Chapitre 2: Classes et Objets

Classes et Objets

Regrouper les objets

- Les objets qui collaborent dans une application sont souvent très nombreux
- Mais on peut le plus souvent dégager des types d'objets : des objets ont une structure et un comportement très proches, sinon identiques
- Par exemple, tous les livres dans une application de gestion d'une bibliothèque
- La notion de classe correspond à cette notion de types d'objets

Classes et Objets

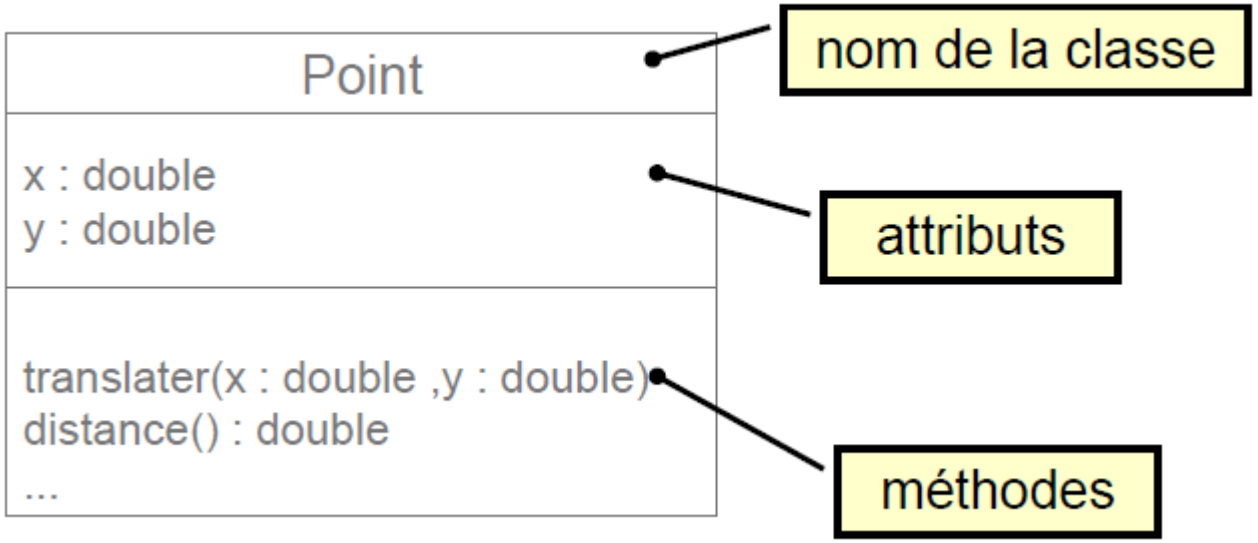
Classe

- Une classe est constituée de descriptions de :
  - *données : que l'on nomme **attributs**.*
  - *Operations: que l'on nomme **méthodes***
- Une **classe** est un **modèle** de définition pour des objets ayant une sémantique commune.
  - ayant même structure (même ensemble d'attributs),
  - ayant même comportement (mêmes opérations, méthodes),
- Les **objets** sont des représentations **dynamiques** (instanciation) du modèle défini pour eux au travers de la classe.
  - *Une classe permet d'**instancier** (créer) plusieurs objets*
  - *Chaque objet est **instance** d'une (seule) classe*

Classes et Objets

Classe

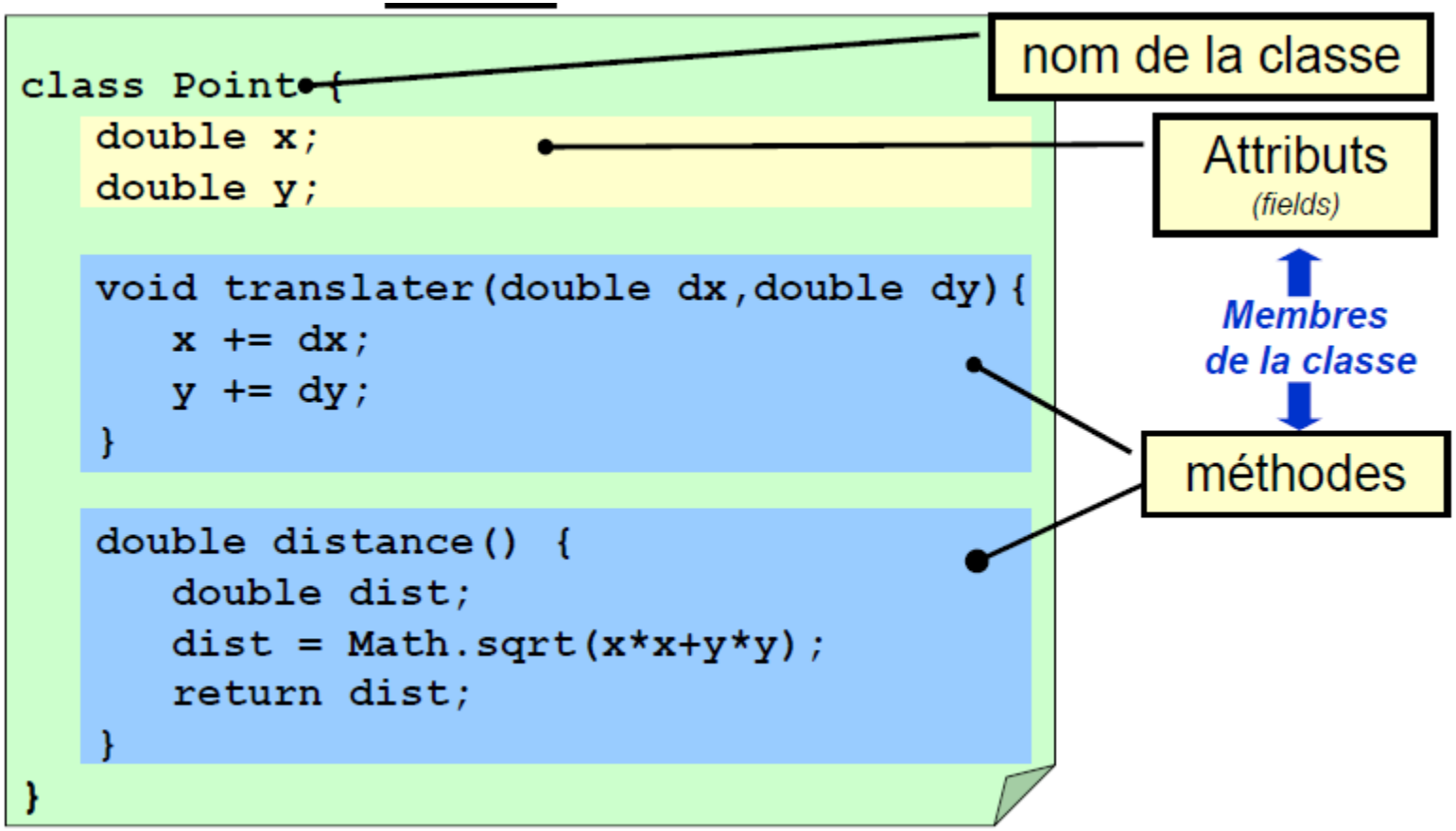
- Notation UML



Classes et Objets

Classe

- Syntaxe Java



Classes et Objets

Eléments d'une classe

- Les constructeurs (il peut y en avoir plusieurs) servent à créer les instances (les objets) de la classe
- Quand une instance est créée, son état est conservé dans les variables d'instance
- Les méthodes déterminent le comportement des instances de la classe quand elles reçoivent un message
- Les variables et les méthodes s'appellent les membres de la classe

Classes et Objets

Exemple : classe Livre

```
public class Livre {
  private String titre, auteur;
  private int nbPages;
  // Constructeur
  public Livre(String unTitre, String unAuteur) {
    titre = unTitre;
    auteur = unAuteur;
  }
  public String getAuteur() {       // accesseur
    return auteur;
  }
  public void setNbPages(int nb) { // modificateur
    nbPages = nb;
  }
}
```

Variables d'instance

Constructeurs

Méthodes

Classes et Objets

Rôles d'une classe

- Une classe est
  - un type qui décrit une structure (variables d'instances) et un comportement (méthodes)
  - un module pour décomposer une application en entités plus petites
  - un générateur d'objets (par ses constructeurs)
- Une classe permet d'encapsuler les objets : les membres **public** sont vus de l'extérieur mais les membres **private** sont cachés

Classes et Objets

Conventions pour les identificateurs

- Les noms de classes commencent par une majuscule (ce sont les seuls avec les constantes) :
  - **Cercle, Object**
- Les mots contenus dans un identificateur commencent par une majuscule :
  - **UneClasse, uneMethode, uneAutreVariable**
- Les constantes sont en majuscules avec les mots séparés par le caractère souligné « _ » :
  - **UNE_CONSTANTE**
- Si possible, des noms pour les classes et des verbes pour les méthodes

Classes et Objets

La référence

- Soit la classe :

```
public class Point {
private int x = 0;
private int y = 0;
}
```

- Création d'une instance (i.e. un objet)
  - Pour utiliser une classe il faut définir une instance de cette classe en utilisant new
    - new Point()
- Création d'une référence
  - Pour accéder à un objet, on utilise une référence à cet objet
    - Point p
  - p fait maintenant référence à un objet qui contient un champ x et un champ y, tous les deux initialisés à 0
- On peut déclarer une référence et définir une instance en une seule ligne :
  - Point p = new Point();
  - Integer i = new Integer(5);

Classes et Objets

Durée de vie

types primitifs :

```
{                                   aucun
    int x = 12;                     x
    {                               x
        int q = 96;                 x, q
    }                               x
}                                   aucun
```

objets :

```
{                                   aucun
    String s;                       s
    s = new String("abc");          s, "abc"
}                                   "abc"
```

- Les objets qui ne sont plus référencés sont détruits automatiquement par le “récupérateur de mémoire” qui fonctionne en arrière-plan

Classes et Objets

Les opérateurs et les objets

Affectation

```
public class Nombre {
public int i; //attribut
}
Nombre n1 = new Nombre ();
Nombre n2 = new Nombre ();
n1.i = 9; // affecter une valeur à un attribut
n2.i = 47; // affecter une valeur à un attribut
n1 = n2; //affecter une référence à une référence
n1.i = 27;
System.out.println(n1.i);
System.out.println(n2.i);
```

```
n1 = new Nombre ();
n2 = new Nombre ();
n1.i = 47;
n2.i = 47;
System.out.println(n1==n2);
System.out.println(n1.i == n2.i);
```

Les opérateurs et les objets

La méthode equals()

Sert à comparer les références

```
public static void main(String[] args){
    n1 = new Nombre ();
    n2 = new Nombre ();
    System.out.println(n1.equals(n2));
}
```

```
public static void main(String[] args){
        String s1 = new String("HELLO");
         String s2 = new String("HELLO");
         System.out.println(s1 == s2);
         System.out.println(s1.equals(s2));
```

Les opérateurs et les objets

La méthode compareTo()

- Sert à comparer deux objets
- Si s1 et s2 sont deux String, s1.compareTo(s2) renvoie
  - 0 si s1 et s2 sont égaux
  - nb>0 si s1>s2
  - nb<0 si S1< s2

```
public static void main(String[] args){
String s1 = new String("Hello");
         String s2 = new String("HELLO");
         System.out.println(s1.compareTo(s2));
         Integer i=5;
         System.out.println(i.compareTo(4));
         System.out.println(i.compareTo(5));
         System.out.println(i.compareTo(6));
         }
```

Classes et Objets

Classes et instances

- Une instance d'une classe est créée par un des constructeurs de la classe
- Une fois qu'elle est créée, l'instance
  - a son propre état interne (les valeurs des variables)
  - partage le code qui détermine son comportement (les méthodes) avec les autres instances de la classe

Classes et Objets

Constructeur

- **Constructeurs** d'une classe :
  - méthodes particulières pour la création d'objets de cette classe
  - méthodes dont le nom est identique au nom de la classe
- rôle d'un constructeur
  - *effectuer certaines initialisations nécessaires pour le nouvel objet créé*
- toute classe JAVA possède au moins un constructeur
  - *si une classe ne définit pas explicitement de constructeur, un constructeur par défaut sans arguments et qui n'effectue aucune initialisation particulière est invoqué*

Classes et Objets

Constructeurs

Création d'une instance

```
public class Employe {
private String nom, prenom;
private double salaire;
// Constructeur
public Employe(String n, String p) {
nom = n;
prenom = p;
}
public void setSalaire(double s){
salaire=s;
}
. . .
public static void main(String[] args) {
Employe e1;
e1 = new Employe("Dupond", "Pierre");
e1.setSalaire(12000);
. . .
}}
```

Variables d'instance

création d'une instance de Employe

Classes et Objets

Constructeurs

Plusieurs constructeurs (surcharge)

```
public class Employe {
private String nom, prenom;
private double salaire;
// Constructeur 1
public Employe(String n, String p) {
nom = n;
prenom = p;
}
// Constructeur 2
public Employe(String n, String p, double s) {
nom = n;
prenom = p;
salaire = s;
}
. . .
e1 = new Employe("Dupond", "Pierre");
e2 = new Employe("Durand", "Jacques", 15000);
```

Constructeurs

Plusieurs constructeurs (surcharge)

```
public class Point {
private int x;
private int y;
// Constructeur 1
public Point() {
x = 0;
y = 0;
}
// Constructeur 2
public Point(int coord1, int coord2) {
x = coord1;
y = coord2;
}}
```

```
public class TestPoint {
public static void main (String []
args) {
Point p,q;
p = new Point();
q = new Point(2,8);
}}
```

Constructeurs

L'objet « courant » : this

- Chaque objet a accès à une référence à lui même
  - la référence this
- Dans le corps d'une méthode, le mot clé **this** désigne l'instance sur laquelle est invoquée la méthode. Ce mot clé est utilisé dans 3 circonstances :
  1. pour accéder aux attributs de l'objets
  2. pour comparer la référence de l'objet invoquant la méthode à une autre référence
  3. pour passer la réference de l'objets invoquant la méthode en paramètre d'une autre méthode

```
constructeur(int maVar){
this.maVar=maVar;
}
```

Constructeurs

L'objet « courant » : this

Implicitement quand dans le corps d'une méthode un attribut est utilisé, c'est un attribut de l'objet courant

this essentiellement utilisé pour lever les ambiguités

```
class Point {
    double x;
    double y;
    void translater(int dx, int dy) {
        x += dx; y += dy;
    }
        <==>  this.x += dx; this.y += dy;

    double distance() {
        return Math.sqrt(x*x+y*y);
    }
    void placerAuPoint(double x ,double y1){
        this.x = x;
        y = y1;
    }
}
```

Classes et Objets

Constructeurs

Un constructeur peut appeler d'autres constructeurs

```
public class Point {
   private double x;
   private double y;

   // constructeurs
   private Point(double x, double y) {

      this.x = x;
      this.y = y;
   }

   public Point(Point p) {

      this(p.x,p.y);
   }

   public Point() {

      this(0.0,0.0);
   }

   ...
}
```

Intérêt :

- Factorisation du code.
- Un constructeur général invoqué par des constructeurs particuliers.

Possibilité de définir des constructeurs privés

Constructeurs

Constructeur par défaut

- Un constructeur sans paramètre est appelé constructeur par défaut explicite
- Conseil :
  - Créer toujours un constructeur même s'il est sans paramètre : programmation propre

Méthodes

- Deux types de méthodes servent à donner accès aux variables depuis l'extérieur de la classe :
  - les accesseurs en lecture pour lire les valeurs des variables ; « accesseur en lecture » est souvent abrégé en « accesseur » ; ***getter* en anglais**
  - les accesseurs en écriture, ou modificateurs, ou mutateurs, pour modifier leur valeur ; ***setter* en anglais**

Classes et Objets

Méthodes

```
public class Employe {
private double salaire;
. . .
public void setSalaire(double unSalaire) {
if (unSalaire >= 0.0)
salaire = unSalaire;
}
public double getSalaire() {
return salaire;
}
…
}
```

Modificateur

Accesseur

Classes et Objets

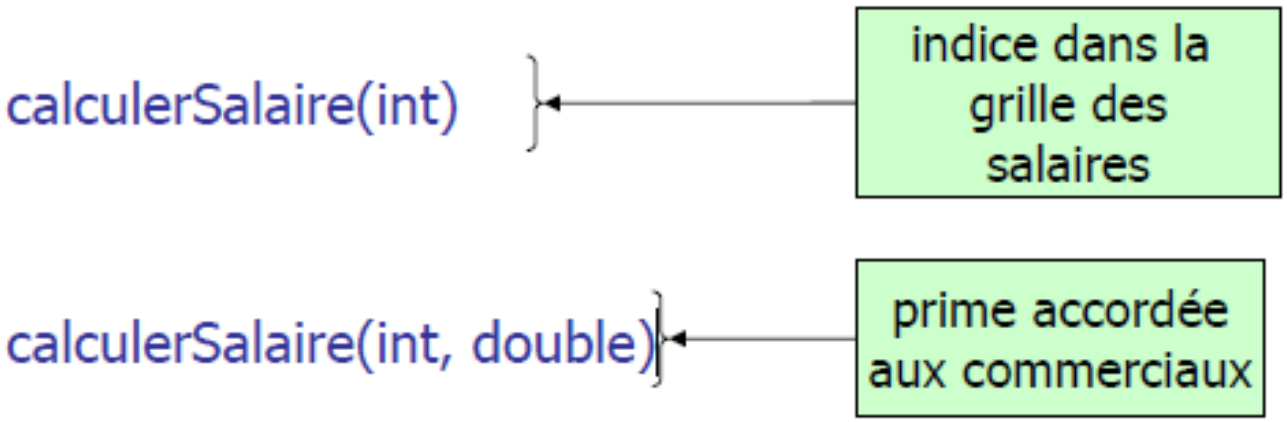
Méthodes

- Sont définies par
  - le nom
  - les paramètres avec leur types (optionnel)
  - le type de retour
  - un qualificateur (public, protected, private)
  - et le corps de la méthode (instructions)
- Utilisation
  - référence.méthode(arguments)
    - p.setX(68);

Méthodes

Surcharge d'une méthode

- En Java, on peut surcharger une méthode, c'est-à-dire, ajouter une méthode qui a le même nom mais pas la même signature qu'une autre méthode :



Méthodes

Surcharge d'une méthode

- En Java, il est interdit de surcharger une méthode en changeant le type de retour
- Autrement dit, on ne peut différencier deux méthodes par leur type de retour
- Par exemple: il est interdit d'avoir ces deux méthodes dans une classe

  - int calculerSalaire(int)
  - double calculerSalaire(int)

toString()

- Il est conseillé d'inclure une méthode **toString** dans toutes les classes que l'on écrit
- Cette méthode renvoie une chaîne de caractères qui décrit l'instance
- Une description compacte et précise peut être très utile lors de la mise au point des programmes
- **System.out.println(*objet*)** affiche la valeur retournée par ***objet*.toString()**

Classes et Objets

toString()

Exemple

```
public class Livre {
...
public String toString() {
return "Livre [titre=" + titre + ",auteur=" +
auteur + ",nbPages=" + nbPages + "]";
}
}
```

Classes et Objets

Les variables

Types de variables

- 1. Locales
  - Déclarées dans une méthode, un constructeur ou un bloc
  - N'existent que localement, ne sont pas visibles de l'extérieur
  - Sorties de leur espace, elles n'existent plus, la mémoire correspondante est libérée
- 2. Variables de classe ou variable static
  - Déclarées avec le mot clé static dans une classe, mais en dehors d'une méthode
  - Définies pour l'ensemble du programme et sont visibles depuis toutes les méthodes
- 3. d'instance
  - Variables qui sont utilisées pour caractériser l'instance d'un objet créée par un constructeur
  - Déclarées dans une classe, mais en dehors des méthodes
  - Appelées aussi membres ou variables de champ
  - Créées quand un objet est créé et détruites quand l'objet est détruit

Variable locale ou variable d'instance ?

- Il arrive d'hésiter entre référencer un objet
  - par une variable locale d'une méthode
  - ou par une variable d'instance de la classe

- Si l'objet est utilisé par plusieurs méthodes de la classe, l'objet devra être référencé par une variable d'instance

Déclaration des variables

- Toute variable doit être déclarée avant d'être utilisée
- Déclaration d'une variable
  - On indique au compilateur que le programme va utiliser une variable de ce nom et de ce type
- Exemple
  - double prime;
  - Employe e1;
  - Point centre;

Classes et Objets

Les variables

- Déclaration des variables : syntaxe
  - **modificateur type identificateur** [= valeur];
- Le nom d'une variable
  - est appelé un identificateur obéissant à certaines règles syntaxiques
- Le type de la variable
  - correspond au type de données qu'elle devra contenir, en l'occurrence soit un type primitif (int, float, char, boolean, etc.), soit un type composite (String, File, Vector, Socket, Array, etc.)
- La valeur de la variable
  - est appelée un littéral, et doit être conforme à son type de données

Classes et Objets

Les variables

- Le modificateur
  - Précise l'accès à la variable

  = private, protected, public ou rien (par défaut)

- représente son type d'accessibilité au sein d'un programme

- D'autres modificateurs peuvent être ajoutés, il s'agît de :
  - **final** pour définir une constante,
  - **static** rendant la variable accessible par la classe entière et même sans l'instanciation de cette classe
  - **transient** interdisant la sérialisation de la variable
  - **volatile** empêchant la modification asynchrone de la variable dans un environnement multi-threads

Classes et Objets

Les variables

- Les modificateurs peuvent apparaître dans n'importe quel ordre
  - public int hauteur = 100;
  - static String unite = "mètres";
  - long population = 6000000000;
  - boolean reussite = false;
  - final char CR = '\r';
  - float = 10.87E5;
  - URL adresse = new URL("http://java.sun.com/");
  - private String[] tableau;
  - transient private long code_secret;
- Il est interdit de définir deux fois la même variable au sein d'une même portée

Les variables

Affectation

- L'affectation d'une valeur à une variable est effectuée par l'instruction

  variable = expression;

- L'expression est calculée et ensuite la valeur calculée est affectée à la variable
- Exemple :
  - x = 3;
  - x = x + 1;

Les variables

Initialisation

- Une variable doit être initialisée
  - i.e. recevoir une valeur avant d'être utilisée dans une expression
- Si elles ne sont pas initialisées par le programmeur, les variables d'instance reçoivent les valeurs par défaut de leur type
  - 0 pour les types numériques, par exemple
- L'utilisation d'une variable locale non initialisée par le programmeur provoque une erreur
  - pas d'initialisation par défaut
- On peut initialiser une variable en la déclarant
- La formule d'initialisation peut être une expression complexe :
  - `double prime = 2000.0;`
  - `Employe e1 = new Employe("Dupond", "Jean");`
  - `double salaire = prime + 5000.0;`

Les variables

Déclaration / création

```
public static void main(String[] args) {
Employe e1;
e1.setSalaire(12000);
```

Il ne faut pas confondre

déclaration d'une variable et création d'un objet référencé par cette variable

« Employe e1; »

déclare que l'on va utiliser une variable **e1** qui référencera un objet de la classe **Employe**, mais aucun objet n'est créé

Il aurait fallu écrire :

```
public static void main(String[] args) {
Employe e1;
e1 = new Employe("Dupond", "Pierre");
e1.setSalaire(12000);
. . .}
```

Les variables

Désigner les variables d'une instance

Soit un objet **o1** ; la valeur d'une variable **v** de **o1** est désignée par **o1.v**

Par exemple,

```
Cercle c1 = new Cercle(p1, 10);
System.out.println(c1.rayon); // affiche 10
```

Remarque : le plus souvent les variables sont **private** et on ne peut pas y accéder directement en dehors de leur classe

Degrés d'encapsulation

- L'encapsulation est le fait de ne montrer et de ne permettre de modifier que ce qui est nécessaire à une bonne utilisation

- Java permet plusieurs degrés d'encapsulation pour les membres (variables et méthodes) et les constructeurs d'une classe

Degrés d'encapsulation

Types d'autorisation d'accès

- **private** : seule la classe dans laquelle il est déclaré a accès (à ce membre ou constructeur)
- **public** : toutes les classes sans exception y ont Accès
- Sinon, **par défaut**, seules les classes du même paquetage que la classe dans lequel il est déclaré y ont accès (un paquetage est un regroupement de classes ; notion étudiée plus loin dans le cours)
- **protected** sera étudié dans le cours sur l'héritage

Protection de l'état interne d'un objet

- Autant que possible l'état d'un objet (les variables d'instance) doit être **private**
- Si on veut autoriser la lecture d'une variable depuis l'extérieur de la classe, on lui associe un accesseur, avec le niveau d'accessibilité que l'on veut
- Si on veut autoriser la modification d'une variable, on lui associe un modificateur, qui permet la modification tout en contrôlant la validité de la modification

Variables de classe

- Certaines variables sont partagées par toutes les instances d'une classe. Ce sont les variables de classe (modificateur **static**)
- Si une variable de classe est initialisée dans sa déclaration, cette initialisation est exécutée une seule fois quand la classe est chargée en mémoire

Méthodes et variables de classe

Variables de classe

```
public class Employe {
private String nom, prenom;
private double salaire;
private static int nbEmployes = 0;
// Constructeur
public Employe(String n, String p) {
nom = n;
prenom = p;
nbEmployes++;
}
. . .
}
```

Méthode de classe

- Une méthode de classe (modificateur **static** en Java) exécute une action indépendante d'une instance particulière de la classe

- Une méthode de classe peut être considérée comme un message envoyé à une classe

- Exemple :

```
public static int getNbEmployes() {
        return nbEmployes;
}
```