6 – La Programmation Orientée Objet

Objet/Classe – Héritage – Encapsulation - Polymorphisme

Méthode orientée objet

- La méthode orientée objet permet de concevoir une application sous la forme d'un ensemble d'objets reliés entre eux par des relations
- Lorsque que l'on programme avec cette méthode, la première question que l'on se pose plus souvent est :
- **«qu'est-ce que je manipule ? »,**
- Au lieu de **« qu'est-ce que je fait ? ».**
- L'une des caractéristiques de cette méthode permet de concevoir de nouveaux objets à partir d'objets existants.

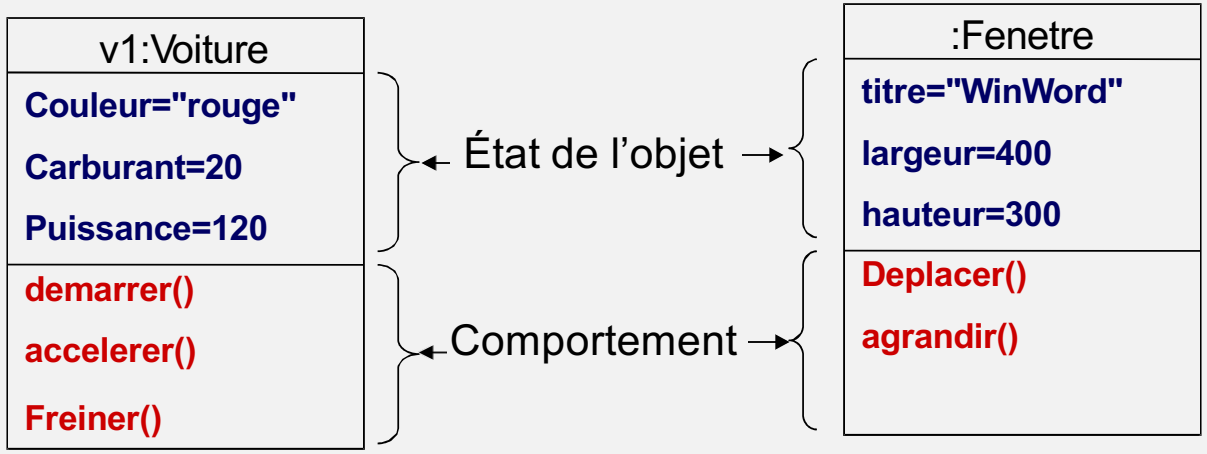
  On peut donc réutiliser les objets dans plusieurs applications.

  La réutilisation du code fut un argument déterminant pour venter les avantages des langages à objets.

  Pour faire la programmation orientée objet il faut maitriser les fondamentaux de l'orienté objet à savoir:
  - **Objet et classe**
  - **Héritage**
  - **Encapsulation (Accessibilité)**
  - **Polymorphisme**

Objet

- Un objet est une structure informatique définie par un état et un comportement
- Objet=état + comportement
  - L'état regroupe les valeurs instantanées de tous les attributs de l'objet.
  - Le comportement regroupe toutes les compétences et décrit les actions et les réactions de l'objet. Autrement dit le comportement est défini par les opérations que l'objet peut effectuer.
- L'état d'un objet peut changer dans le temps.
- Généralement, c'est le comportement qui modifie l'état de l'objet
- Exemples:



Identité d'un objet

- En plus de son état, un objet possède une identité qui caractérise son existence propre.
- Cette identité s'appelle également référence ou handle de l'objet
- En terme informatique de bas niveau, l'identité d'un objet représente son adresse mémoire.
- Deux objets ne peuvent pas avoir la même identité: c'est-à-dire que deux objet ne peuvent pas avoir le même emplacement mémoire.

Classes

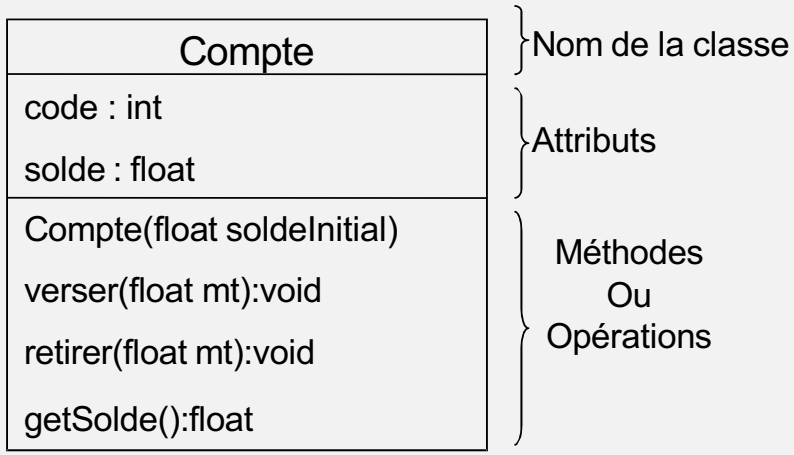
- Les objets qui ont des caractéristiques communes sont regroupés dans une entité appelé classe.
- La classe décrit le domaine de définition d'un ensemble d'objets.
- Chaque objet appartient à une classe
- Les généralités sont contenues dans les classe et les particularités dans les objets.
- Les objets informatique sont construits à partir de leur classe par un processus qui s'appelle l'instanciation.
- Tout objet est une instance d'une classe.

Caractéristique d'une classe

- Une classe est définit par:
  - Les attributs
  - Les méthodes
- Les attributs permettent de décrire l'état de des objets de cette classe.
  - Chaque attribut est défini par:
    - Son nom
    - Son type
    - Éventuellement sa valeur initiale
- Les méthodes permettent de décrire le comportement des objets de cette classe.
  - Une méthode représente une procédure ou une fonction qui permet d'exécuter un certain nombre d'instructions.
- Parmi les méthode d'une classe, existe deux méthodes particulières:
  - Une méthode qui est appelée au moment de la création d'un objet de cette classe. Cette méthode est appelée **CONSTRUCTEUR**
  - Une méthode qui est appelée au moment de la destruction d'un objet. Cette méthode s'appelle le **DESTRUCTEUR**

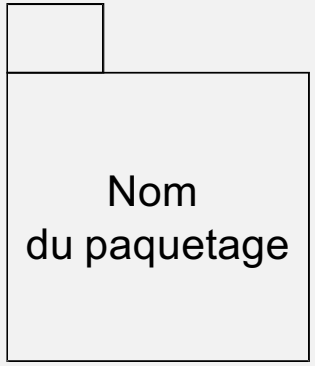
Représentation UML d'une classe

- Une classe est représenté par un rectangle à 3 compartiments:
  - Un compartiment qui contient le nom de la classe
  - Un compartiment qui contient la déclaration des attributs
  - Un compartiment qui contient les méthodes
- Exemples:



Les classes sont stockées dans des packages

- Les packages offrent un mécanisme général pour la partition des modèles et le regroupement des éléments de la modélisation
- Chaque package est représenté graphiquement par un dossier
- Les packages divisent et organisent les modèles de la même manière que les dossier organisent le système de fichier

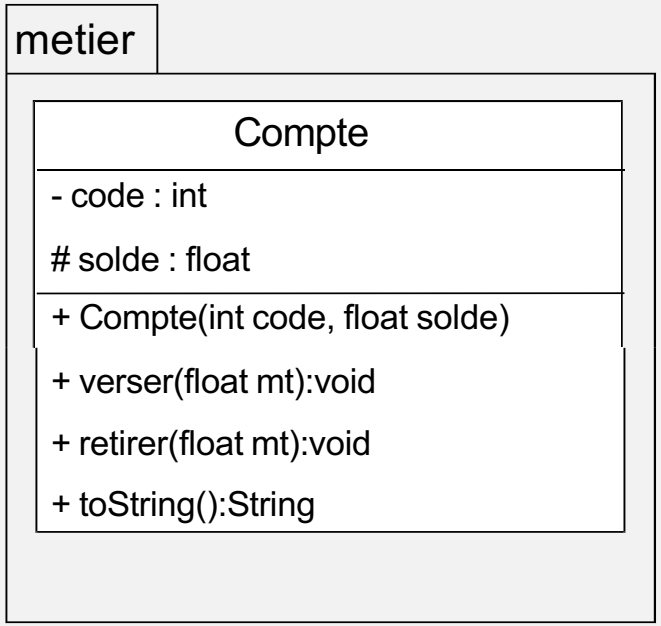


Accessibilité au membres d'une classe

- Dans java, il existe 4 **niveaux** de **protection** :
  - **private** (-) : Un membre privé d'une classe n'est accessible qu'à l'intérieur de cette classe.
  - **protected** (#) : un membre protégé d'une classe est accessible à :
    - L'intérieur de cette classe
    - Aux classes dérivées de cette classe.
  - **public** (+) : accès à partir de toute entité interne ou externe à la classe
  - **Autorisation par défaut** : dans java, en l'absence des trois autorisations précédentes, l'autorisation par défaut est **package.** Cette autorisation indique que uniquement les classes du même package ont l'autorisation d'accès.

Exemple d'implémentation d'une classe

avec Java

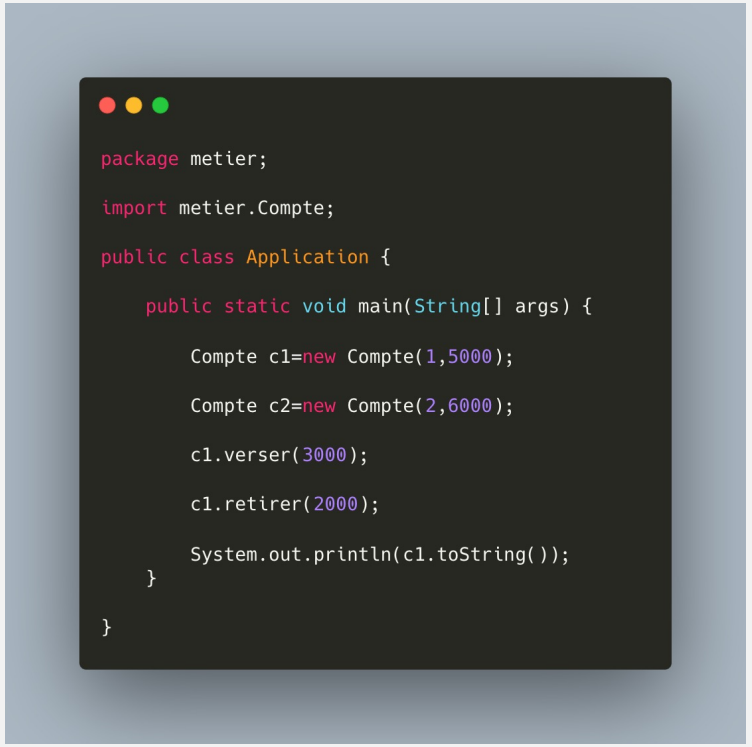



```java
package metier;

public class Compte {
    // Attributs
    private int code;
    protected float solde;
    // Constructeur
    public Compte(int c,float s){  code=c;
    solde=s;
    }
    // Méthode pour verser un montant
    public void verser(float mt){  solde+=mt;
    }
    // Méthode pour retirer un montant
    public void retirer(float mt){  solde-=mt;
    }
    // Une méthode qui retourne l'état du compte
    public String toString(){
    return(" Code="+code+" Solde="+solde);
}
```

Création des objets dans java

- Dans java, pour créer un objet d'une classe , On utilise la commande new suivie du constructeur de la classe.
- La commande new Crée un objet dans l'espace mémoire et retourne l'adresse mémoire de celui-ci.
- Cette adresse mémoire devrait être affectée à une variable qui représente l'identité de l'objet. Cette référence est appelée handle.



```java
package metier;

import metier.Compte;

public class Application {

    public static void main(String[] args) {

        Compte c1=new Compte(1,5000);

        Compte c2=new Compte(2,6000);

        c1.verser(3000);

        c1.retirer(2000);

        System.out.println(c1.toString());
    }

}
```

c1:Compte
code=1 solde=6000
verser(float mt) retirer(float mt) toString()

c2:Compte

code=2

solde=6000

verser(float mt)

retirer(float mt)

toString()

Constructeur par défaut

- Quand on ne définit aucun constructeur pour une classe, le compilateur crée le constructeur par défaut.
- Le constructeur par défaut n'a aucun paramètre et ne fait aucune initialisation

```java
// Exemple de classe :
public class Personne {

// Les Attributs

    private int code;
    private String nom;

// Les  Méthodes

    public void setNom(String n){
      this.nom=n;
    }
    public String getNom(){
      return nom;
    }

}
```

Instanciation en utilisant le constructeur par défaut :

```java
Personne p = new Personne();
p.setNom("AZER");
System.out.println(p.getNom());
```

Getters et Setters

- Les attributs privés d'une classe ne sont accessibles qu'à l'intérieur de la classe.
- Pour donner la possibilité à d'autres classes d'accéder aux membres privés, il faut définir dans la classes des méthodes publiques qui permettent de :
  - lire la variables privés. Ce genre de méthodes s'appellent les **accesseurs** ou **Getters**
  - modifier les variables privés. Ce genre de méthodes s'appellent les **mutateurs** ou **Setters**
- Les getters sont des méthodes qui 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 toujours le même type que l'attribut correspondant.
  - Par exemple, dans la classe CompteSimple, nous avons défini un attribut privé :
    ```
    private String nom;
    ```
  - Le getter de cette variable est :
    ```
    public String getNom( ){ return nom;
    }
    ```
- Les setters sont des méthodes qui 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 qui est de meme type que la variable:
  - Exemple:
    ```
    public void setNom( String n ){
    this.nom = n;
    }
    ```

Encapsulation

```
public class Application {
  public static void main(String[] args) {
        Personne p=new Personne();
        p.setNom("AZER");
        System.out.println(p.getNom());
 }
}
```

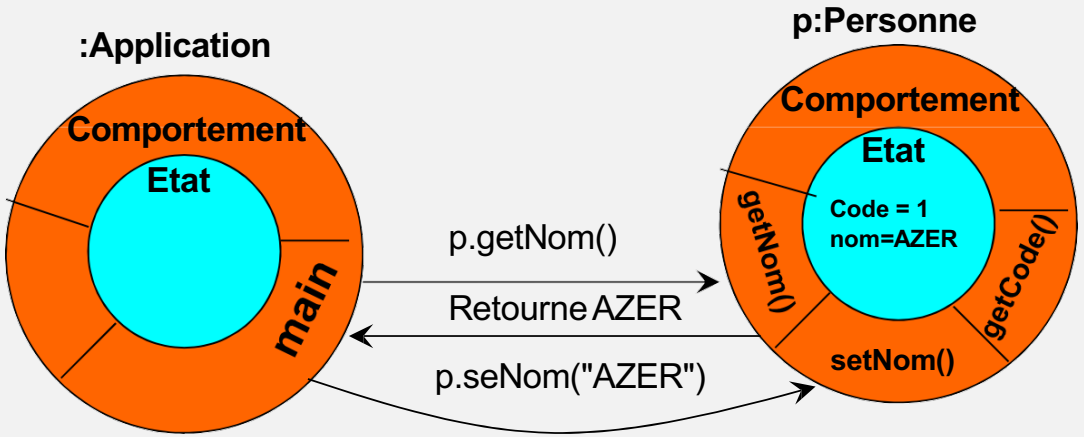
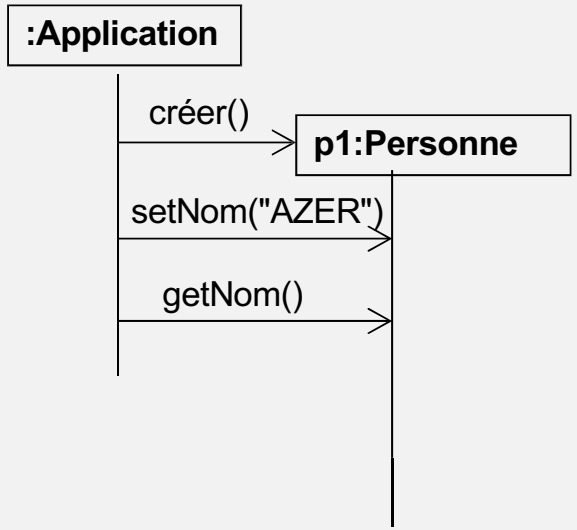


Diagramme de séquence :



- Généralement, l'état d'un objet est privé ou protégé et son comportement est publique
- Quand l'état de l'objet est privé Seules ses méthodes ont le droit d'y accéder
- Quand l'état de l'objet est protégé, les méthodes des classes dérivées.

Membres statiques d'une classe.

- Dans l'exemple de la classe Compte, chaque objet Compte possède ses propres variables code et solde. Les variables code et solde sont appelées variables d'instances.
- Les objets d'une même classe peuvent partager des mêmes variables qui sont stockées au niveau de la classe. Ce genre de variables, s'appellent les variables statiques ou variables de classes.
- Un attribut statique d'une classe est un attribut qui appartient à la classe et partagé par tous les objets de cette classe.
- Comme un attribut une méthode peut être déclarée statique, ce qui signifie qu'elle appartient à la classe et partagée par toutes les instances de cette classe.
- Dans la notation UML, les membres statiques d'une classe sont soulignés.

Exemple:

Compte
- code : int # solde : float - nbComptes:int
+ Compte(float solde) + verser(float mt):void + retirer(float mt):void + toString():String + getNbComptes():int

```java
package metier;
public class Compte {

// Variables d'instances
  private int code;
  private float solde;

// Variable de classe ou statique
  private static int nbComptes;

  public Compte(float solde){
     this.code=++nbComptes;
     this.solde=solde;
 }

// Méthode pour verser un montant
  public void verser(float mt){
    solde+=mt;
 }
// Méthode pour retirer un montant
  public void retirer(float mt){
    solde-=mt;
  }
// retourne l'état du compte
  public String toString(){
   return(" Code="+code+" Solde="+solde);
  }
// retourne la valeur de nbComptes
  public static int getNbComptes(){
              return(nbComptes);
  }
}
```

- Supposant nous voulions ajouter à la classe Compte une variable qui permet de stocker le nombre le comptes créés.
- Comme la valeur de variable nbComptes est la même pour tous les objets, celle-ci sera déclarée statique. Si non, elle sera dupliquée dans chaque nouveau objet créé.
- La valeur de nbComptes est au départ initialisée à 0, et pendant la création d'une nouvelle instance (au niveau du constructeur), nbCompte est incrémentée et on profite de la valeur de nbComptes pour initialiser le code du compte.

Application de test

```
package test;
import metier.Compte;
public class Application {
public static void main(String[] args) {
 Compte  c1=new  Compte(5000);
 Compte c2=new Compte(6000);
 c1.verser(3000);
 c1.retirer(2000);
 System.out.println(c1.toString());
 System.out.println(Compte.nbComptes)
 System.out.println(c1.nbComptes)
}
}
```

Classe Compte
nbCompte=2
getNbComptes()

c1:Compte	c2:Compte
code=1 solde=6000	code=2 solde=6000
verser(float mt) retirer(float mt) toString()	verser(float mt) retirer(float mt) toString()

Destruction des objets : Garbage Collector

- Dans certains langages de programmation, le programmeur doit s 'occuper lui même de détruire les objets inutilisables.
- Java détruit automatiquement tous les objets inutilisables en utilisant ce qu 'on appelle le **garbage collector (ramasseur d 'ordures).** Qui s 'exécute automatiquement dès que la mémoire disponible est inférieure à un certain seuil.
- Tous les objets qui ne sont pas retenus par des handles seront détruits.
- Ce phénomène ralenti parfois le fonctionnement de java.
- Pour signaler au garbage collector que vous vouler détruire un objet d'une classe, vous pouvez faire appel à la méthode finalize() redéfinie dans la classe.

Héritage et accessibilité

Héritage

- Dans la programmation orientée objet, l'héritage offre un moyen très efficace qui permet la réutilisation du code.
- En effet une classe peut hériter d'une autre classe des attributs et des méthodes.
- L'héritage, quand il peut être exploité, fait gagner beaucoup de temps en terme de développement et en terme de maintenance des applications.
- La réutilisation du code fut un argument déterminant pour venter les méthodes orientées objets.