

# Formation 'Développeur Java Web Mobile'

## Module 1 – Séance 11



### Module 1 36h

#### CONCEPTS OBJETS ÉLÉMENTAIRES

- Séance 1 Le métier de Développeur
- Séance 2 5 atouts Java + 5 repères clés développeur Java
- Séance 3 Installation et analyse du JDK
- Séance 4 Premier programme java + Syntaxe (instruction, méthode)
- Séance 5 Syntaxe Java (création de methodes)
- Séance 6 Syntaxe Java (boucles)
- Séance 7 Syntaxe Java (types primitifs)
- Séance 8 Prise en main outil Eclipse + 15 bonnes pratiques
- Séance 9 Génération documentation : outil javadoc.exe
- Séance 10 Création de classes et d'Objets
- Séance 11 Encapsulation (propriétés private, protected)
- Séance 12 Constructeur (définition + codage)

# Sommaire Module 1 – Séance 11

- **Encapsulation : une notion clé de la POO** 3
- **L'encapsulation pour le concepteur** 5
- **Notions de Getter et Setter** 6
- **Exemple : Employe pour le concepteur** 7
- **Exemple : Employe pour le développeur** 8
- **Questions** 11

# Encapsulation

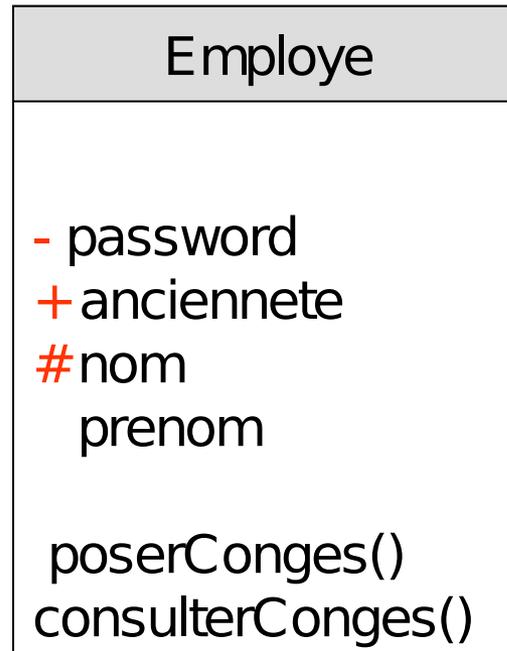
- L'encapsulation consiste à réduire la visibilité des propriétés et méthodes d'une classe, afin de n'exposer que les fonctionnalités réellement nécessaires pour les classes utilisatrices.
- Cette réduction de visibilité est rendue possible avec
  - **private** : visibilité interne à la classe
  - **public** : visibilité pour toutes les classes
  - **protected** : visibilité pour les classes enfants
  - **default** (aucun des modes ci-dessus n'est spécifié) : visibilité pour les classes du même package

# Tableau des visibilité

Classe Membre	la classe	une classe dans le même package	une classe qui hérite dans le package	une classe qui hérite hors package	une autre classe hors package
public	Oui	Oui	Oui	Oui	Oui
protected	Oui	Oui	Oui	Oui	Non
∅	Oui	Oui	Oui	Non	Non
private	Oui	Non	Non	Non	Non

# Notation UML de l'encapsulation

private  
public  
protected  
default



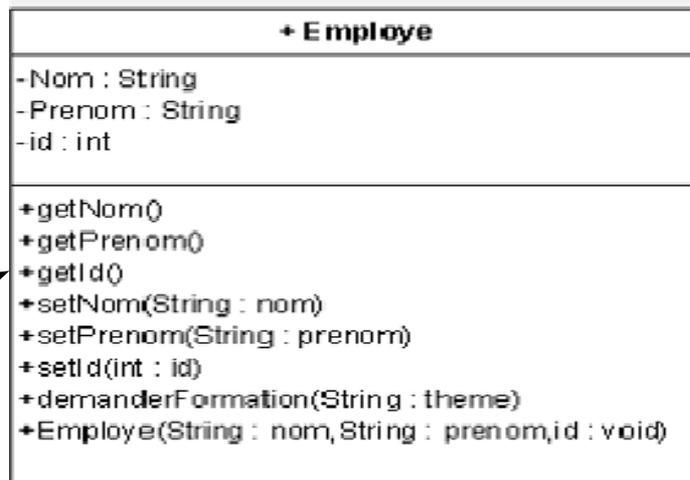
# Les 'getters' et 'setters'

- Les attributs privés ne sont pas accessibles : il leur faut des accesseurs dédiés
  - Lecture : « getter »
  - Ecriture : « setter »
- Les noms de ces opérations suivent la convention suivante :
  - Ils sont constitués de get ou set suivant le type d'accesseur, suivi du nom de l'attribut débutant par une majuscule :
    - Exemple d'attribut:
      - *private String password;*
    - Exemple d'accesseurs sur l'attribut password :
      - *public String getPassword()*
      - *public void setPassword(String p)*

# Encapsulation

- Bonne pratique : masquer les propriétés (**private**)
- On ne peut accéder 'directement' aux propriétés
  - Passer par des méthodes publiques que la classe expose dans sa grande bonté
  - Méthodes publiques **getters / setters**
  - Exemple 1 : `employe1.id` → echec !
  - Exemple 2 : `employe1.getId()` retourne l'id

(-) De l'extérieur de la classe, on ne peut accéder directement à la propriété id



(+) De l'extérieur de la classe, on peut lire et modifier la propriété id en utilisant les méthodes `getId()` et `setId()` fournies par la classe

# Coder une classe : exemple 3

```

Employee3.java x
1 public class Employe3 {
2
3     private String nom ;           // Propriété 'nom'
4     private String prenom ;
5
6     public String getNom() { // Getter pour propriété 'nom'
7         return nom;
8     }
9
10    public void setNom(String nom) { // Setter pour propriété 'nom'
11        this.nom = nom;
12    }
13
14    public String getPrenom() { // Getter pour propriété 'prenom'
15        return prenom;
16    }
17
18    public void setPrenom(String prenom) { // Setter pour propriété 'prenom'
19        this.prenom = prenom;
20    }
21
22    public void poserConges () {
23        System.out.println("L'employe " + this.nom + " " + this.prenom + " pose des Conges");
24    }
25 }
    
```

1

Propriétés

2

Getters  
Setters

3

Méthodes  
métier

# Le code suivant compile t'il ? Pourquoi ?

MonApplication2.java

```
1 public class MonApplication2 {
2
3     public static void main (String arg[] ) {
4         System.out.println ("Bienvenue dans mon application de gestion ressources humaines");
5
6         Employe employe1 = new Employe() ;
7
8         System.out.println ("Employe 1 : " + employe1.nom ) ;
9     }
10 }
```



```
C:\ObjisCoteDivoire\seance11\code>javac MonApplication2.java
MonApplication2.java:8: error: nom has private access in Employe
    System.out.println ("Employe 1 : " + employe1.nom ) ;
                                               ^
1 error
C:\ObjisCoteDivoire\seance11\code>
```

# Modification de l'état en utilisant un Setter()

MonApplication.java

```
1 public class MonApplication {
2
3     public static void main (String arg[]) {
4         System.out.println ("Bienvenue dans mon application de gestion ressources humaines");
5
6         Employe employe1 = new Employe() ;
7
8         System.out.println ("Employe 1 : " + employe1.getNom() + " " + employe1.getPrenom() ) ;
9
10        employe1.setPrenom("Serge") ;
11        employe1.setNom("KOMI") ;
12
13        System.out.println ("Employe 1 : " + employe1.getNom() + " " + employe1.getPrenom() ) ;
14    }
15 }
```



```
C:\ObjisCoteDivoire\seance11\code>java MonApplication
Bienvenue dans mon application de gestion ressources humaines
Employe 1 : null null
Employe 1 : KOMI Serge
C:\ObjisCoteDivoire\seance11\code>
```

# Questions Module1-Séance11

- **Qu'est ce programmer en Orienté Objet ?**
- **Point commun et différence Classe / Objet ?**
- **Qu'est ce que l'encapsulation ?**
- **Expliquez la visibilité **public****
- **Expliquez les visibilités **private**, **protected****
- **Expliquez l'instruction `String nom;`**
- **Qu'est ce qu'un Getter ?**
- **Qu'est ce qu'un Setter ?**
- **Comment se manifeste l'encapsulation**
  - **Dans le travail du concepteur ?**
  - **Dans le travail du développeur ?**
- **Mettez en évidence encapsulation dans code**