جامعة هواري بومدين
للعلوم والتكنولوجيا
U S T H B

Chapitre 3

Modificateurs et Niveaux de visibilité

S. BOUKHEDOUMA

USTHB – FEI – département d'Informatique
Laboratoire des Systèmes Informatiques -LSI
sboukhedouma@usthb.dz

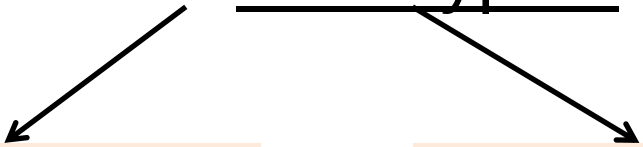
Modificateur ?

Modificateur: Mot clé du langage qui a une signification précise

- **Le modificateur static :** permet de définir des attributs/méthodes de classe (dits attributs/méthodes statiques)
- **Le modificateur final:** permet de définir des attributs constants

Le modificateur static

En java, il existe deux types d'attributs



Attribut d'instance

Spécifique à chaque objet de la classe

Aucun mot clé nécessaire

Attribut de classe

Commun (ou partagé) à tous les objets de la classe

Un attribut statique est déclaré avec le mot clé **static**

Le modificateur static

Exemple d'une classe avec attribut static

```
class Segment
{  Point debut, fin;
   static String couleur;

//constructeur
Segment (Point d, Point f)
{ debut = new Point (d.x, d.y);
   fin = new Point (f.x, f.y);
   couleur = "green";}

//méthode de classe
static void setCouleur (String col)
      {couleur = col;} }
```

debut et fin: sont des attributs d'instance

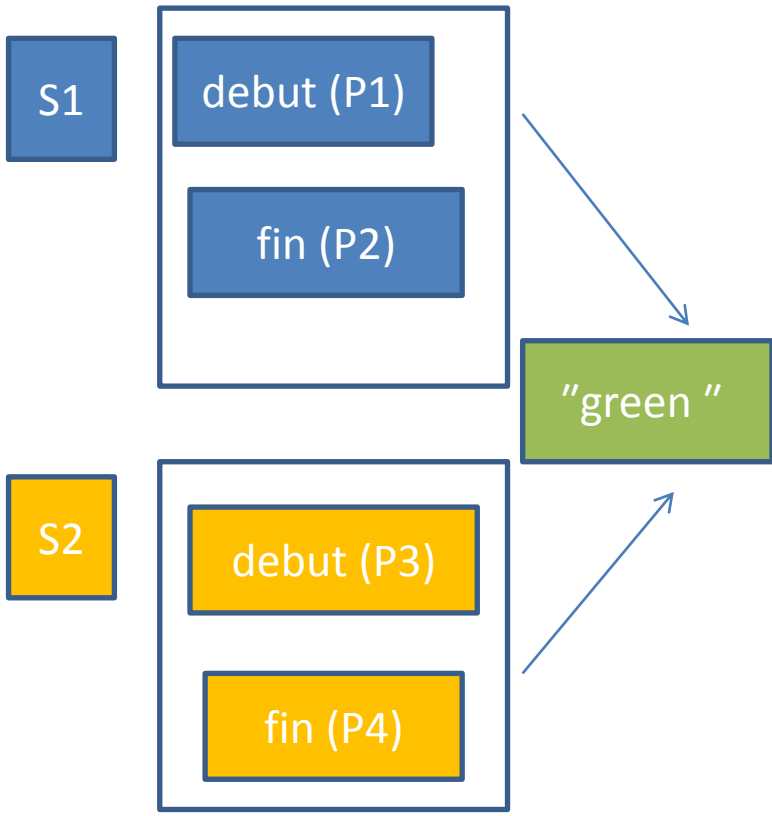
Couleur: est un attribut de classe (static)

La méthode setCouleur qui modifie la couleur est une **méthode static**

Le modificateur static

Dans une méthode main, on peut avoir

```
Point P1 = new Point (2, -5);
Point P2 = new Point (1, 1);
Point P3 = new Point(2, 2);
Point P4 = new Point(0, 6);

Segment S1 = new Segment (P1, P2);
Segment S2 = new Segment (P3, P4);
```



L'attribut couleur (**static)** existe en **un seul exemplaire**, c'est **une seule case mémoire partagée** par tous les objets de type Segment.

Le modificateur static

Donc tous les objets Segment auront la même couleur

Si on veut changer la couleur des segments, elle va changer pour **tous les segments créés**.

Il faut appeler la méthode **setCouleur** (static)

Une méthode static est appelée **avec le nom de la classe**

On écrira : **Segment.setCouleur** (“blue”);

Les objets S1 et S2 vont changer de couleur en même temps et auront la couleur “blue”.

Le modificateur static

En java, il existe deux types de méthodes

Méthode d'instance

Spécifique à chaque objet de la classe
Aucun mot clé nécessaire

Méthode de classe

Indépendante des objets de la classe
Une méthode de classe est déclarée avec le mot clé **static**

Une méthode de classe peut même être appelée sans création d'objets

Le modificateur static

Règles d'utilisation

- Les **attributs statiques** peuvent jouer le rôle de variables globales dans un programme.

- A l'intérieur de la classe (ou ses classes dérivées –voir chapitre sur l'héritage), les **attributs statiques** peuvent être utilisés directement (i.e sans précéder leurs noms du nom de la classe).

- Pendant l'exécution du programme, dès qu'une classe est invoquée, son « bytecode » est chargée en mémoire. Aussitôt, les **attributs statiques** de la classe sont créés en mémoire (en un seul exemplaire) et les **méthodes statiques** de la classe sont disponibles.

Le modificateur static

Règles d'utilisation (suite)

- Quel que soit le nombre d'instanciations (0 ou plusieurs objets crées), **un attribut statique existe en un seul exemplaire** car il a la même valeur pour toutes les instances créées.

- Une **méthode statique** peut être exécutée indépendamment d'une instanciation (sans créer des objets, comme les méthodes de la classe **Math, Math.pow(…), Math.sprt(…)**)

- Une **méthode statique** ne doit pas utiliser des attributs d'instance de sa classe, elle ne doit utiliser **que les attributs statiques**.

- Il faudra déclarer comme **statique (static**), une méthode qui n'utilise que **des champs statiques (déclarés static)**.

Le modificateur static

Autre exemple: compteur d'objets

On veut **numéroter les objets** créés à partir d'une classe de manière séquentielle à partir de 1.

Chaque objet aura un **numéro différent des autres**.

Il nous faut un **compteur** pour suivre la numérotation séquentielle.

Le compteur devra être déclaré **static** **pour l'incrémenter à chaque création d'objet**

Le modificateur static

Autre exemple: compteur d'objets

On reprend la classe **Point** déjà vue en cours et on ajoute la numérotation séquentielle des objets de type Point

```
class Point
{  int numéro; double x, y;
   static int cpt;

//constructeur
  Point (double x, doible y)
   { this.x = x; thix.y = y;
      cpt++;
      numéro = cpt;}
 … // méthodes
 }
```

Numéro, x, y : sont des attributs d'instance

cpt: est un attribut de classe (static)

A chaque création d'objet, le constructeur incrémente cpt et donne un nouveau numéro à l'objet Point créé

Le modificateur static

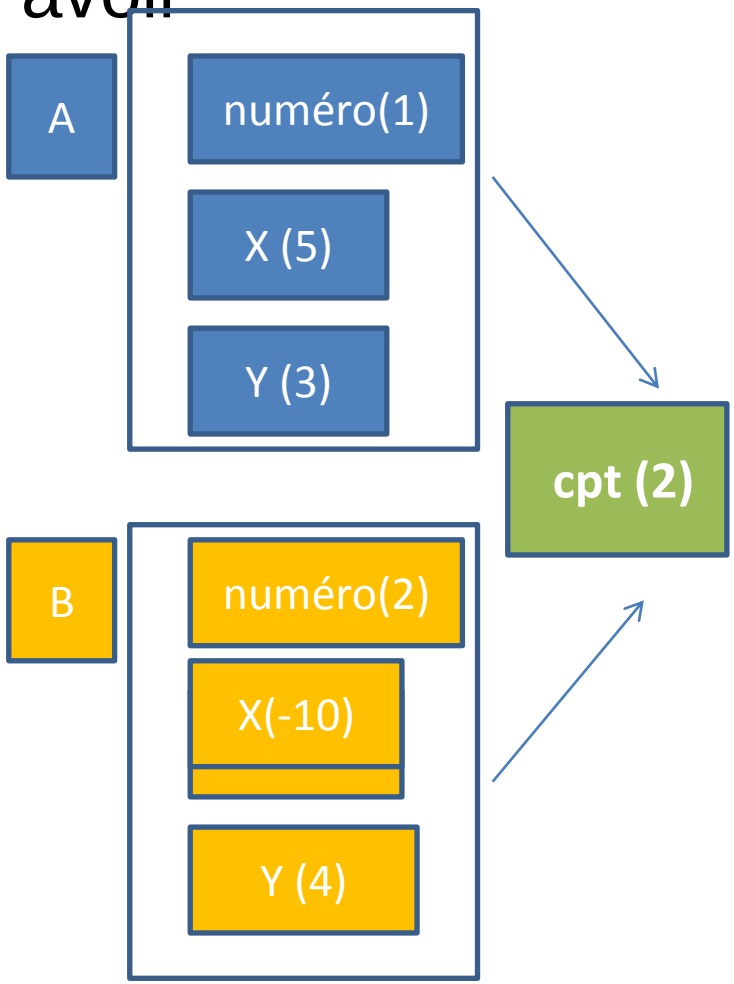
Dans une méthode main, on peut avoir

```
Point A = new Point (5, 3);
Point B = new Point (-10, 4);
```

L'objet A a le **numéro 1**
L'objet B a le **numéro 2**

Si on crée un autre objet P (-6, 10), il aura le numéro 3
Et ainsi de suite…

Dans **cpt,** on a toujours la **dernière valeur**



Le modificateur static

Exemple: méthode **d'instance** et méthode **de classe**

On reprend la classe Point déjà définie avec deux attributs x et y.

On ajoute à la classe Point, deux méthodes **égal** qui vérifient l'égalité de deux objets de type Point:

-Une méthode **d'instance égal**

- Une méthode **de classe égal**

Le modificateur static

Exemple: méthode **d'instance** et méthode **de classe**

```
class Point
{double x, y;
//constructeur
  Point (double x, double y)
   { this.x = x; thix.y = y;}
 … // méthodes
```

Une méthode static ne peut pas utiliser la référence "this", on doit faire passer les deux paramètres P1 et P2.

```
boolean égal (Point P2)
 { return (this.x == P2.x && thix.y == P2.y);}

static boolean égal (Point P1, Point P2)
 { return (P1.x == P2.x && P1.y == P2.y);}
}
```

Le modificateur static

Dans une méthode main, on peut avoir:

```
Point A = new Point (5, 3);
Point B = new Point (-10, 4);
```

On peut vérifier l'égalité entre deux points, en utilisant la méthode **égal d'instance** ou la méthode **égal de calsse**

```
if (P1.égal(P2) == true) …          //méthode d'instance

if (Point.égal(P1, P2) == true) … // méthode de classe
```

Le modificateur final

Le mot clé **final** est utilisé pour définir des attributs constants (constantes)

Une constante est une variable qui **ne peut pas changer de valeur**

Le modificateur final

En java, il existe deux types de constantes

Constante d'instance

Spécifique à chaque objet de la classe

Définie avec le mot clé **final**

Constante de classe

Commune (ou partagée) à tous les objets de la classe

Une constante de classe est déclarée avec les mots clés **final static**

Le modificateur final

Exemple de constantes

```
class TestConst
{   int x;
    final int C1;
    final static  int C2 = 100;

//constructeur
  TestConst (int x, int c)
   { this.x = x; C1 = c;}

… // méthodes }
```

X est un attribut d'instance

C1 est une constante d'instance. La valeur de C1 peut être différente d'un objet à l'autre mais ne peut pas changer après son initialisation

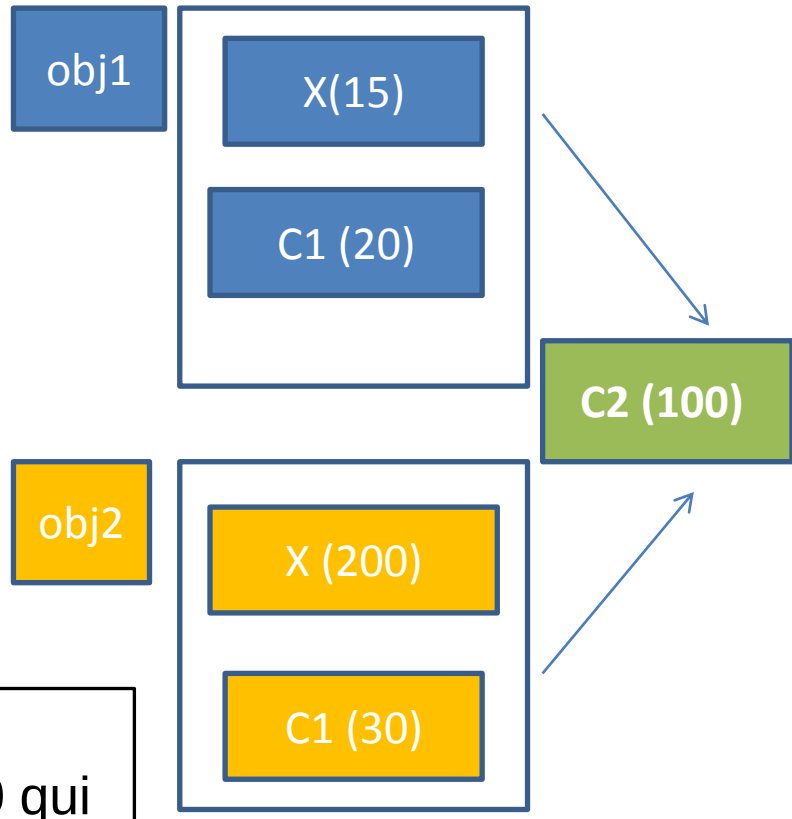
C2 est une constante de classe (**static)** existe en **un seul exemplaire**, c'est **une seule case mémoire partagée** par tous les objets de type TestConst.

C2 doit être initialisée à la déclaration

Le modificateur final

Dans une méthode main, on peut avoir

```
TestConst obj1 =
        new TestConst (15, 20);
TestConst obj2 =
        new TestConst (200, 30);
```



La constante C1 est propre à chaque objet et peut avoir une **valeur différente** pour chaque objet (20 et 30) mais ne change pas.

La constante C2 est la même pour tous les objets de la classe et possède la **valeur 100** qui ne change pas

Le modificateur final

Dans la classe Math, est définie une constante de classe appelée **PI** qui vaut 3,14
Définie comme suit: final static double PI= 3,14

Pour l'utiliser, on écrit **Math.PI**

Dans la classe Integer, sont définies deux constantes de classe appelées MINVALUE et MAXVALUE

Pour les utiliser, on écrit **Integer.MINVALUE** et **Integer.MAXVALUE**

Les modificateurs de visibilité

Les modificateurs de visibilité

Notion de package

Les modificateurs de visibilité sont liés à la **notion de package**

Un **package** est une structure logique qui regroupe un **ensemble de classes**

Les classes de la librairie fournie par java sont organisées en packages

Les modificateurs de visibilité

Quelques packages de java

java.lang : comporte les classes System, Integer, Float, Math, String, StringBuilder, …

java.io : comporte les classes File, Writer, Reader, …

java.util : comporte les classes Scanner, Random, Vector, ArrayList, Arrays, …

java.awt : comporte les classes Menu, Window, Button, MenuBar, Color, …

Le programmeur peut aussi organiser ses classes en packages

Les modificateurs de visibilité

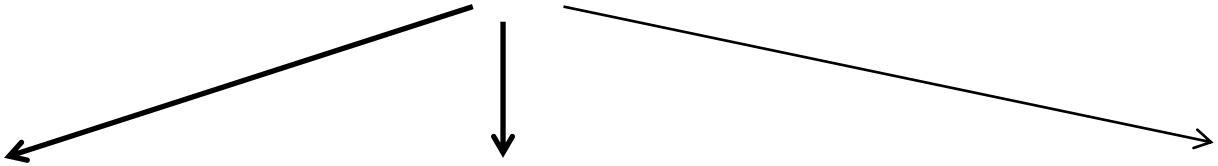
Pour utiliser une classe appartenant à un package externe, il **faut importer la classe**

Exemples

```
import java.util.Scanner;
import java.awt.Color;
import java.util.Random;
import Pack1.Maclasse;    //importer une classe créée par le programmeur
import java.util.*;     //importer toutes les classes du package java.util
```

Les modificateurs de visibilité

En java, On distingue 3 cas



La visibilité d'une classe

La visibilité d'une méthode

La visibilité d'un attribut

La visibilité consiste en la possiblité d'utiliser une classe, une méthode ou un attribut dans:

- **la même classe**,
- **une autre classe du même package**
- ou **une autre classe d'un package externe**

Les modificateurs de visibilité

Les niveaux de visibilité d'une classe

Pour une classe, On distingue 2 niveaux

La visibilité publique "public"

La visibilité par défaut (aucun mot clé)

Une classe **"public"** est visible dans toutes les classes du **même package** - **et dans toutes les classes de packages externes (il faut l'importer)**

Une classe qui n'est pas **"public"** est visible uniquement dans toutes les classes du **même package**

Les modificateurs de visibilité

Les niveaux de visibilité d'une classe

Pack1;

```
class A
 {…}

public class B
{…
  { …A a = new
          A(…);… }
…}
```

Pack 2;

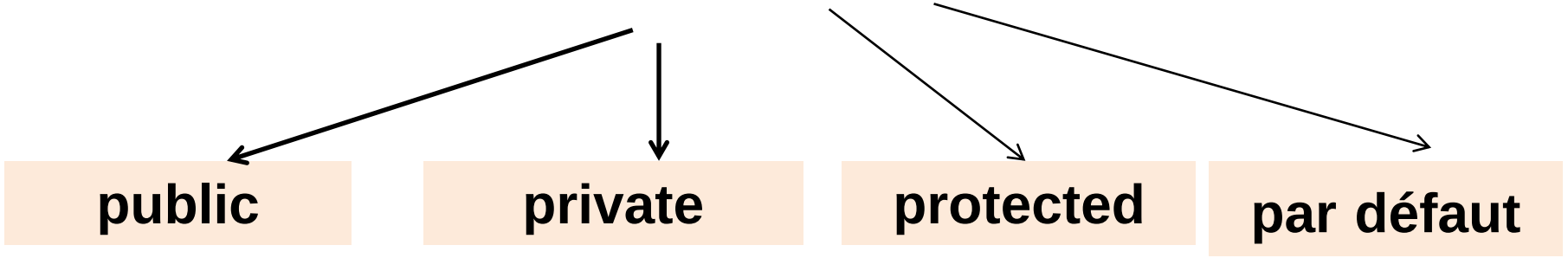
```
Import Pack1.B;  // on importe  la classe B

class C
 {…
   { … B b = new  B(…);  }
     // la classe B est utilisée dans un
package externe Pack2
…}
 public class D  {…}
```

La classe A n'est pas publique, elle ne peut pas être utilisée dans un package **externe (Pack2)**. Elle a la visibilité **package (par défaut)**

Les modificateurs de visibilité

Les niveaux de visibilité d'une méthode/attribut

On distingue quatre niveaux de visibilité



Le niveau "**protected" est lié à l'héritage** (classes dérivées) – voir chapitre 5.

Le niveau **"par défaut"** (ou le niveau package) n'a pas de modificateur spécifique (comme pour la classe)**.**

Les modificateurs de visibilité

Visibilité d'une méthode

Une méthode déclarée **"public"** est visible (i.e **peut être appelée**) dans:

- toutes les classes du **même package**
- **et** dans toutes les classes de **packages externes (si la classe qui la contient est "public")**

Une méthode **"par défaut" (aucun mot clé)** est visible uniquement dans toutes les classes du **même package**

Une méthode déclarée **"private"** est visible uniquement dans **sa propre classe**.

Les modificateurs de visibilité

Les niveaux de visibilité d'une méthode

Pack1;

```
class A
 {…
    private void
          meth1()
      {…}
      int meth2 ( )   // visibilité  package
           {…}
      public  int meth 3()
      {…}
}

public class B
{…
   { …A a = new  A(…);
              a.meth2(); a.meth3(), … }
  …}
```

Dans class B, on peut appeler les méthodes **meth2** et **meth3** (définies dans A), sur un objet de **type A**

On ne peut pas appeler la méthode **meth1** dans class B car la méthode **est privée (private)**

Pour appeler **meth3** dans une classe **externe (autre package)**, il faut déclarer la classe A **"public"** et il faut l'**importer**

Les modificateurs de visibilité

Visibilité d'un attribut

Un attribut déclaré **"public"** est visible (i.e **peut être référencé**) dans:

- toutes les classes du **même package**
- **et** dans toutes les classes de **packages externes (si la classe qui le contient est "public")**

Un attribut **"par défaut" (aucun mot clé)** est visible uniquement dans toutes les classes du **même package**

Un attribut déclaré **"private"** est visible uniquement dans **sa propre classe**.

Les modificateurs de visibilité

Visibilité d'un attribut

Un attribut d'instance est référencé précédé d'une référence d'objet de sa classe (il faut créer un objet de la classe)

Un attribut **de classe (static)** est référencé précédé du nom de la classe

Exemple : Math.PI (**PI** est un attribut **constant, static** et **public** défini dans la classe Math)

public final static double PI = 3,14;

Les modificateurs de visibilité

Accès aux attributs "private " – les accessurs

Un attribut qui est déclaré **"private**" ne peut être référencé **que dans sa prore classe**

Pour donner la possiblité d'y accéder dans une autre classe du même package (ou d'un package externe), on définit des méthodes **"public" get et set** qui permettent de lire ou de modifier l'attribut

Les méthodes **get et set** sont appelées **"accesseurs"**

Les modificateurs de visibilité

Exemple: attributs "private" – les accesseurs

Pack1

```
public class A
 {…
    private int  x;
    public  int getX()
      { return x;}
    public void setX(…)
       {x =…; }
}

public class B
{…
  { …A a = new  A(…);
         System.out.print (a.getX());}
…}
```

Pack 2;

```
Import Pack1.A;  // on importe la classe A

class C
 {…
   { … A a = new  a(…); }
        a.setX(…); }
     // la classe A est utilisée dans un
package externe Pack2
…
```

On accède à l'attribut x (private) via l'accesseur **getX** pour la lecture et **setX** pour la modification

Les modificateurs de visibilité

Principe d'encapsulation

Les attributs doivent être déclarés **"private"** afin de respecter le **principe d'encapsulation** (et contrôler leur utilisation via les accesseurs)

les attributs sont alors **cachés (encapsulés)** dans la classe qui les contient.

Ces attributs sont manipulés à travers les méthodes qui sont dans la classe.

Le programmeur **peut donner** accès à l'attribut:

- En **lecture** (via une méthode **get**)
- En **écriture** (via une méthode **set**
- **aucun accès** (ni get ni set, l'attribut est complètement caché)

Les modificateurs de visibilité

Règles générales

Il faut noter que

1. les attributs doivent être déclarés **"private" (ou protected – à voir plus loin)** pour respecter **l'encapsulation**

2. les méthodes doivent être déclarées **"public" (ou protected)** pour favoriser l'**utilisation/réutilisation**

3. Les classes doivent être déclarées **"public"** pour favoriser la **réutilisation**