

ELÉMENTS DE PROGRAMMATION JAVA



Programme avec une seule classe = classe exécutable

2

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello world");  
    }  
}
```

- ❑ L'exécution du programme commence par l'exécution d'une classe qui doit implémenter une méthode particulière **public static void main(String[] args)**.
- ❑ L'exécution (après compilation) de cette classe se fait de la manière suivante :

```
C:\>java HelloWorld
```

- ❑ le tableau de chaînes de caractères **args** qui est un paramètre d'entrée de la méthode main contient des valeurs précisées à l'exécution.
- ❑ Si la classe avait été exécutée par la ligne de commande suivante :

```
C:\>java HelloWorld 4 Bonjour @ 123
```

Le tableau de chaînes de caractères contiendrait 4 éléments dont les valeurs seraient respectivement "4", "Bonjour", "@", "123" qui pourront être exploités dans le programme principale.

Programme avec plusieurs classes

3

```
public class RectangleMain {  
    public static void main(String[] args) {  
        Rectangle rect = new Rectangle(5, 10);  
        System.out.println("La surface est " + rect.surface());  
    }  
}  
  
class Rectangle {  
    int longueur;  
    int largeur;  
    int origine_x;  
    int origine_y;  
  
    Rectangle(int lon, int lar) {  
        this.longueur = lon;  
        this.largeur = lar;  
        this.origine_x = 0;  
        this.origine_y = 0;  
    }  
  
    void deplace(int x, int y) {  
        this.origine_x = this.origine_x + x;  
        this.origine_y = this.origine_y + y;  
    }  
  
    int surface() {  
        return this.longueur * this.largeur;  
    }  
}
```

Il est souhaitable d'implémenter **chaque classe dans un fichier séparé** et il est indispensable que **ce fichier ait le même nom que celui de la classe**.

- Dans l'exemple ci-dessus, deux fichiers ont ainsi été créés : RectangleMain.java et Rectangle.java.

- Compilation : `C:\>javac RectangleMain.java Rectangle.java`

- Exécution : `C:\>java RectangleMain`

Packages

4

- ❑ Un grand nombre de classes, fournies par Java SE, implémentent des données et traitements génériques utilisables par un grand nombre d'applications.
- ❑ Ces classes forment l'API (Application Programmer Interface) du langage Java. Une documentation en ligne pour l'API java SE 8 est disponible à l'URL : <http://docs.oracle.com/javase/8/docs/api/>
- ❑ Toutes ces classes sont organisées en **packages** (ou bibliothèques) dédiés à un thème précis afin de faciliter la modularité.
- ❑ Chaque paquetage porte un nom. Ce nom est soit un simple identificateur ou une suite d'identificateurs séparés par des points.

Exemples de packages Java

5

- ❑ Parmi les packages Java, on peut citer les suivants :

<code>java.applet</code>	Classes de base pour les applets
<code>java.awt</code>	Classes d'interface graphique AWT
<code>java.io</code>	Classes d'entrées/sorties (flux, fichiers)
<code>java.lang</code>	Classes de support du langage
<code>java.math</code>	Classes permettant la gestion de grands nombres.
<code>java.net</code>	Classes de support réseau (URL, sockets)
<code>java.rmi</code>	Classes pour les méthodes invoquées à partir de machines virtuelles non locales.
<code>java.security</code>	Classes et interfaces pour la gestion de la sécurité.
<code>java.sql</code>	Classes pour l'utilisation de JDBC.
<code>java.text</code>	Classes pour la manipulation de textes, de dates et de nombres dans plusieurs langages.
<code>java.util</code>	Classes d'utilitaires (vecteurs, hashtable)
<code>javax.swing</code>	Classes d'interface graphique

N.B : Le paquetage `java.lang` est importé automatiquement par le compilateur.

Comment accéder aux classes d'un package

6

- ❑ Pour accéder à une classe d'un package donné, il faut préalablement importer cette classe ou son package.
- ❑ Par exemple, la classe **Date** appartenant au package **java.util** qui implémente un ensemble de méthodes de traitement sur une date peut être importée de deux manières :
 - ❑ une seule classe du package est importée :

```
import java.util.Date;
```
 - ❑ Toutes les classes du package sont importées (même les classes non utilisées) :

```
import java.util.*;
```

Le programme suivant utilise cette classe pour afficher la date actuelle :

```
import java.util.Date;

public class DateMain {
    public static void main(String[] args) {
        Date today = new Date();
        System.out.println("Nous sommes le " + today.toString());
    }
}
```

Comment accéder aux classes d'un package

7

Remarque :

- `import java.awt.*;` : Cette instruction ne va pas importer de manière récursive les classes se trouvant dans `awt` et dans ses sous paquets. Elle va importer donc que les classes du package `awt`.
- Si vous avez besoin d'utiliser les classes de event, vous devez les importer aussi comme suit :
`import java.awt.event.*;`

Comment créer vos propres packages ?

8

- ❑ Il est possible de créer vos propres packages en précisant, avant la déclaration d'une classe, le package auquel elle appartient. Pour assigner la classe précédente à un package, nommé **fr.emse**.
- ❑ il faut modifier le fichier de cette classe comme suit :

```
package fr.emse;  
  
import java.util.Date;  
  
public class DateMain {  
    ...  
}
```

- ❑ Enfin, il faut que le chemin d'accès du fichier DateMain.java corresponde au nom de son package.
- ❑ Celui-ci doit donc être situé dans un répertoire fr/emse/DateMain.java accessible à partir des chemins d'accès définis lors de la compilation ou de l'exécution.

Visibilité des classes, variables (attributs) et méthodes

9

- En Java, comme dans beaucoup de langages orientés objet, les classes, les attributs et les méthodes bénéficient de niveaux d'accessibilité ou de visibilité, qui indiquent dans quelles circonstances on peut accéder à ces éléments.

Élément	Autorisations
Variable	Lecture et écriture
Méthode	Appel de la méthode
Classe	Instanciation d'objets de cette classe et accès aux variables et méthodes de classe

- Ces niveaux sont au nombre de 4, correspondant à 3 mots-clés utilisés comme modificateurs d'accès : **private**, **protected** et **public**. La quatrième possibilité est de ne pas spécifier de modificateur (comportement par **défaut**).

Modificateurs d'accès : private, protected, public

10

Le tableau résume les différents mode d'accès des membres d'une classe.

Modificateur du membre	<code>private</code>	<code>aucun</code>	<code>protected</code>	<code>public</code>
Accès depuis la classe	Oui	Oui	Oui	Oui
Accès depuis une classe du même package	Non	Oui	Oui	Oui
Accès depuis une sous-classe	Non	Non	Oui	Oui
Accès depuis toute autre classe	Non	Non	Non	Oui

- **private** : Un attribut ou une méthode déclarée private n'est accessible que depuis l'intérieur de la même classe.
- **protected** : Un attribut ou une méthode déclarée protected est accessible uniquement aux classes d'un package et à ses sous-classes même si elles sont définies dans un package différent.
- **public** : Une classe, un attribut ou une méthode déclarée public est visible par toutes les classes et les méthodes.

Exercice

11

```
package com.moimeme.temps;

class Horloge
{
    // corps de la classe
}

public class Calendrier
{
    void ajouteJour()
    {
        // corps de la methode
    }

    int mois;

    // suite de la classe
}
```

Questions :

- a. Quelle classes peuvent accéder à la classe **Horloge** (instancier la classe horloge) ?
Réponse : Toutes les classes du package **com.moimeme.temps**
- b. Quelle classes peuvent accéder à la classe **Calendrier** (instancier la classe Calendrier) ?
Réponse : Toutes les classes.
- c. Quelle classes peuvent accéder à la méthode **ajouterJour()** de la classe **Calendrier** ?
Réponse : Toutes les classes du package **com.moimeme.temps**
- d. Quelle classes peuvent accéder à l'attribut **mois** ? **Réponse** : Toutes les classes du package **com.moimeme.temps**

Modificateurs d'accès : private, protected, public

12

- ❑ En général, il est souhaitable que les modificateurs d'accès soient limités.
- ❑ Le modificateur d'accès public, qui est utilisé systématiquement par les programmeurs débutants, ne doit être utilisé que s'il est indispensable.
- ❑ Cette restriction permet d'éviter des erreurs lors d'accès à des méthodes ou de modifications de variables sans connaître totalement leur rôle.

Encapsulation

13

- ❑ L'**encapsulation** est un des piliers de la programmation orientée objet, il vise à regrouper les données et leur traitements associés dans des classes.
- ❑ L'**encapsulation** est un principe qui garantie qu'aucun champ(donnée) d'un objet ne pourra être modifier pour corrompre l'état d'un objet sans une vérification préalable.
- ❑ L'état de l'objet ne pourra être modifié (s'il est modifiable) que par des méthodes vérifiant les données rentrées.

Encapsulation ?

14

Exemple d'un point en coordonnée polaire :

(les deux classes appartiennent au même package)

```
class Point {  
    double rho;  
    double theta;  
  
    void init(double x,double y) {  
        rho=Math.hypot(x,y);  
        theta=Math.atan2(x,y);  
    }  
}
```

```
class AutreClass {  
    void autreMethod(Point p) {  
        p.init(2.0,3.0);  
        System.out.println(p.theta);  
  
        p.rho=-1; // aie aie !!  
    }  
}
```

- ❑ La classe Point ne garantie pas l'encapsulation !
- ❑ **Solution** : utiliser les modificateurs de visibilité.

Visibilité et encapsulation

15

Un champs est toujours privé !!

```
class Point {  
  
    private double rho;  
    private double theta;  
  
    public void init(double x,double y) {  
        rho=Math.hypot(x,y);  
        theta=Math.atan2(x,y);  
    }  
}
```

```
class AutreClass {  
    void autreMethod(Point p) {  
        p.init(2.0,3.0);  
        System.out.println(p.theta); // ne compile pas  
        p.rho=-1; // ne compile pas  
    }  
}
```

Visibilité et encapsulation

16

toString pour l'affichage!

```
class Point {  
  
    private double rho;  
    private double theta;  
  
    public void init(double x, double y) {  
        rho=Math.hypot(x,y);  
        theta=Math.atan2(x,y);  
    }  
  
    public String toString() {  
        return rho+", "+theta;  
    }  
}
```

```
class AutreClass {  
    void autreMethod(Point p) {  
        p.init(2.0,3.0);  
        System.out.println(p.toString());  
    }  
}
```


Mais il y a un problème

17

Supposons que l'on souhaite que rho ne soit jamais égal à zéro.

On peut obtenir la
valeur des champs
avant l'initialisation

```
class Point {  
    private double rho;  
    private double theta;  
  
    public void init(double x, double y) {  
        rho = Math.hypot(x, y);  
        theta = Math.atan2(x, y);  
    }  
    public String toString() {  
        return rho + ", " + theta;  
    }  
}
```

```
class AutreClass {  
    void autreMethod(Point p) {  
        System.out.println(p.toString()); // affiche 0,0 oups  
        p.init(2.0, 3.0);  
    }  
}
```

Solution: constructeurs

Les constructeurs

18

Le constructeur est écrit pour garantir les invariants.

```
class Point {  
    private double rho;  
    private double theta;  
  
    public Point(double rho, double theta) {  
        if (rho <= 0)  
            throw new IllegalArgumentException(  
                "illegal rho "+rho);  
        this.rho=rho;  
        this.theta=theta;  
    }  
    public String toString() {  
        return rho+" "+theta;  
    }  
}
```

On ne peut pas créer un objet sans appeler de constructeur.

```
public class AnotherClass {  
    public static void main(String[] args) {  
        Point p=new Point(2,3);  
  
        Point p2=new Point();  
        // cannot find symbol constructor Point()  
    }  
}
```

Les constructeurs

19

- ❑ Chaque constructeur doit avoir le même nom que la classe où il est défini et n'a aucune valeur de retour (c'est l'objet créé qui est renvoyé).
- ❑ Dans l'exemple suivant, les 3 constructeurs initialisent la valeur des données encapsulées :

Constructeur avec
paramètre

Constructeur par
défaut (sans
paramètre)

Constructeur avec
paramètre

```
public class Pixel {  
    private int x;  
    private int y;  
    public Pixel(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public Pixel() {  
        this(0,0);  
    }  
    public Pixel(int v) {  
        this(v, v);  
    }  
}
```

Constructeur par défaut par défaut

20

Si aucun constructeur n'est défini explicitement, le compilateur rajoute un constructeur public sans paramètre (Constructeur par défaut par défaut).

```
public class Point {  
    private double x;  
    private double y;  
  
    public static void main(String[] args) {  
        Point p=new Point(); // ok  
    }  
}
```

Cette version minimale du constructeur par défaut initialise les attributs avec les valeurs par défaut 0, 0.0, et false pour les types de base et null pour les objets.

N.B : Dès qu'au moins un constructeur a été spécifié, ce constructeur par défaut par défaut n'est plus fourni.

Problème : les constructeurs initialisent seulement

21

```
public class Pixel {
    private int x;
    private int y;
    public Pixel(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public boolean sameAs(Pixel p) {
        return (this.x==p.x) && (this.y==p.y);
    }
    public static void main(String[] args) {
        Pixel p0 = new Pixel(0,0);
        Pixel p1 = new Pixel(1,3);
        boolean b = p0.sameAs(p1); // false
    }
}

class OtherClass {
    public static void main(String[] args) {
        Pixel p0 = new Pixel(0,0);
        p0.sameAs(p); // true
        p0.x = 1; // error: x has private
                // access in Pixel
    }
}
```

On arrive
pas à lire
ou
modifier
les
attributs

La solution : Les accesseurs et mutateurs (Getters and setters)

Les accesseurs (getters) et mutateurs (setters)

22

- ❑ Un accesseur (getters) est une méthode permettant de récupérer le contenu d'une donnée membre protégée.
- ❑ Un mutateur (setters) est une méthode permettant de modifier le contenu d'une donnée membre protégée.

```
public class Point {  
    private double x;  
    private double y;  
  
    public Point(double x, double y) {  
        this.x=x;  
        this.y=y;  
    }  
    public double getX() {  
        return x;  
    }  
    public void setX(double x) {  
        this.x=x;  
    }  
    public double getY() {  
        return y;  
    }  
    public void setY(double y) {  
        this.y=y;  
    }  
}
```

Classification des méthodes d'un objet :

- ❑ **Constructeur** : Initialise l'objet.
- ❑ **Accesseur ou Getter** : Exporte les données (souvent partiellement).
- ❑ **Mutateur ou Setter** : Importe les données (en les vérifiant).
- ❑ **Méthode métier** (business method) : Effectue des traitement en fonction des données.

L'encapsulation :

Pour respecter le principe d'encapsulation, les attributs sont privés et les méthodes sont publiques. Les méthodes nécessaires au fonctionnement interne des classes sont elles-aussi privées.

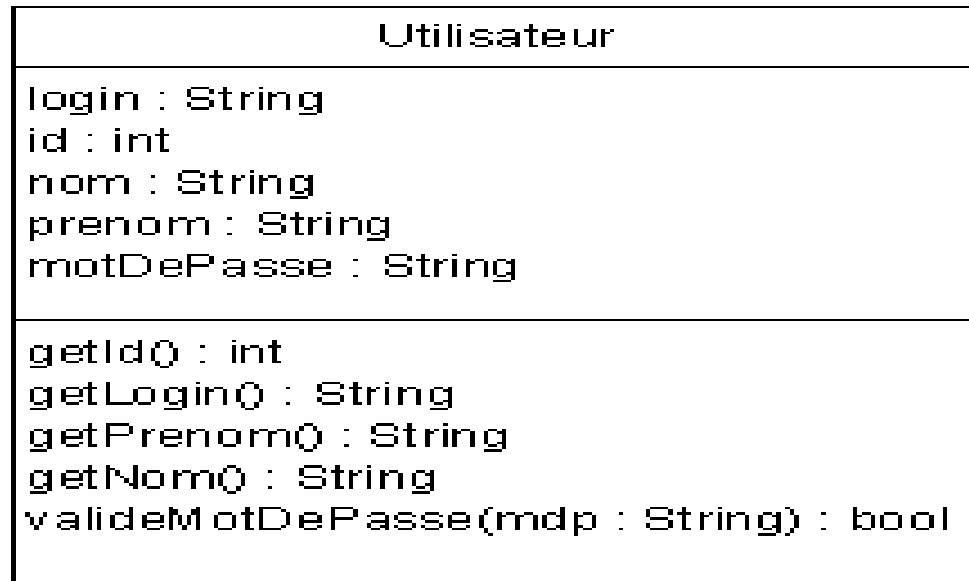
Intérêt de L'encapsulation :

- ❑ Simplification de l'utilisation des objets,
- ❑ Meilleure robustesse du programme,
- ❑ Simplification de la maintenance globale de l'application

Exercice

24

Traduire la représentation UML de la classe **Utilisateur** en code Java.



Le mot clé this

25

- ❑ Le mot-clé **this** désigne en permanence l'objet dans lequel on se trouve c.à.d. qu'il correspond à la référence de l'objet sur lequel la méthode courante a été appelée.
- ❑ Il peut être utilisé pour rendre le code explicite et non ambigu.
- ❑ Par exemple, si dans une méthode, on a un paramètre ayant le même nom qu'un attribut de la classe dont la méthode fait partie, on peut désigner explicitement l'attribut grâce à **this**.

Le mot clé this

26

```
public class Pixel {  
    private int x;  
    private int y;  
    public Pixel(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public Pixel() {  
        this(0,0);  
    }  
    public Pixel(int v) {  
        this(v, v);  
    }  
}
```

Le mot clé this

27

```
class Main {  
    public void toto() {  
        Matrix m1=...  
        Matrix m2=...  
        if (m1.equals(m2)) {  
            ...  
        }  
    }  
}
```

```
class Matrix {  
    boolean equals(Matrix m) {  
        if (this==m)  
            return true;  
        ...  
    }  
}
```

Ici, **this** aura la valeur de m1 et m la valeur de m2

Le mot clé static

28

- ❑ Static est un modificateur en Java qui s'applique aux éléments suivants:
 - ❑ les variables
 - ❑ les méthodes
- ❑ Pour créer un membre statique (variable, méthode), précédez sa déclaration avec le mot-clé static.
- ❑ Lorsqu'un membre est déclaré statique, il est accessible avant la création des objets de sa classe et sans référence à aucun objet.

Static variable

29

- ❑ Lorsqu'une variable est déclarée comme statique, une seule copie de variable est créée et partagée entre tous les objets au niveau de la classe.
- ❑ Les variables statiques sont essentiellement des variables globales.
- ❑ Toutes les instances de la classe partagent la même variable statique.

Exemple

30

Comparez les deux implémentations suivantes de la classe Chien :

```
public class Chien {  
    private int nbChien;  
  
    public Chien() {  
        nbChien++;  
    }  
}
```

```
public class Chien {  
    private static int nbChien;  
  
    public Chien() {  
        nbChien++;  
    }  
}
```

Static method

31

- ❑ Bien que Java soit un langage objet, il existe des cas où une instance de classe est inutile. Le mot clé static permet alors à une méthode de s'exécuter sans avoir à instancier la classe qui la contient.
- ❑ L'appel à une méthode statique se fait alors en utilisant le nom de la classe, plutôt que le nom de l'objet.
- ❑ Une méthode statique ne peut pas utiliser des variables d'instance non statiques.
- ❑ En général, une classe possédant des méthodes statiques n'est pas conçue pour être instanciée.
- ❑ Une méthode statique ne peut pas utiliser de méthodes non statiques.

Static method : Example

32

```
public class MaClassMath {  
    ...  
    public static int min( int a, int b) {  
        // retourne la plus petite valeur  
    }  
}  
  
public TestMaClassMath {  
    public static void main( String []args){  
        int x = MaClassMath.min(21,4);  
    }  
}
```


Quand utiliser des variables et des méthodes statiques?

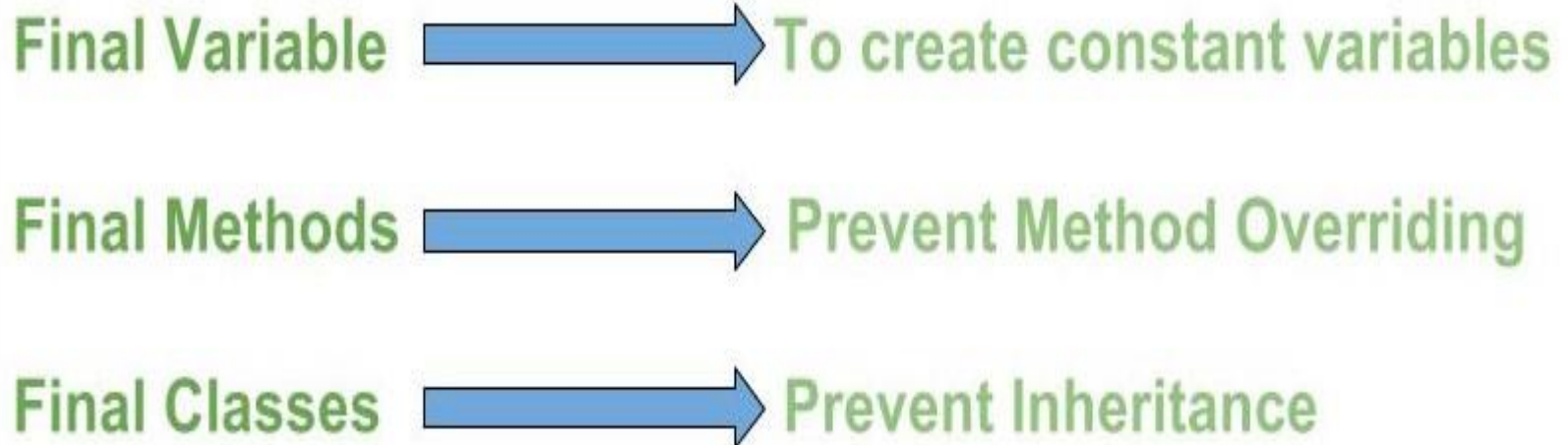
33

- ❑ Utilisez la variable statique pour la propriété commune à tous les objets. Par exemple, dans la classe Étudiant, tous les étudiants partagent le même nom de collège.
- ❑ Utilisez des méthodes statiques lorsque le comportement de ces méthodes ne dépend pas de la valeur des variables d'instance.

Le mot clé final

34

- **final** est un modificateur applicable uniquement à une **variable** (attribut), une **méthode** ou une **classe**.




final variable (attribut)

35

- ❑ Une variable déclarée finale ne change plus de valeur une fois initialisée au moment de sa déclaration ou au moment de l'appel d'un constructeur.
- ❑ Elle ne pourra donc plus être modifiée par le programme (toute tentative de modification produira un message d'erreur lors de la compilation).

Exemples :

```
1 package sct;  
2  
3 public class Person {  
4     private String name="Amit";  
5     private final int age= 30 ;  
6  
7     public void finalDemo(){  
8         name = "Amit Himani";  
9         age=35;  
10    }  
11 }  
12  
13
```



→ Problème : constante pour chaque objet.

Solution : public static final

36

- ❑ une variable (attribut) **statique et finale** est une constante de la classe.
- ❑ Par convention, elle est notée en majuscule, un blanc souligné séparant les mots.
- ❑ Pour rendre la constante d'une classe accessible à tout le monde, il faut la déclarer avec le modificateur d'accès **public**.

```
class MesConstantes {  
    public static final double PI_APPROX = 3.1415;  
}
```

```
// ailleurs dans le programme  
int i = 2 * MesConstantes.PI_APPROX;
```

final method

37

- ❑ Lorsqu'une méthode est déclarée avec le mot clé final, elle est appelée méthode finale.
- ❑ Une méthode finale ne peut pas être redéfinie (overriding).
- ❑ Nous devons déclarer les méthodes avec le mot-clé final pour lesquelles nous devons suivre la même implémentation dans toutes les classes dérivées.

final method : example

38

```
29 class UNIX {
30
31     protected final void whoAmI () {
32         System.out.println("I am UNIX");
33     }
34
35 }
36
37 class Linux extends UNIX {
38
39     public void whoAmI () {
40         System.out.println("I am Linux");
41     }
42 }
```

Cannot override the final method from UNIX

1 quick fix available:

[Remove 'final' modifier of 'UNIX.whoAmI'\(..\)](#)

Press 'F2' for focus

final class


39

- ❑ Lorsqu'une classe est déclarée avec le mot clé final, elle est appelée classe finale.
- ❑ Une classe finale ne peut pas être étendue (héritée).
- ❑ Il y a deux utilisations d'une classe finale:
 - ❑ L'une est certainement d'empêcher l'héritage, car les classes finales ne peuvent pas être étendues.
 - ❑ L'autre utilisation de final avec des classes est de créer une classe immuable comme la classe String prédéfinie. Vous ne pouvez pas rendre une classe immuable sans la rendre finale.


final class : example

40

```
1 package com.javaguides.corejava.keywords.finalkeyword;
2
3 final class Person {
4     private String firstName;
5     private String lastName;
6     public String getFirstName() {
7         return firstName;
8     }
9     public void setFirstName(String firstName) {
10         this.firstName = firstName;
11     }
12     public String getLastName() {
13         return lastName;
14     }
15     public void setLastName(String lastName) {
16         this.lastName = lastName;
17     }
18 }
19
20 class Employee extends Person {
21     |
22 }
```

 The type Employee cannot subclass the final class Person

1 quick fix available:

 [Remove 'final' modifier of 'Person'](#)

Press 'F2' for focus

Simple immutable class : example

41

```
1 package com.programmer.gate.beans;
2
3 public final class ImmutableStudent {
4
5     private final int id;
6     private final String name;
7
8     public ImmutableStudent(int id, String name) {
9         this.name = name;
10        this.id = id;
11    }
12
13    public int getId() {
14        return id;
15    }
16
17    public String getName() {
18        return name;
19    }
20 }
```

N.B : - Tous les attributs d'une classe immuable doivent être déclarés **final**.
- Eviter les setters.

Mais ça reste un simple exemple d'une classe immuable !



Merci de votre attention