POO : Séance 8
Les modificateurs final et static

1èr semestre – Année universitaire 2024/2025

Le modificateur final

• Ce modificateur permet d'indiquer les éléments du programme qui ne doivent pas être modifiés.

- ✓ Possible pour les classes, méthodes et variables.
- ✓ Utile surtout pour les variables.
- ✓ Moins courant pour les méthodes et les classes.

Variables finales : Exemple

```
public class cercle
{
      double rayon, surface;
      final double PI = 3.14;
      Cercle(double r)
      {
      rayon = r;
      surface = PI * rayon
      * rayon;
      PI = 3.15;
      }
}
```

Erreur!

```
public class ExempleFinal
{
      public static void main(String[] args)
      {
            final Cercle C;
            C = new Cercle(2.5);
            C = new Cercle(25.7);
            methode(1, 2);
      }
      public void methode(int a, final int b)
       {
            a = 3;
            b = 4;
       }
}
```

Objet final

OK première fois

Faux car deuxième fois

OK, car paramètre n'est pas final

Faux, car paramètre final

Méthodes finales

- Si l'on ajoute **final** à une méthode :

☞ Impossible de redéfinir (masquer) la méthode dans une sous-classe.

- Exemple : on aimerait que la calculerMoyenne() de la classe Cours soit appliquée de la même façon pour toutes les classes dérivées.

```
class Cours
{
      ...
      final double calculerMoyenne ()
       {
      return ((notecc*0.2) + (noteexam*0.8));
      }
}
```

```
Class RO extends Cours
{
double calculerMoyenne ()
       {
      return ((notecc*0.7) +
        (noteexam*0.3));
      }
      }
```

Erreur! Méthode finale
ne peut pas être redéfinie

Classes finales

- Si l'on ajoute **final** à une classe :

☞ Il devient impossible d'étendre la classe par une sous-classe

- Exemple : On aimerait que la classe RO n'ait jamais de sous-classe.

```
final class RO extends Cours
    {
    RO();
    }
```

- Une classe finale s'utilise comme d'habitude :

```
RO ro = new RO(...);
```

mais **Attention** : une classe finale ne peut jamais avoir de sous-classe !

```
final class Optimisation extends RO
    {…}
```

→ Erreur

Le modificateur static

- S'utilise pour les variables et les méthodes

- Si on ajoute static à une variable :

  -La valeur de la variable est partagée entre toutes les instances de la classe.

  -Pas possible pour les variables locales (d'instance).

- Si on ajoute static à une méthode :

  -On peut appeler la méthode sans construire d'objet.

  -Diverses restrictions sur le contenu de la méthode statique

Catégories de variables (1)

- Nos variables jusqu'à maintenant :

  - Variables d'instance :
    -Décrivent les attributs d'un objet.
    -Aussi appelées variables dynamiques.

  - Variables locales:
    -Déclarées à l'intérieur d'une méthode.

  - Paramètres:
    -Pour envoyer des valeurs à une méthode.
    -S'utilisent comme variables locales.

- Nouveau cette semaine:

  - Variables statiques = variables de classe

Variables d'instance et de classe

- La différence entre les variables d'instance et les variables de classe :

  ➔ Le nombre de zones réservées en mémoire

- Variable d'instance :

  ➔ Réservation d'une zone pour chaque objet construit avec new.

  ➔ Résultat : chaque objet a sa propre zone/valeur pour la variable d'instance.

- Variable de classe (variable statique):

  ➔ Déclaration précédé par static.
  ➔ Réservation d'une zone lors du chargement de la classe
  ➔ Aucune zone réservée quand un objet est construit avec new
  ➔ Résultat : tous les objets se réfèrent à la même zone/valeur pour la variable de classe

Exemple

```
class A
 {
      int b = 1;
      public static int c = 10;
      void modifier()
      {
            b++;
            c++;
      }
}
```

Variable d'instance

```
class Abc
 {
public static void main(String[] args)
      {
      A v1, v2, v3;
      v1 = new A();
      v2 = new A();
      v3 = new A();
      v1.modifier();
      v2.modifier();
      A.c++;
      }
}
```

Variable de classe

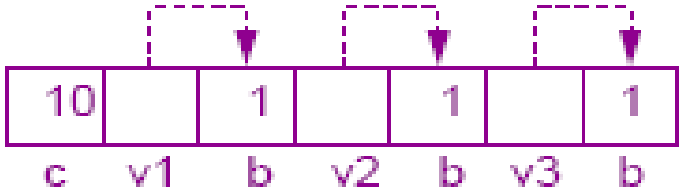
Exécution de l'exemple (1)

I. A v1, v2, v3;

- Dès que la classe A est mentionnée :
  -Réservation d'une zone spéciale pour les variables statiques de la classe.
  -Initialisation des variables statiques.

II. v1 = new A(); v2 = new A(); v3 = new A();

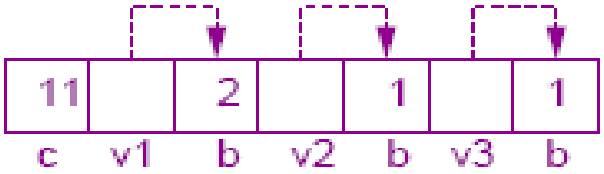
- Pour chaque objet construit ...
- Réservation d'une zone pour b
- Aucune réservation de zone pour c



Exécution de l'exemple (2)

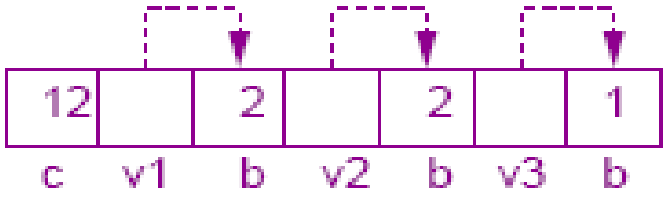
III. V1.modifier()

- Modification de la variable b de l'objet v1
- Modification de la variable c de la classe A



IV. V2.modifier()

- Modification de la variable b de l'objet v2
- Modification de la variable c de la classe A
  (même zone en mémoire que pour v1)



Exécution de l'exemple (3)

V. A.c++

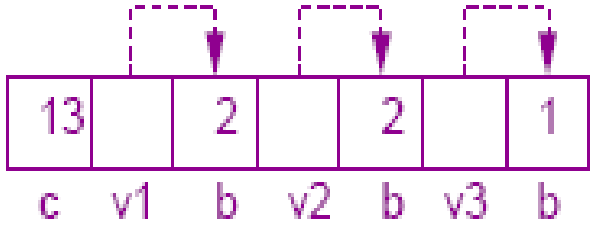
- Il est possible d'accéder à une variable statique par le nom de la classe, sans passer par un objet.

- Possibilité qui existe seulement pour les variables statiques.

- Possible puisque les variables statiques sont initialisées dès que la classe est mentionnée.

- On peut accéder à une variable statique même si aucun objet n'a été construit.

- Evidemment pas possible si la variable statique est aussi privée.



Pourquoi utiliser static

- Variable d'instance vs variable de classe :

  - Modification d'une variable d'instance : La valeur change seulement pour l'objet actuel.

  - Modification d'une variable de classe : La valeur change pour tous les objets de la classe

- A quoi sert une variable statique :

  - Bonne raison d'utiliser une variable statique : Représentation d'une valeur qui est commune à tous les objets de la classe. Exp: nombre d'objets créés

  - Mauvaise raison d'utiliser une variable statique : Programmer de manière non orientée objet en Java.

Constantes : final et static

- Puisqu'une variable finale ne peut pas être modifiée :

➔ Inutile de stocker une valeur pour chaque objet de la classe

➔ Une constante est normalement toujours final static

```
class Cercle
 {
      double rayon, surface;
      // Une variable PI pour chaque cercle:
      final double PI = 3.14;
      // Une variable PI pour tous les cercles:
      final static double PI = 3.14; // Beaucoup mieux!!
      Cercle(double r)
       {...}
}
```

Méthodes statiques

- Similairement, si on ajoute **static** à une méthode :

-On peut accéder à la méthode à travers un objet mais aussi sans objet

```
class A
{
void methode1()
{
System.out.println("Méthode 1");
}
static void methode2()
{
System.out.println("Méthode 2");
}
}
```

```
class ExempleMethodeStatique
{
public static void main(String[] args)
    {
        A v = new A();
        v.methode1();
        v.methode2();
        A.methode2();
        A.methode1();
    }
}
```

v.methode1(); → OK
v.methode2(); → OK
A.methode2(); → OK
A.methode1(); → Faux

Restrictions sur les Méthodes statiques

- Puisqu'une méthode statique peut être appelée avec ou sans objet :

  -Le compilateur ne peut pas être sûr que l'objet **this** (l'objet en cours) existe pendant l'exécution de la méthode.

  -Il ne peut donc pas admettre l'accès aux variables/méthodes d'instance (car elles dépendent de this).

- Conclusion pour les accès dans la même classe :

  -Une méthode statique peut seulement accéder à d'autres méthodes statiques et à des variables statiques

Utilité des méthodes statiques

- Méthodes qui ne sont pas liées à un objet.

- Exemple :

-Classe mettant à disposition des utilitaires mathématiques divers.
-La classe sert seulement à stocker des méthodes utilitaires.

```
class MathUtils
 {
      final static double PI = 3.14;
      static double puissanceTrois(double d)
      {
      return d*d*d;
      }
}
```

Utilité des méthodes statiques (2)

- Utilisation de la classe MathUtils :

-Calculer y = $\pi \cdot x^3$ pour x = 5.7.

-On peut accéder aux variables/méthodes statiques sans construire d'objet.

```
class Calcul
{
public static void main(String[] args)
{
double x = 5.7;
double y = MathUtils.PI *MathUtils.puissanceTrois(x);
System.out.println(y);
}
}
```

Utilité des méthodes statiques (3)

```
class Calcul
{
    public static void main(String[] args)
    {
        MathUtils m = new MathUtils();
        double x = 5.7;
        double y = m.PI * m.puissanceTrois(x);
        System.out.println(y);
    }
}
```

- Egalement possible mais inutile car l'objet m n'a pas d'intérêt pour le programme.