) : void - Personne
- setNom(String nom) : void - Personne

Press 'Ctrl+Space' to show Template Proposals



```
1 package com.scolarite;
2
3 public class Program {
4     public static void main(String[] args) {
5         Personne p=new Personne();
6         Etudiant et=new Etudiant();
7         p.
8     }
9 }
10
11 }
12
```

- afficher_infos() : void - Personne
- equals(Object obj) : boolean - Object
- getAge() : int - Personne
- getClass() : Class<?> - Object
- getNom() : String - Personne
- getPrenom() : String - Personne
- hashCode() : int - Object
- manger() : void - Personne
- notify() : void - Object
- notifyAll() : void - Object
- setAge(int age) : void - Personne
- setNom(String nom) : void - Personne

Press 'Ctrl+Space' to show Template Proposals

Même si la classe étudiant est vide, l'objet etud et l'objet p contiennent les mêmes méthodes

Héritage

Dans cette étape nous allons traiter les concepts suivants:

- **Création des objets de la classe dérivée**
- **Visibilité des attributs**
- Le mot clé base
- Personnaliser la classe dérivée
- Traitement des membres spécifiques à la classe dérivée
- Compatibilité de type entre la classe de base et la classe dérivée
- Exemples de classe de base prédéfinies

Héritage

Visibilité des attributs de la classe mère dans la classe fille

- Les attributs privés de la classe mère ne sont pas directement accessibles dans les classes filles
- Les attributs protégés de la classe mère sont directement accessibles dans les classes filles

Exercice:

Écrire un exemple pour vérifier les propriétés en dessus

- Selon les besoins, le concepteur fixe la visibilité des attributs de la classe mère à `protected` ou `private`
- Généralement, les attributs de la classe mère sont `protected`

Héritage

Les attributs protégés de la classe mère

```
Personne.java  Etudiant.java  *Program.java
1 package com.scolarite;
2 public class Personne {
3     protected String nom;
4     protected String prenom;
5     protected int age;
6     public void afficher_infos(){
7         System.out.println("nom:"+nom);
8         System.out.println("prénom:"+prenom);
9         System.out.println("age:"+age);
10    }
11    public void manger(){
12        System.out.println("je mange, ham ham");
13    }
14    public String getNom() {
15        return nom;
16    }
17    public void setNom(String nom) {
18        this.nom = nom;
19    }
20    public String getPrenom() {
21        return prenom;
22    }
23    public void setPrenom(String prenom) {
24        this.prenom = prenom;
25    }
}
```

Personne
#nom #prenom #age
+afficher_infos() +manger()

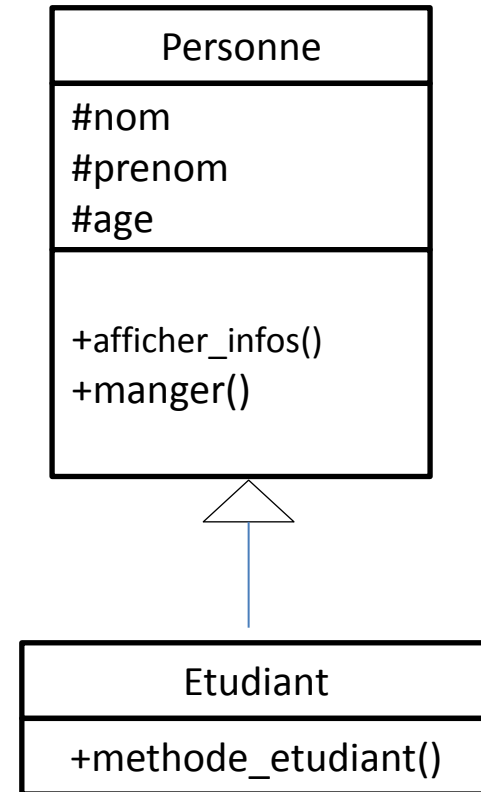
```
Personne.java  Etudiant.java  *Program.java
1 package com.scolarite;
2
3 public class Etudiant extends Personne{
4     public Etudiant(String nom,String prenom,int age){
5         this.nom=nom;
6         this.prenom=prenom;
7         this.age=age;
8     }
9 }
```

- Les attributs protégés de la classe mère sont directement accessibles dans les classes filles

Héritage

Les attributs protégés de la classe mère

Compléter le code de la classe étudiant en implémentant la méthode « `methode_etudiant()` » pour vérifier que les attributs **protégés** de la classe **personne** **sont directement** accessibles aux méthodes de la classe fille.

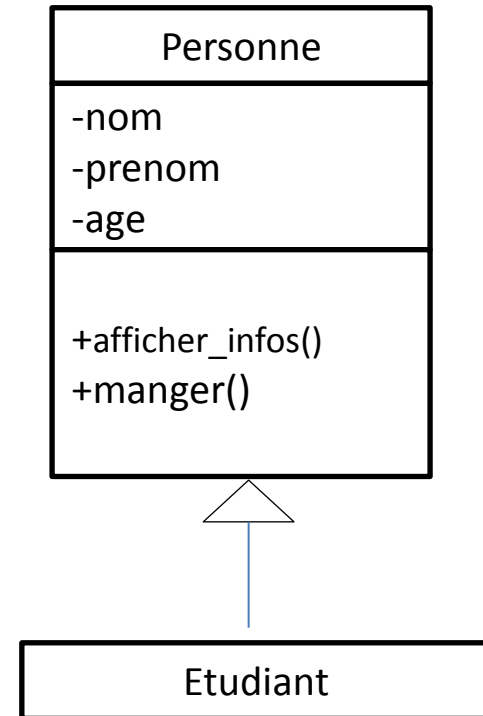


Héritage

Les attributs privés de la classe mère

Modifier la visibilité des attributs de classe personne en private. Quels sont les instructions qui ne deviennent plus validés.

Corriger le code de la classe Etudiant



Héritage

Dans cette étape nous allons traiter les concepts suivants:

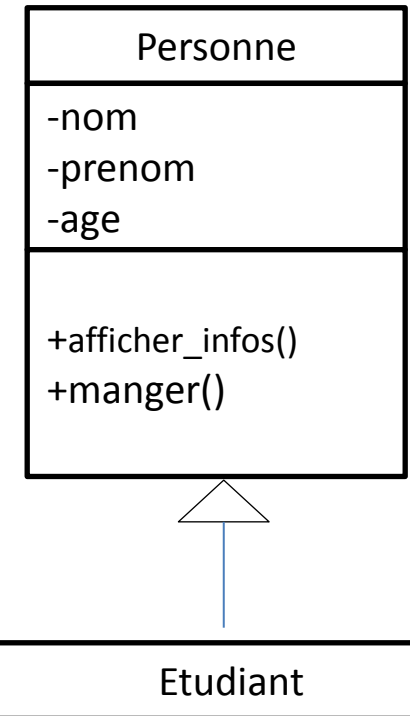
- **Création des objets de la classe dérivée**
- **Visibilité des attributs**
- **Le mot clé super**
- Personnaliser la classe dérivée
- Traitement des membres spécifiques à la classe dérivée
- Compatibilité de type entre la classe de base et la classe dérivée
- Exemples de classe de base prédéfinies

Le mot clé **super**

```
Personne.java  Etudiant.java  Program.java
1 package com.scolarite;
2 public class Personne {
3     protected String nom;
4     protected String prenom;
5     protected int age;
6
7     public Personne(String nom,String prenom,int age){
8         this.nom=nom;
9         this.prenom=prenom;
10        this.age=age;
11    }
12 }
```

```
Personne.java  Etudiant.java  Program.java
1 package com.scolarite;
2
3 public class Etudiant extends Personne{
4     public Etudiant(String nom,String prenom,int age){
5         super(nom,prenom,age);
6     }
7 }
8 }
```

```
Personne.java  Etudiant.java  Program.java
1 package com.scolarite;
2
3 public class Program {
4     public static void main(String[] args) {
5         Personne p=new Personne("Ali", "Baba",99);
6         Etudiant etud=new Etudiant("Tati", "Tata",19);
7     }
8 }
9
10 }
```



Dans le constructeur de la classe fille, le mot clé **super** lui permet d'utiliser le constructeur de classe mère.

Héritage

Dans cette étape nous allons traiter les concepts suivants:

- **Création des objets de la classe dérivée**
- **Visibilité des attributs**
- **Le mot clé super**
- **Personnaliser la classe dérivée**
- **Traitement des membres spécifiques à la classe dérivée**
- **Compatibilité de type entre la classe de base et la classe dérivée**
- **Exemples de classe de base prédéfinies**

Héritage

Ajouter des attributs et des méthodes spécifiques à la classe fille

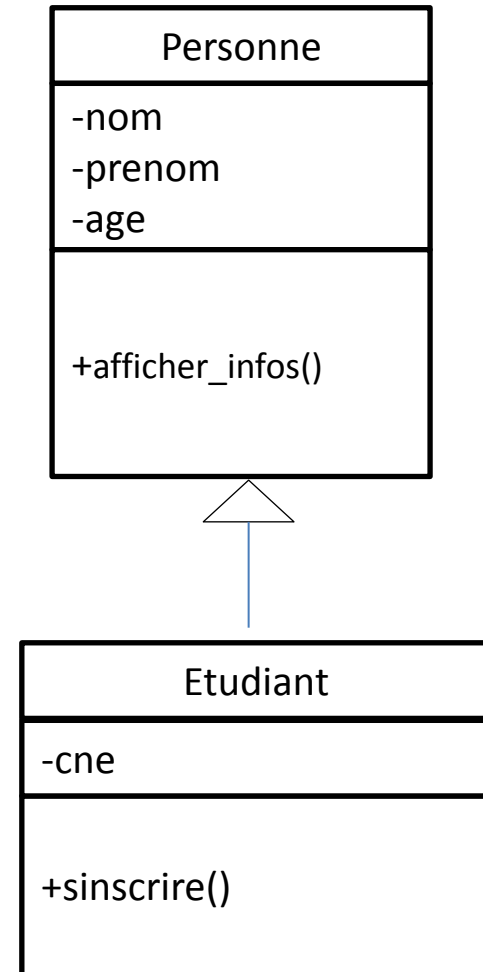
```
package com.scolarite;

public class Etudiant extends Personne{
    private String cne;

    public Etudiant(String nom,String prenom,int age){
        super(nom,prenom,age);
    }

    public Etudiant(String nom,String prenom,int age,String cne){
        super(nom,prenom,age);
        this.cne=cne;
    }

    public void inscrire(){
        System.out.println("je souhaite m'inscrire en ...");
    }
}
```



Héritage

Dans cette étape nous allons traiter les concepts suivants:

- **Création des objets de la classe dérivée**
- **Visibilité des attributs**
- **Le mot clé base**
- **Personnaliser la classe dérivée**
- **Traitement des membres spécifiques à la classe dérivée**
- **Compatibilité de type entre la classe de base et la classe dérivée**
- **Exemples de classe de base prédéfinies**

Redéfinition des méthodes

- Quand une classe hérite d'une autre classe, elle peut redéfinir les méthodes héritées.
- Par exemple, dans la classe Etudiant on doit personnaliser les méthode **afficher_infos()** pour tenir en compte le cne

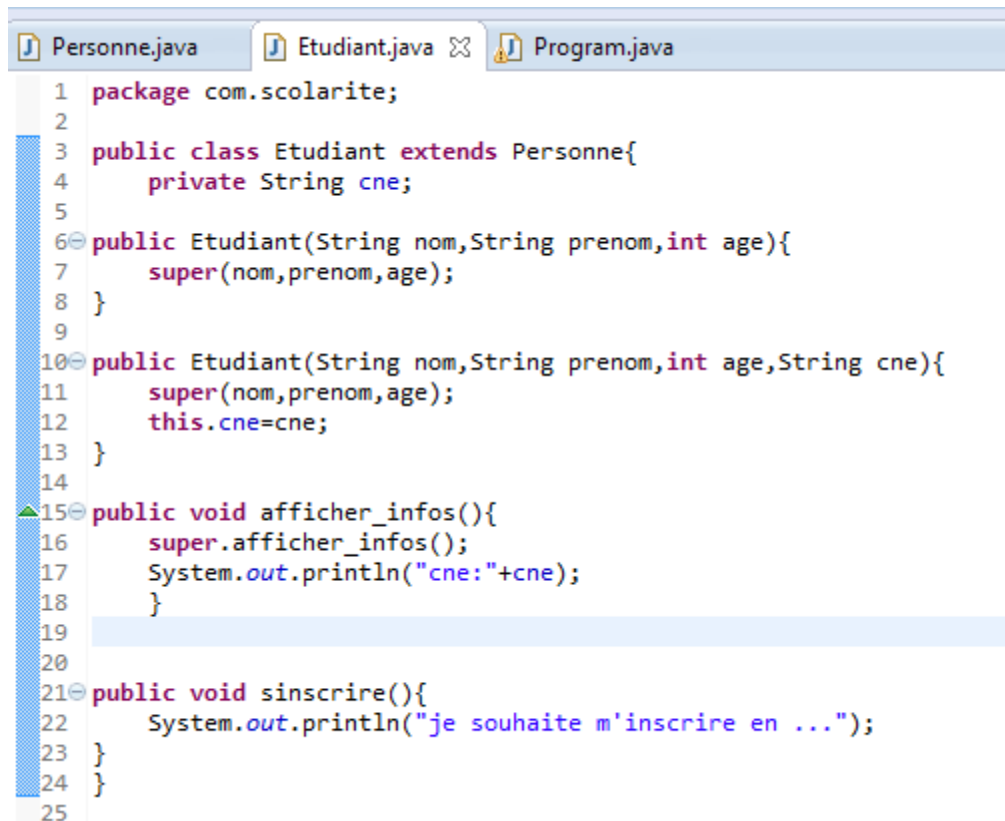
```
Personne.java  Etudiant.java  Program.java
1 package com.scolarite;
2 public class Personne {
3     protected String nom;
4     protected String prenom;
5     protected int age;
6
7     public Personne(String nom,String prenom,int age){
8         this.nom=nom;
9         this.prenom=prenom;
10        this.age=age;
11    }
12
13    public void afficher_infos(){
14        System.out.println("nom:"+nom);
15        System.out.println("prénom:"+prenom);
16        System.out.println("age:"+age);
17    }
18
```

```
Personne.java  Etudiant.java  Program.java
1 package com.scolarite;
2
3 public class Etudiant extends Personne{
4     private String cne;
5
6     public Etudiant(String nom,String prenom,int age){
7         super(nom,prenom,age);
8     }
9
10    public Etudiant(String nom,String prenom,int age,String cne){
11        super(nom,prenom,age);
12        this.cne=cne;
13    }
14
15    public void afficher_infos(){
16        System.out.println("nom:"+nom);
17        System.out.println("prénom:"+prenom);
18        System.out.println("age:"+age);
19        System.out.println("cne:"+cne);
20    }
21
```

Modifier la visibilité des attributs de la classe personne en private: il seront plus visibles dans la classe fille

Comment procéder pour résoudre ce problème ?

Redéfinition des méthodes



```
1 package com.scolarite;
2
3 public class Etudiant extends Personne{
4     private String cne;
5
6     public Etudiant(String nom,String prenom,int age){
7         super(nom,prenom,age);
8     }
9
10    public Etudiant(String nom,String prenom,int age,String cne){
11        super(nom,prenom,age);
12        this.cne=cne;
13    }
14
15    public void afficher_infos(){
16        super.afficher_infos();
17        System.out.println("cne:"+cne);
18    }
19
20
21    public void sinscrire(){
22        System.out.println("je souhaite m'inscrire en ...");
23    }
24 }
25
```

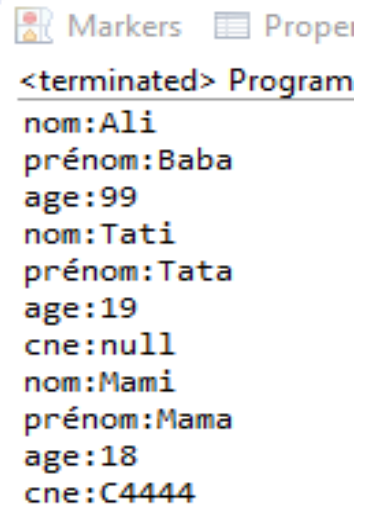
La méthode **afficher_infos()** est redéfinie pour prendre en compte les spécificités d'affichage d'un **objet étudiant**

Pour appeler la méthode redéfinie, on utilise le mot clé **super** suivi de « . »

Traitement des attributs spécifiques à la classe fille

Exemple d'utilisation

```
package com.scolarite;  
  
public class Program {  
    public static void main(String[] args) {  
        Personne p=new Personne("Ali", "Baba",99);  
        Etudiant etud1=new Etudiant("Tati", "Tata",19);  
        Etudiant etud2=new Etudiant("Mami", "Mama",18,"C4444");  
        p.afficher_infos();  
        etud1.afficher_infos();  
        etud2.afficher_infos();  
    }  
}
```



Markers Proper

<terminated> Program

nom:Ali
prénom:Baba
age:99
nom:Tati
prénom:Tata
age:19
cne:null
nom:Mami
prénom:Mama
age:18
cne:C4444

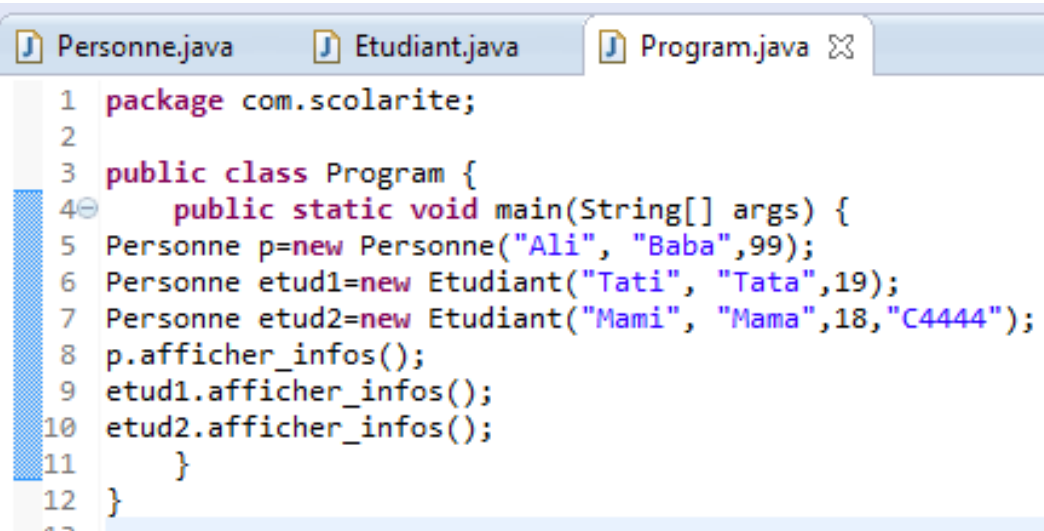
Héritage

Dans cette étape nous allons traiter les concepts suivants:

- Création des objets de la classe dérivée
- Visibilité des attributs
- Le mot clé base
- Personnaliser la classe dérivée
- Traitement des membres spécifiques à la classe dérivée
- Compatibilité de type entre la classe de base et la classe dérivée
- Exemples de classe de base prédéfinies

Compatibilité de type entre la classe fille et la classe mère

Exemple d'utilisation N°1



```
1 package com.scolarite;
2
3 public class Program {
4     public static void main(String[] args) {
5         Personne p=new Personne("Ali", "Baba",99);
6         Personne etud1=new Etudiant("Tati", "Tata",19);
7         Personne etud2=new Etudiant("Mami", "Mama",18,"C4444");
8         p.afficher_infos();
9         etud1.afficher_infos();
10        etud2.afficher_infos();
11    }
12 }
```



```
<terminated> Program (1) [Java App
nom:Ali
prénom:Baba
age:99
nom:Tati
prénom:Tata
age:19
cne:null
nom:Mami
prénom:Mama
age:18
cne:C4444
```

- Un objet de la classe Etudiant peut se considérer comme un objet de la classe Personne

Héritage

Dans cette étape nous allons traiter les concepts suivants:

- **Création des objets de la classe dérivée**
- **Visibilité des attributs**
- **Le mot clé base**
- **Personnaliser la classe dérivée**
- **Traitement des membres spécifiques à la classe dérivée**
- **Compatibilité de type entre la classe de base et la classe dérivée**
- **Exemples de classe de base prédéfinies**

Classe de base prédéfinies

Classe **Object** de Java

La classe **Object** est une classe prédéfinie dans Java
Par défaut, toutes les classes héritent de la classe **Object**

Les principales méthodes de la classe **Object**:

- La méthode `equals()`
- La méthode `hashCode()`
- La méthode `getType()`
- La méthode `toString()`

Exercice

Redéfinir la méthode `equals` pour comparer deux étudiants

Redéfinir la méthode `toString` de classe `Personne` puis de la classe `Etudiant` pour récupérer les informations de leurs objets

Héritage

Avantages

- L'héritage offre plusieurs avantages:
 - Réutilisation du code: ne pas démarrer du zéro
 - Maintenabilité: maîtrise davantage du code
 - Évolutivité: la réutilisation et la maintenabilité assure les éléments de base pour une meilleure évolution

Héritage

Dans cette étape nous avons traiter les concepts suivants:

- Création des objets de la classe dérivée
- Visibilité des attributs
- Le mot clé base
- Personnaliser la classe dérivée
- Traitement des membres spécifiques à la classe dérivée
- Compatibilité de type entre la classe de base et la classe dérivée
- Exemples de classe de base prédéfinies

Héritage

Application

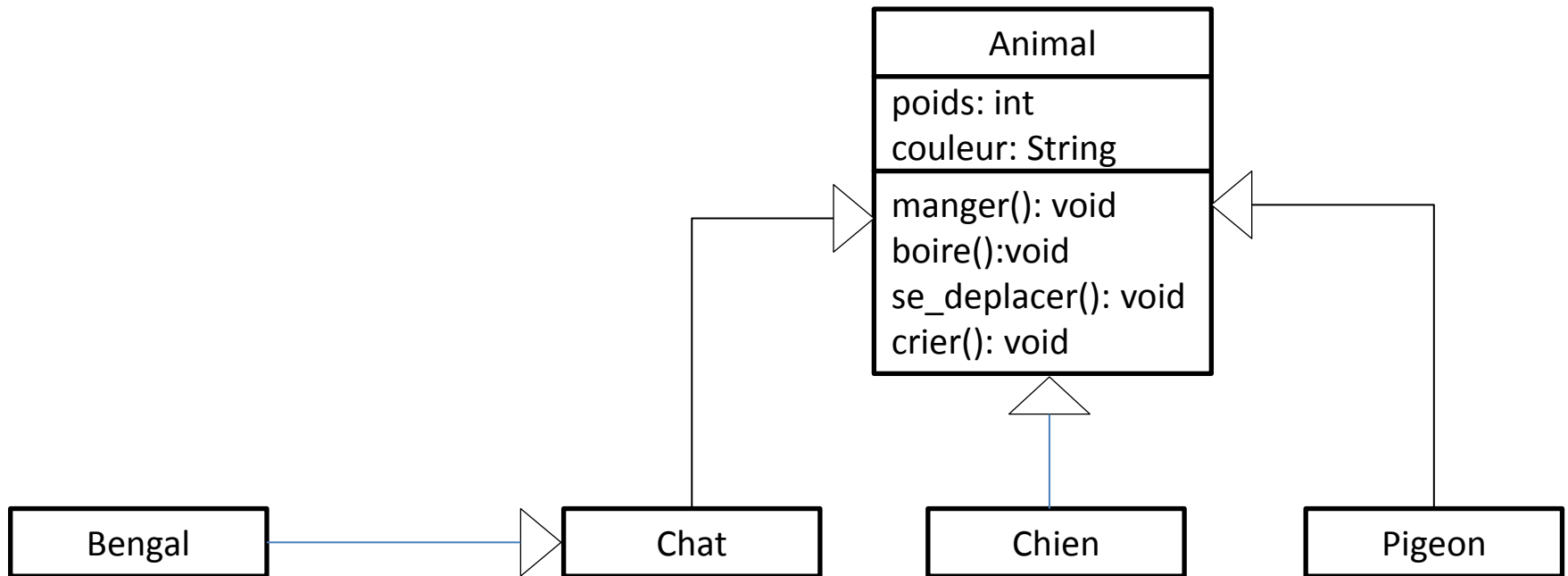


Diagramme de classe qui montre l'héritage

Implémenter ce diagramme de classe

Concept N° 4

Polymorphisme

Le polymorphisme offre aux objets la possibilité d'appartenir à plusieurs catégories à la fois.

Ce concept nous permet de faire des opérations sur des objets indépendamment de leur catégories.

Exemple 1

3 pommes + 5 oranges = 8 fruits

Exemple 2

3 Ingénieurs + 5 Techniciens = 8 fruits

Le sur-casting des objets:

Une façon de décrire l'exemple consistant à additionner des pommes et des oranges serait d'imaginer que nous disons pommes et oranges mais que nous manipulons en réalité des fruits.

Nous pourrions écrire alors la formule correcte :

3 (fruits) pommes
+ 5 (fruits) oranges

----- --
= 8 fruits

Cette façon de voir les choses implique que les pommes et les oranges soient "transformés" en fruits préalablement à l'établissement du problème. Cette transformation est appelée *sur-casting*

Fin du chapitre 3