POO 2024

En Java

A.BRIGHEN Univ Jijel
poojijel2020@gmail.com

2 Bases de POO (suite)

Les classes et les objets

Bases de POO

Les classes et les objets

Nous avons vu précédemment que dans la programmation **procédurale**, les **données** et les **traitement** apparaissent comme des **entités séparées**.

Par exemple, si on veut créer un programme qui permet de calculer la **surface d'un rectangle**, on doit déclarer deux variable **hauteur** et **largeur**, puis définir une **fonction surface** avec **deux paramètres** qui désignent la largeur et la hauteur. A la fin la fonction **renvoie** comme valeur la surface du triangle

3

Bases de POO

Les classes et les objets

```
1
2  package exemples;
3  public class Exemples {
4
5  public static void main(String[] args) {
6     double hauteur=4;
7     double largeur=3.5;
8     System.out.println("La surface= "+ surface(hauteur, largeur));
9      }
10    static double surface(double h, double l)
11     {
12     return l*h;
13     }
14 }
```

Dans ce cas les données, qui sont stockées dans des variables, sont **séparées des traitements** (la fonction surface) et n'ont aucun lien sémantique.

4

Bases de POO

Les classes et les objets

Dans la **POO**, **les données et les traitements** sont **regroupées** en une seule et même entité appelé "**objet**" qui est instancié d'une **classe**.

La programmation orientée objet (POO) est basée sur ces deux concepts : **Classe et Objet**

5

Bases de POO

Les classes et les objets

Un **objet** est une chose, à la fois tangible et intangible, qu'on peut imaginer. Chaque objet a son **propre état**, **comportement** et **identité**.

Exemples:

- pour un programme permettant de suivre les étudiants résidents dans une cité universitaire, on peut avoir de nombreux objets ***Student***, ***Room*** et ***Floor***.
- Pour un autre programme permettant de suivre les clients et l'inventaire d'un magasin de vélos, on peut avoir ***Client***, ***Vélo*** et de nombreux autres types d'objets ,

6

Bases de POO

Les classes et les objets

Un **objet** est une chose, à la fois tangible et intangible, qu'on peut imaginer. Chaque objet a son **propre état**, **comportement** et **identité**.

Un objet est composé de **données** et **d'opérations** qui manipulent ces données

Exemple:

- un objet ***Étudiant*** peut avoir des **données** telles que le nom, le sexe, la date de naissance, l'adresse, le numéro de téléphone et l'âge, ainsi que des **opérations** permettant d'affecter et de modifier ces valeurs de données.

7

Bases de POO

Les classes et les objets

R Dans un même programme on peut avoir de nombreux objets du même type. Par exemple, dans le programme de magasin de vélos, on s'attend à voir de nombreux vélos et autres objets.

R Un objet doit être instanciée d'une classe. Donc, une classe doit être définie avant la création d'une instance (objet) de la classe.

- Une **classe** est une sorte de forme ou de modèle qui décrit et impose ce que **les objets** peuvent et ne peuvent pas faire.

En d'autres termes:

- Une ***classe*** est **un nouveau type de données** dont les instances sont des **objets**.

8

Bases de POO

Les classes et les objets

- Un objet est appelé une **instance** d'une **classe**.
- Un objet est une **instance** d'exactement **d'une seule classe**.
- Une **classe** se compose de **variables** d'instance et de **méthodes**.
- Pour déclarer une nouvelle classe il suffit d'utiliser le mot clé ***"class"*** suivi du **nom de la classe** :

```
1 class MyClasse {
2 ...
3 }
```

9

Bases de POO

Les classes et les objets

- Une fois une nouvelle classe est définit, supposons que le nom de la classe soit ***MyClass***, il est possible de créer des **objets de ce type**
- Ces objets sont des **instances** de la classe ***MyClass***.
- Chaque **objet** créé possède tous les **attributs** de la classe dont il est issu.

```
1 MyClass obja=new MyClass();
```

```
1 MyClass obja;//Line 1
2 obja=new MyClass();//Line 2
```

A la fin de la première ligne, **obja** est une **référence**. Jusqu'à ce point, il n'y a pas d'allocation mémoire. Mais une fois le mot ***new*** s'exécute (Line2), la mémoire est allouée.

10

Bases de POO

Les classes et les objets

R Dans la deuxième ligne, le nom de la classe est suivi des parenthèses. Elles servent pour les constructeurs (constructors). Les constructeurs sont utilisés pour décrire ce qui se passe lorsqu'un objet est créé. Les attributs de l'objet prennent des valeurs initiales données par un constructeur. Pour chaque classe, il existe un constructeur par défaut qui initialise les attributs à des valeurs initiales par défaut.

```
1 MyClass obja=new MyClass();
```

```
1 MyClass obja;//Line 1
2 obja=new MyClass();//Line 2
```

11

Bases de POO

Les classes et les objets

```
16 // Definition of a new class Rectangle
17 class Rectangle{
18     double hauteur;
19     double largeur;
20
21     double surface()
22     {   double s=hauteur*largeur;
23         return s;
24     }
25 }
```

12

Bases de POO

Les classes et les objets

```
1
2 package exemples;
3 public class Exemples {
4
5 public static void main(String[] args) {
6     //Create new object r of class Rectangle
7     Rectangle r=new Rectangle();
8     r.hauteur=4.3;
9     r.largeur=5.2;
10    System.out.println(r.hauteur);
11    System.out.println(r.largeur);
12    System.out.println(r.surface());
13 }
14 }
15
```

13

Bases de POO

Les classes et les objets

R La variable qui désigne l'objet créé est une référence à l'objet, et non pas l'objet lui-même.Supposant la création des deux objets $r1$ et $r2$ de types Rectangle. L'affectation $r1 \leftarrow r2$ affecte à $r1$ la référence à l'objet désigné par r2 (voir le code suivant). Après l'affectation, les références $r1$ et $r2$ désignent le même objet.

```
1 Rectangle r1=new Rectangle();//Create r1
2 Rectangle r2=new Rectangle();//Create r2
3 //r1 and r2 are tow different objects
4 r1=r2;//Here, r1 and r2 designate the same object
```

14

Bases de POO

Les classes et les objets

```java
package newclasse;

public class NewClasse {

    public static void main(String[] args) {
    Person Alice=new Person();//create new object Alice of Person
        Alice.name="Alice Dunod";
        Alice.adress="Madrid";
        Alice.age=23;
        Alice.Afficher();
    }
}

class Person
{
    String name;
    String adress;
    int age;
void Afficher()
{
System.out.println(name+" "+age+" ans habit a "+adress);
}

}
```

15

Bases de POO

Les classes et les objets: les attributs

- Les données stockées dans un objet **représentent l'état de l'objet**. Dans la de la POO, ces données sont appelées ***attributs***
- Les **attributs** contiennent les informations qui permettent de **différencier** un objet d'un autre
- Pour déclarer des **attributs** d'une classe, on utilise la syntaxe suivant :

```
class My_class {
type_attribut1 name_attribut1;
type_attribut2 name_attribut2;
type_attribut3 name_attribut3;
...
}
```

16

Bases de POO

Les classes et les objets: les attributs

- **L'accès** aux **valeur des attributs** d'une instance se fait comme suit :

```
object_name.name_attribut; //Access to the attribute value
```

```
Alice.name     //Access to the attribute name's value
Alice.adress //Access to the attribute adress's value
Alice.age    //Access to the attribute age's value
```

```
r.hauteur     //Access to the attribute "hauteur" value
r.largeur //Access to the attribute "largeur" value
```

17

Bases de POO

Les classes et les objets: les attributs

```
package classes.examples;
class ClassA
{
        int i=5;
}
class ClassEx1
{
        public static void main(String args[])
        {
                System.out.println("*** A Simple class with 2 objects-obA And obB ***");
                ClassA obA=new ClassA();
                ClassA obB=new ClassA();
                System.out.println("obA.i ="+ obA.i);
                System.out.println("obB.i ="+ obB.i);

        }
}
```

18

Bases de POO

Les classes et les objets: les attributs

Une classe peut avoir deux types d'attributs: **les variables d'instance** et **les variables de classe**, également appelées respectivement, variables **non statiques** et **statiques**.

Les variables d'instance représentent l'état d'un objet de la classe. **Une copie** de toutes les variables d'instance existe pour **chaque objet** de la classe.

19

Bases de POO

Les classes et les objets: les attributs

Les variables de classe (**static variable**) représentent l'état de la classe elle-même. **Une seule copie** des variables de classe existe pour une classe.

Si une variable est déclarée **statique (static)**, alors la valeur de la variable est la **même** pour toutes les instances, et nous n'avons pas besoin de créer un objet pour appeler cette variable

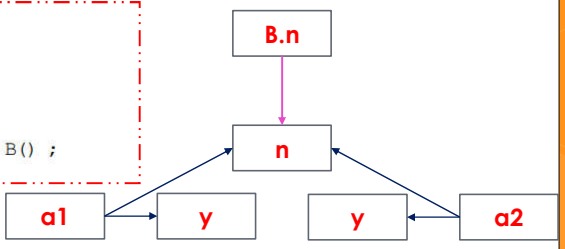
20

Bases de POO

Les classes et les objets: les attributs

```
class B
{ static int n ;
  float y ;
}

B a1 = new B(), a2 = new B() ;
```




```
a1.y=10; a2.Y=12; a1.n=3; a2.n=5;
System.out.println(a1.y + " " +a2.y+" "+ a1.n + " " +a2.n);
//10 12 5 5
a1.y++; a2.Y++; a2.n++;
System.out.println(a1.y + " " +a2.y+" "+ a1.n + " " +a2.n);
//11 13 6 6
B.n=7;
System.out.println(a1.y + " " +a2.y+" "+ a1.n + " " +a2.n);
//11 13 7 7
```

21

Bases de POO

Les classes et les objets: les Méthodes

Une classe peut avoir deux types de méthodes :

- **les méthodes d'instance (objet) et**
- **les méthodes de classe**.

Les méthodes d'instance et les méthodes de classe sont également appelées, respectivement, méthodes **non statiques** et méthodes **statiques**.

Une **méthode d'instance** est utilisée pour implémenter le comportement des instances (objets) de la classe.

22

Bases de POO

Les classes et les objets: les Méthodes

Une méthode d'instance ne peut être invoquée que dans le contexte d'une instance de la classe.

La syntaxe pour invoquer une méthode d'instance est la suivante :

```
objectname.instanceMethodName(actual-parameters);

// Exemple:

String st=new String("Hello");
st.length();//length() est une méthode d'instance
```

23

Bases de POO

Les classes et les objets: les Méthodes

- Une **méthode de classe** ou une **méthode *statique*** est une méthode qui n'agit pas sur des variables **d'instance** mais uniquement sur des variables de classe.

Une méthode de classe :

☆ peut être appelée même sans avoir instancié la classe

☆ peut accéder uniquement à des variables et méthodes **statiques**

☆ dans la déclaration, le nom de la méthode est précédé du mot clé ***static***

☆ peut être appelée avec la notation ***classe.methode()*** au lieu de ***objet.methode()***

24

Bases de POO

Les classes et les objets: les Méthodes

```java
// Exemples des méthodes de classes:

int e = Math.abs(j);
boolean m=Character.isLowercase('Z');

int tab[]={12,8,16,10,14};
System.out.println(max(tab));
```

25

Bases de POO

Les classes et les objets: les Méthodes

L'absence du mot ***static*** dans la déclaration d'une méthode rendre de la méthode une méthode d'instance (**non statique**)

```java
// A static or class method
static void aClassMethod() {

// Method's body goes here

}

// A non-static or instance method
void anInstanceMethod() {

// Method's body goes here

}
```

26

Bases de POO

Les classes et les objets: les Méthodes

	Méthode de classe	Méthode d'instance
Déclaration	Mot clé **static**	Absence du Mot clé static
Appelle (syntaxe)	ClassName.MethodeN(args);	ObjectName.MethodN (args);
Accès aux attributs	**Attributs statiques**	Tous les attributs
Appelle (condition)	Peut être appeler sans avoir instancié la classe	Un objet doit être instancié avant l'appelle de la méthode

- Si une variable (attribut) est déclarée **statique**, alors la valeur de la variable est la **même** pour toutes les instances, et nous n'avons pas besoin de créer un objet pour appeler cette variable.

27

Bases de POO

Les classes et les objets: Les constructeurs

R Dans la deuxième ligne, le nom de la classe est suivi des parenthèses. Elles servent pour les constructeurs (constructors). Les constructeurs sont utilisés pour décrire ce qui se passe lorsqu'un objet est créé. Les attributs de l'objet prennent des valeurs initiales données par un constructeur. Pour chaque classe, il existe un constructeur par défaut qui initialise les attributs à des valeurs initiales par défaut.

```
1 MyClass obja=new MyClass();
```

```
1 MyClass obja;//Line 1
2 obja=new MyClass();//Line 2
```

28

Bases de POO

Les classes et les objets: Les constructeurs

☆ la notion de **constructeur** permet **d'automatiser** le mécanisme **d'initialisation** d'un objet.

☆ cette initialisation ne sera pas limitée à la mise en place de valeurs initiales; il pourra s'agir de **n'importe quelles actions utiles** au bon fonctionnement de l'objet.

☆ Un constructeur n'est rien d'autre qu'une **méthode**, **sans valeur de retour**, portant le **même nom que la classe**.

☆ Il peut disposer d'un nombre quelconque **d'arguments** (éventuellement aucun)

29

Bases de POO

Les classes et les objets: Les constructeurs

☆ Le nom du constructeur doit toujours être le même que celui de la classe.

☆ Il ne doit pas y avoir de **type de retour** du constructeur.

☆ Un constructeur peut avoir **plusieurs arguments** ou **zéro arguments**

☆ Un constructeur est toujours invoqué lorsqu'un objet est instancié

☆ Un constructeur ne doit pas déclarer avec les mots clés **abstract**, **static** et **final**

30

Bases de POO

Les classes et les objets: Les constructeurs

☆ Lorsque aucun constructeur explicite n'est définit, le compilateur crée un **constructeur par défaut**

☆ Lorsque un constructeur **explicite paramétré** est définit, l'utilisation d'un constructeur sans argument génère une **erreur de compilation**.

☆ Dans ce ca, un constructeur sans argument doit être crée **manuellement**, avant de l'invoquer.

31

Bases de POO

Les classes et les objets: Les constructeurs

```java
// Exemples des  constructeur:

class A {
    int a;
    String b;

    A(int x, String s) {
        a = x;
        b = s;
    }
}
```

```java
// Exemples des  constructeur:

public class Main{
    public static void main(String[] args) {
        A a = new A(1, "One");
        System.out.println(a.a + " " + a.b);
//      A a1= new A(); ERROR
    }
}
```

32

Bases de POO

Les classes et les objets: Les constructeurs

```java
// Exemples des  constructeurs:

class A {
    int a;
    String b;

    A(int x, String s) {
        a = x;
        b = s;
    }

    A(){
        a = 0;
        b = " ";
    }
}
```

```java
// Exemples des  constructeurs:

public class Main{
    public static void main(String[] args) {
        A a = new A(1, "One");
        System.out.println(a.a + " " + a.b);

        A c= new A();
       System.out.println(c.a + " " + c.b);
    }
}
```

33

Bases de POO

Les classes et les objets: Les constructeurs

☆ Une classe peut avoir **plusieurs constructeurs** (des constructeurs surchargés)

☆ La surcharge des constructeurs est effectuée pour initialiser les attributs d'une classe de **différentes manières**.

☆ Les constructeurs surchargés doivent différer par le nombre de paramètres ou le type de données des paramètres qui leur sont transmis.

☆ **La signature** de chaque constructeur doit être différente des autres

34

Bases de POO

Les classes et les objets: les Méthodes

	Méthode	Constructeur
Nom	Nom de choix	Même nom de la classe
Type de retour	N'importe quel type +void	Aucun
arguments	+ieurs ou zéro	+ieurs ou zéro
Signature	Nom, type, ordre et nombre de paramètres	type, ordre et nombre de paramètres
Mot Static	Oui possible	Non
Nivaux d'accès	Public, private, protected	Public, private, protected

35

Bases de POO

Les classes et les objets: Exercices

```
Définissez une classe Rectangle, pour représenter des rectangles.

1. Un objet Rectangle aura deux attributs, une hauteur et une
    largeur.

2. Vous definirez un constructeur par défaut qui initialisera les
    deux attributs á zéro, ainsi qu un constructeur qui
    initialisera un rectangle á partir de deux parametres réels.

3. vous définirez une méthode qui calcul la surface d un rectangle

4. ajouter une méthode affiche, qui affiche les propriétés et la
    surface d un rectangle de la forme
la hauteur du rectangle est ...
la largeur du rectangle est ...
la surface du rectangle est ...

5. Vous ajouter une méthode statique, isEqual, qui compare deux
    rectangles. La méthodes isEqual renvoie true si les  des deux
    rectangles ont la meme surface, false sinon

6. Dans une classe de teste, utiliser la classe Rectangle pour cré
    er deux objets Rectangles,  les afficher  et les comparer (
    isEqual).
```

36

Bases de POO

Les classes et les objets: Exercices

Définissez une classe Complexe, pour représenter les nombres de l ensemble C.

1. Un objet Complexe aura deux attributs, une partie réelle et une partie imaginaire.

2. Vous definirez un constructeur par défaut qui initialisera les deux attributs á zéro, ainsi qu un constructeur qui initialisera un nombre Complexe á partir de deux parametres ré els.

3. Vous écrirez la méthode toString qui donne une représentation d un nombre Complexe sous la forme (r,i).

4. vous définirez une méthode qui affiche un nombre complexe de la forme <a i+ b>

5. Vous ajouter une méthode statique qui réalise la somme de deux nombres complexes

6. Dans une classe de teste, tester la classe Complexe

37

Bases de POO

Les classes et les objets: Niveaux de Visibilité

selon leur **niveau de visibilité** (**accessibilité**), les classes, les attributs et les méthodes peuvent être **accessibles** ou non **accessibles** depuis des classes du même paquetage ou d'autres paquetages.

Pour définir les niveaux de visibilité, on utilise les **modificateurs** de visibilité ***private***, ***protected*** et ***public***. Ces modificateurs sont placés devant l'en-tête d'une classe, d'une méthode ou devant le type d'un attribut

Si **aucun modificateur** n'est signalé, il s'agit de la **visibilité par défaut** qui est la **visibilité package**, c'est-à-dire que la classe (l'attribut ou la méthode) est visible dans le package où elle est définie

38

Bases de POO

Les classes et les objets: Niveaux de Visibilité

Afin d'utiliser une classe (déclarée *public*) dans un **package externe**, il faut l'importer. Pour cala, il suffit d'inclure en début de fichier le chemin d'accès à la classe qu'on veut importer en utilisant l'instruction **import**

```
1
2 import nomPackage.nomClasse ;
3
4 import java.util.Random ;
5
6 import java.util.Scanner;
```

39

Bases de POO

Les classes et les objets: Niveaux de Visibilité

Une classe possède par défaut la visibilité **package**. Si le modificateur **public** n'est pas indiqué, elle est **accessible par toute classe** qui est défini dans le **même package** et elle n'est pas accessible dans les **packages externes** (les autres).

On peut attribuer à une classe le modificateur **public** qui la rend accessible de n'importe quel autre package (package externe), c'est le **plus haut niveau de visibilité**.

40

Bases de POO

Les classes et les objets: Niveaux de Visibilité

Une seule classe **publique** doit être définit par fichier. Cette classe publique doit avoir exactement le **même nom** que celui du **fichier**

Un fichier Java peut contenir plus qu'une seule définition de classe. Mais, outre la classe **publique**, les autres seront considérés comme des classes restreintes au package dans lequel est placé le fichier. Elles ne devront porter aucun **qualificateur de visibilité**

41

Bases de POO

Les classes et les objets: Niveaux de Visibilité

```
1
2   package visibilitetest;
3
4   public class VisibiliteTest {
5
6   }
7
8   class A{
9       // Corps
10  }
11
12  public class B{
13      // Corps de la classe B
14  }
15
16  private class C{
17      // Corps de la classe C
18  }
19
20  class D{
21      // Corps de la classe D
22  }
23
```

class B is public, should be declared in a file named B.java

(Alt-Enter shows hints)

42

Bases de POO

Les classes et les objets: Niveaux de Visibilité

Une **classe interne** est une classe définie à l'intérieur d'une autre classe. Si une classe est définie sous forme de **membre d'une autre classe**, on peut utiliser un des niveaux de visibilité classiques : **public**, **protected**, **private**.

Un **membre** (classes, attributs, méthodes) d'une classe C peut être :

private : accessible seulement à la classe ;

par défaut (package friendly): valeur par défaut, accessible aux classes dans le même paquetage ;

protected : accessible aux classes dans le même paquetage et à toute classe **dérivée** en dehors du paquetage;

public : accessible à toutes les classes.

43

Bases de POO

Les classes et les objets: Niveaux de Visibilité

```java
package p1;

public class C {
  public int a;
  int b;
  private int d;

  int s(){
      return a+b;
  }

  public int m(){
      return a*b*d;
  }

}
class D {
    int f;
    private int k;
}
```

```java
package p1;

public class P1 {
  // code
}
```

```java
package p2;
import p1.*;
public class P2 {
    // code ..
}
```

Visibilité	Classes & Membres de classes								
↓	C	D	a	b	s	m	f	k	d
C	√							X	
D	√							√	
P1	√							X	
P2	√							X	

44

Bases de POO

Les classes et les objets: Niveaux de Visibilité

R En pratique, il est recommandé de définir :

- ☆ Toutes les classes publiques afin d'augmenter la réutilisation
- ☆ Toutes les méthodes publiques (ou protégées) afin de pouvoir les invoquer dans des packages externes
- ☆ Tous les attributs privés (ou protégés) afin de garder le maximum de contrôle sur leur manipulation

45

Bases de POO

Les classes et les objets: Les accesseurs

Pour accéder aux **attributs *privés*** à l'extérieur de leur classe, on doit fournir des méthodes spécifiques appelées ***méthodes d'accès*** ou ***accesseurs*** **:** les méthodes ***get et set***. .

La méthode **get** permet de **renvoyer** la valeur de l'attribut

La méthode **set** permet de **modifier** la valeur de l'attribut

parfois, on fournit juste une méthode ***get*** pour un accès en lecture seulement

parfois, un **attribut privé** peut être ***Caché*** (n'est pas accessible ni en en lecture ni en écriture, aucune méthode)

46

Bases de POO

Les classes et les objets: Les accesseurs

```java
package rectangle;
public class Rectangle{
    private float h;
    private float l;

public Rectangle(){//Co
    h=0;
    l=0;
    }
public Rectangle(float a,float b){//Constructeur 2
    h=a;
    l=b;
    }

// Accésseurs
public float getH(){return h;}   //getter
public float getL(){return l;}  //getter
public void  setH(float a){h=a;}  //setter
public void  setL(float a){l=a;} //setter
}
```

```java
public static void main(String[] args) {
    Rectangle r=new Rectangle(12,16);
    r.setH(10);
    r.setL(14);
    System.out.println(r.getH()+" "+r.getL());
}
```

47

Bases de POO

Les classes et les objets: Le mot clés this

```java
package a;
public class A {
    int x;
    float y;

A(int x, float y){
        x=x;
        y=y;
    }

void Afficher(){
        int x=12;
        float y=3;
        System.out.println("x= "+x+"y= "+y);
    }
}
```

Pour éliminer **la confusion** entre les attributs de la classe et les paramètres des méthodes portant le même nom, on utilise le mot clés ***this***

48

Bases de POO

Les classes et les objets: Le mot clés this

```
1
2   package a;
3   public class A {
4       int x;
5       float y;
6
7   A(int x, float y){
8           this.x=x;
9           this.y=y;
10      }
11
12  void Afficher(){
            int x=12;
            float y=3;
15          System.out.println("x= "+this.x+"y= "+this.y);
16      }
17  }
```

Le mot-clé ***this*** est un mot-clé spécial utilisé en java pour instancier les **variables de classe**. Et pour référencer l'instance de l'objet courant de la classe

49

Bases de POO

Les classes et les objets: mot clé final

Le mot-clé ***final*** indique qu'un élément **ne peut être changé** dans la suite du programme.

Il peut s'appliquer aux :

- ☆ **méthodes** d'une classe
- ☆ **attributs** d'une classe
- ☆ à **la classe** elle-même.
- ☆ sur les **paramètres** d'une méthode
- ☆ sur les **variables locales** d'une méthode

50

Bases de POO

Les classes et les objets: mot clé final

Le mot-clé ***final*** Java permet de déclarer que **la valeur d'une variable** (attribut) ne doit pas être modifiée pendant l'exécution du programme

```
1  package a;
2  public class A {
3      int x;
4      float y;
5
6      final double pi=Math.PI;
7      final double Z;
8
9  A(int x, float y){
10     this.x=x;
11     this.y=y;
12
13     cannot assign a value to final variable pi
14     ----
15     (Alt-Enter shows hints)
16     pi=3.14;//ERROR
17     Z=x;
18 }
```

On est pas obligé **d'initialiser** une variable déclarée ***final*** lors de sa déclaration. En effet, Java demande simplement qu'une variable déclarée ***final*** ne **reçoive qu'une seule fois** une **valeur**

L'affectation doit être effectuée, au plus tard, dans le constructeur de la classe

51

Bases de POO

Les classes et les objets: mot clé final

Si la variable **finale** est un **objet**, c'est la **référence** vers l'objet qui devient **constante** et non sa valeur. Ceci s'applique également aux **tableaux** qui sont aussi des références.

```
1  package a;
2  public class A {
3      int x;
4      float y;
5      final double pi=Math.PI;
6
7      final double Z[]=new double[10];
8
9  A(int x, float y){
10     double[]k=new double[3];
11
12     cannot assign a value to final variable Z
13     ----
14     (Alt-Enter shows hints)
15     Z=k;
16     Z[0]=x;Z[1]=y;
17 }
```

52

Les classes et les objets: mot clé final

Une **méthode** indiquée comme final ne peut être **redéfinie** dans une classe **dérivée**.

```
public final float somme(){
    return x+y;
}
```

Les **classes final** ne peuvent être **dérivées (hérité)**.

```
package a;
public final class A {
```