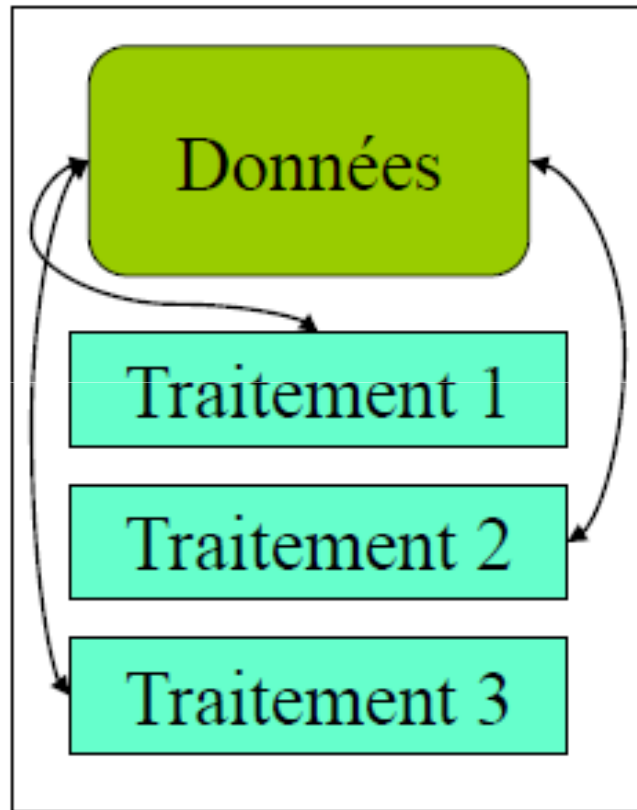


Introduction à la POO

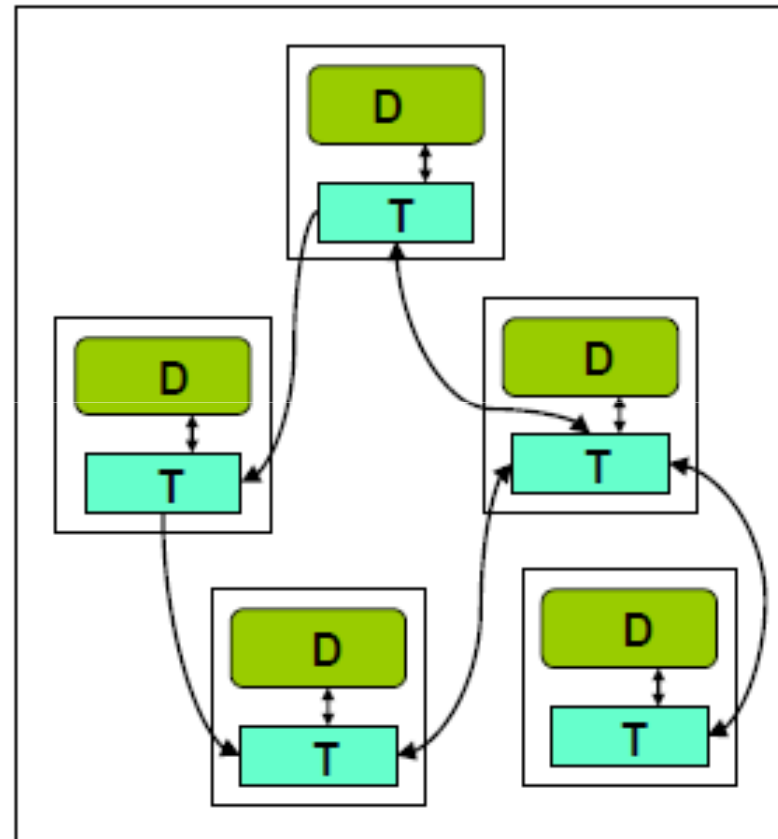
Langage JAVA

1

POO vs Programmation Procédurale



Procédurale



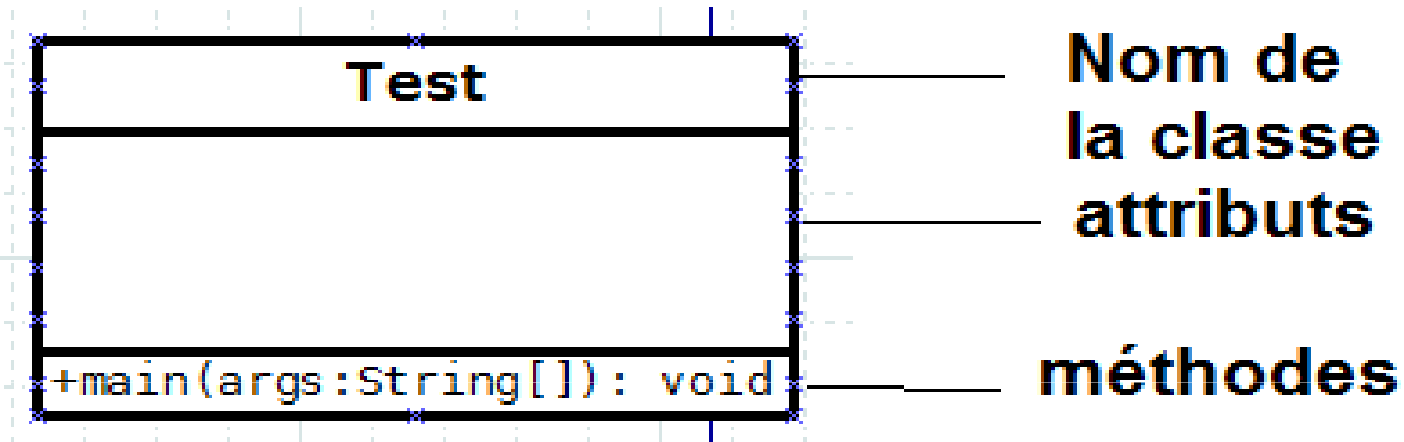
POO

Qu'est ce que JAVA

Le langage Java a été introduit par la société SUN en 1995. Il possède de nombreuses caractéristiques :

- ❑ C'est un langage orienté objet
- ❑ C'est un langage compilé : avant d'être exécuté, il doit être traduit dans le langage de la machine sur laquelle il doit fonctionner
- ❑ Il emprunte sa syntaxe en grande partie du langage C
- ❑ Les programmes Java peuvent être exécutés sous forme **d'applications indépendantes** ou **distribuées** à travers le réseau et exécutées par un navigateur Internet sous forme **d'applets ou servelets**.

Structure d'un programme simple en JAVA

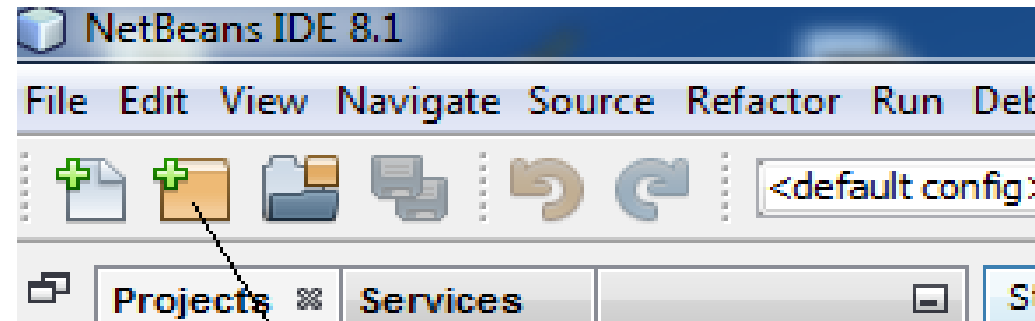


```
public class Test{  
    public static void main(String[] args)  
    { // ici le programme principal  
    }  
}
```

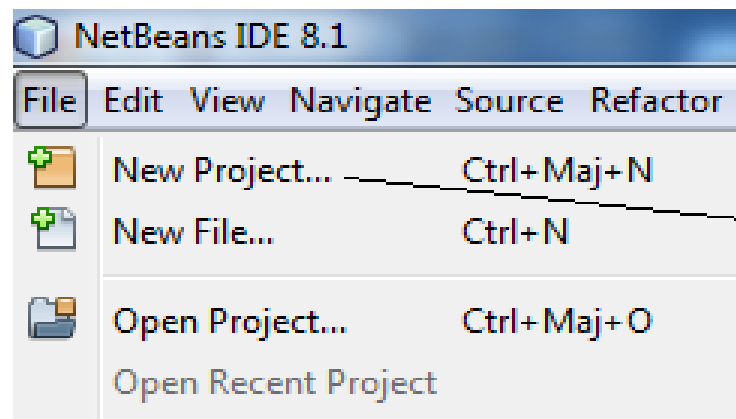
Illustration de la structure d'un programme en JAVA

- ☐ En Java, un projet est divisé en groupes appelés Paquetage (package).
- ☐ Un paquetage peut contenir plusieurs classes
- ☐ Le mot public au niveau d'une classe signifie qu'elle peut être exploitée localement dans son paquetage ou par d'autres paquetage de son projet.
- ☐ Le mot public au niveau de la méthode main signifie qu'elle est accessible depuis l'extérieur de sa classe.

Créer un nouveau Projet sous IDE NetBeans

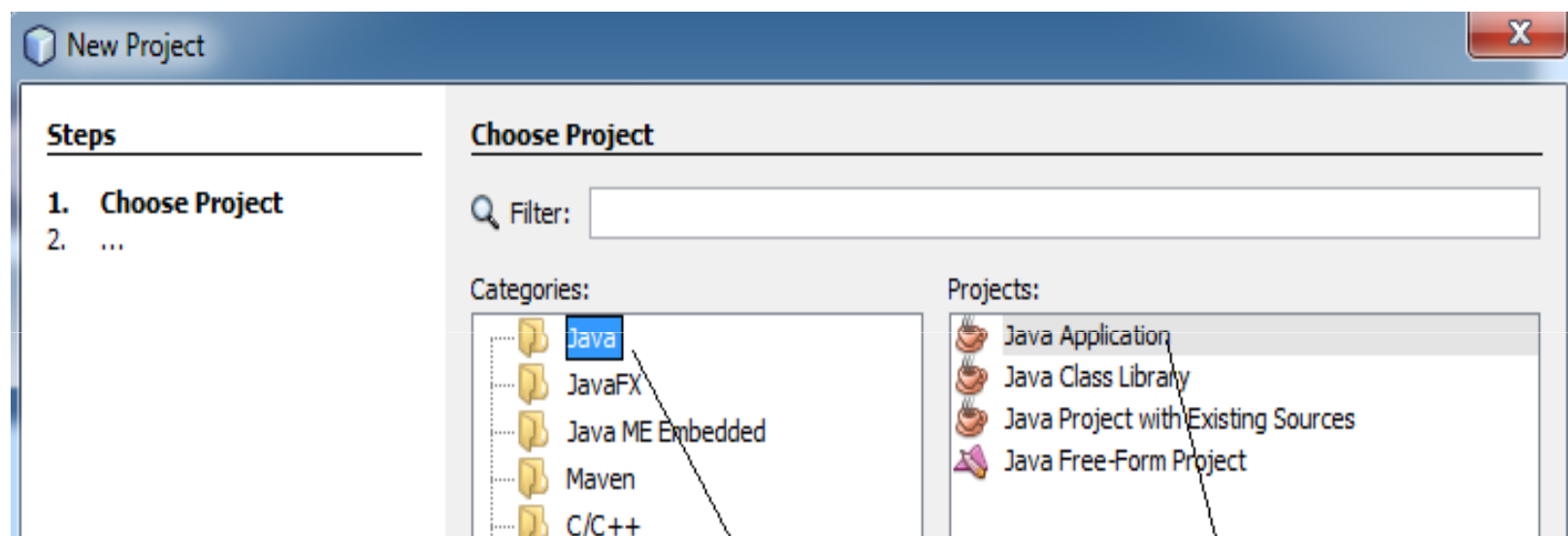


un clic ici



via menu File

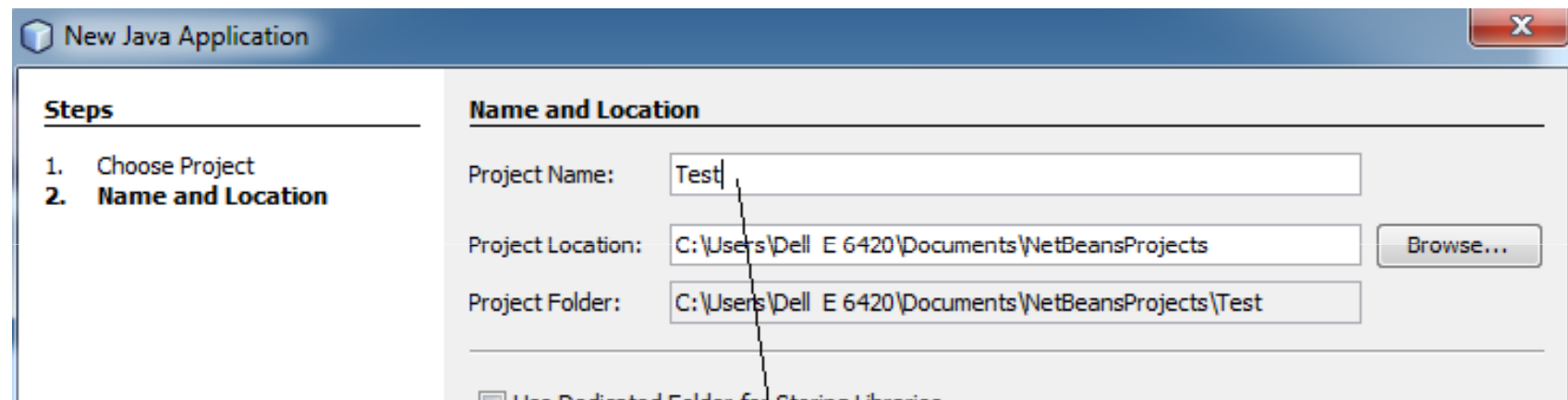
Créer un nouveau Projet sous IDE NetBeans –suite-



catégorie de
projet: Java

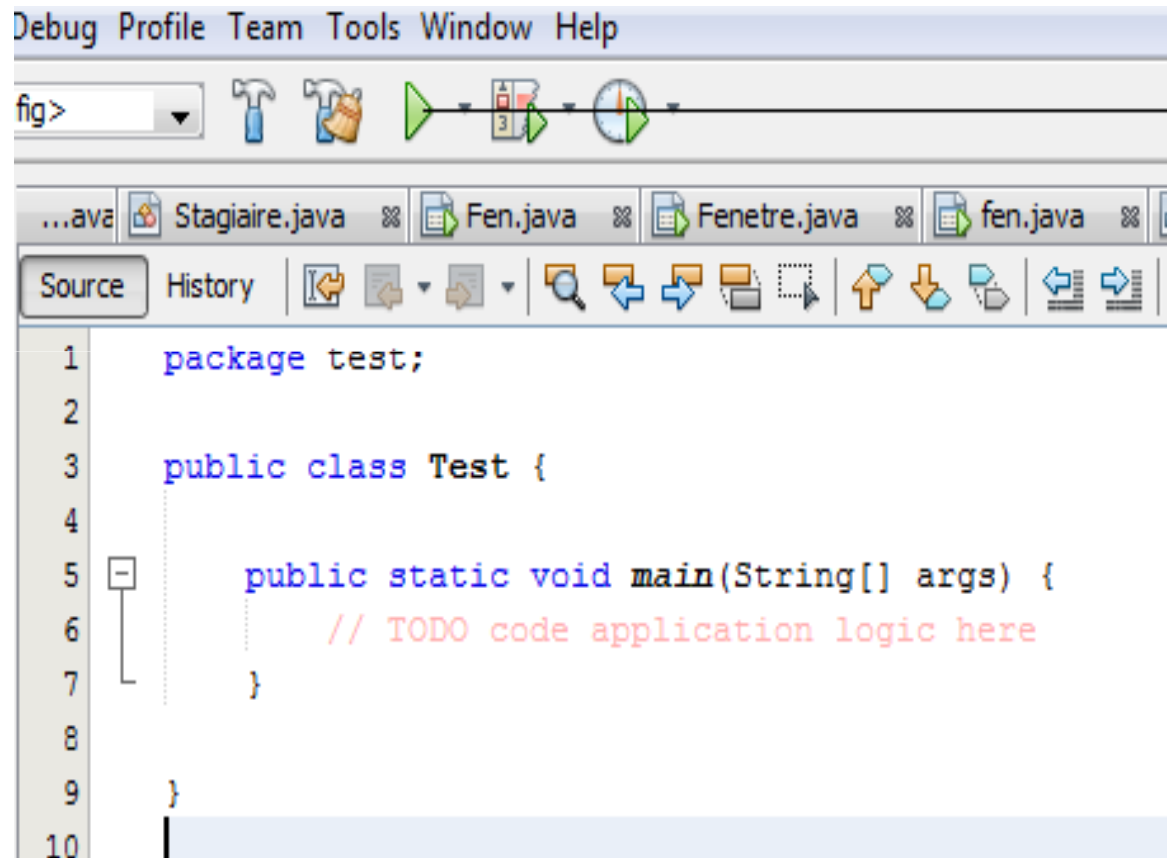
type de projet:
Java Application

Créer un nouveau Projet sous IDE NetBeans –suite-



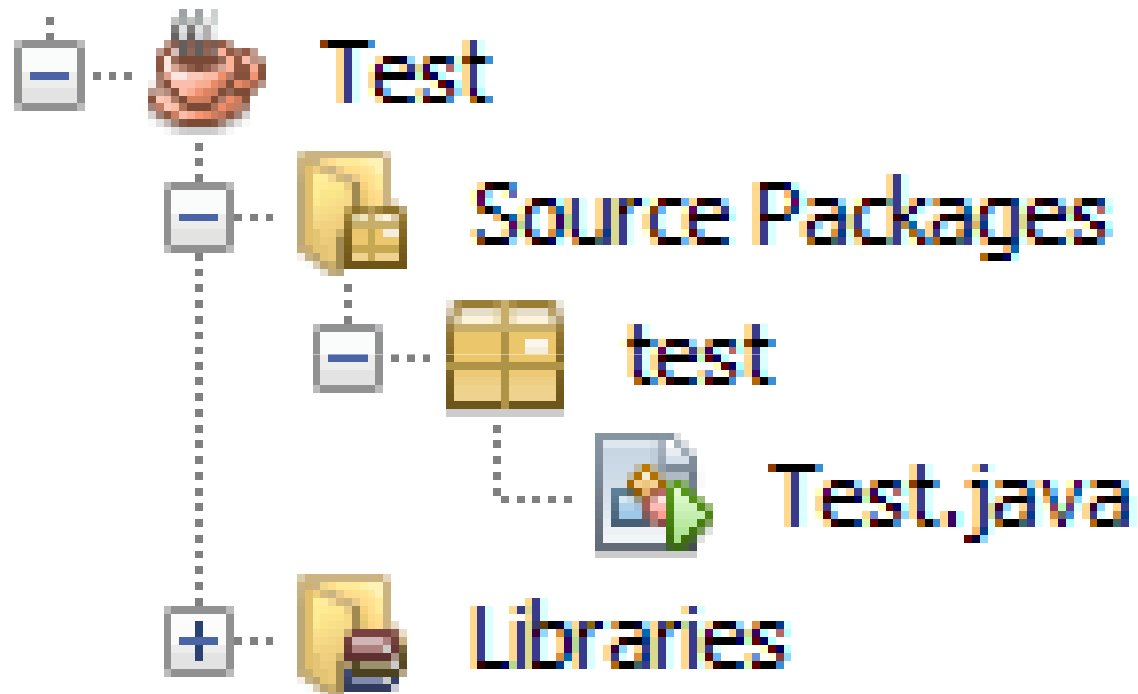
Nom de notre projet

Code généré



bouton d'execution
ou
F6

Arborescence du projet



Types primitifs en JAVA

Même types qu'en C:

- int
- float
- double
- short
- long
- char
- void

Plus deux nouveaux types :

- boolean
- byte

Plus la classe :

- String

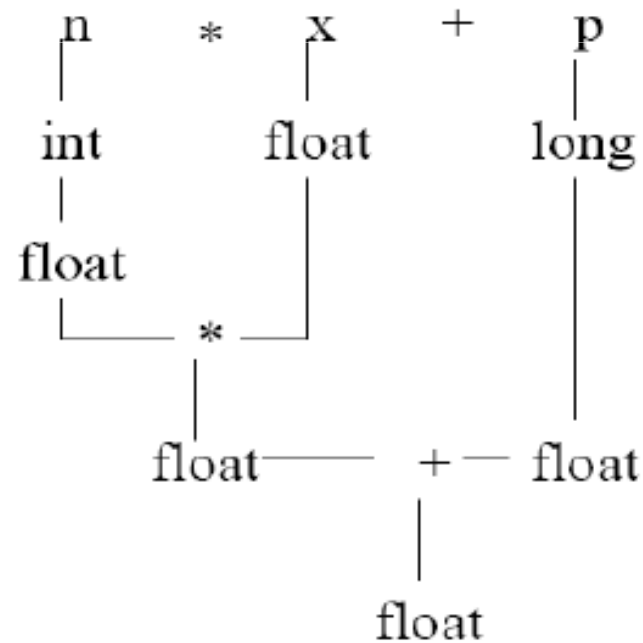
Déclaration :

```
int monEntier;  
String chaine;  
boolean test;
```

Conversion de type

Le problème de la conversion de type :

```
float x; float res;  
int n; int p;  
res=n*x+p;
```



Conversion de type –suite–

- byte \rightarrow short \rightarrow int \rightarrow long \rightarrow float \rightarrow double
- char \rightarrow int \rightarrow long \rightarrow float \rightarrow double

Opérateurs arithmétiques

Il existe cinq opérateurs principaux :

- $+$: addition $a+b$
- $-$: soustraction $a-b$
- $/$: division a/b
- $*$: multiplication $a*b$
- $\%$: modulo, $a \% b$ calcule le reste de la division de a par b .

Opérateurs de comparaison

- $<$: inférieur strictement à
- $<=$: inférieur ou égal à
- $>$: supérieur à
- $>=$: supérieur ou égal à
- $==$: égal à
- $!=$: différent de.

Opérateurs logiques

- `!` : négation
- `&` : "ET"
- `^` : "OU" exclusif
- `|` : "OU" inclusif
- `&&` : "ET" avec court-circuit
- `||` : "OU" inclusif avec court-circuit

Priorité des opérateurs

Opérateurs	associativité
() [] . ++(postfixé) -(postfixé)	g à d
+(unaire) -(unaire) ++(préfixé) -(préfixé) (unaire)! cast new	d à g
/ %	g à d
+ -	g à d
<< >> >>>	g à d
< <= > >= instanceof	g à d
== !=	g à d
&	g à d
^	g à d
—	g à d
&&	g à d
—	g à d
? :	g à d
= += -= *= /= %= <<= >>= >>>= & = = ^ =	d à g

Structures de contrôle et débranchements

Boucles répétitives

while do...while for

Instructions pour faire des choix

if...else switch...case

Commande d'affichage

`System.out.print(.....);`

Affichage sans retour à la ligne

`System.out.println(.....);`

Affichage avec retour à la ligne

Exemple1: On veut afficher le message Bonjour

`System.out.println("Bonjour");`

Exemple2: On veut afficher une variable avec sa valeur

`int A = 34; // on veut afficher A = 34`

`System.out.println("A= " + A);`

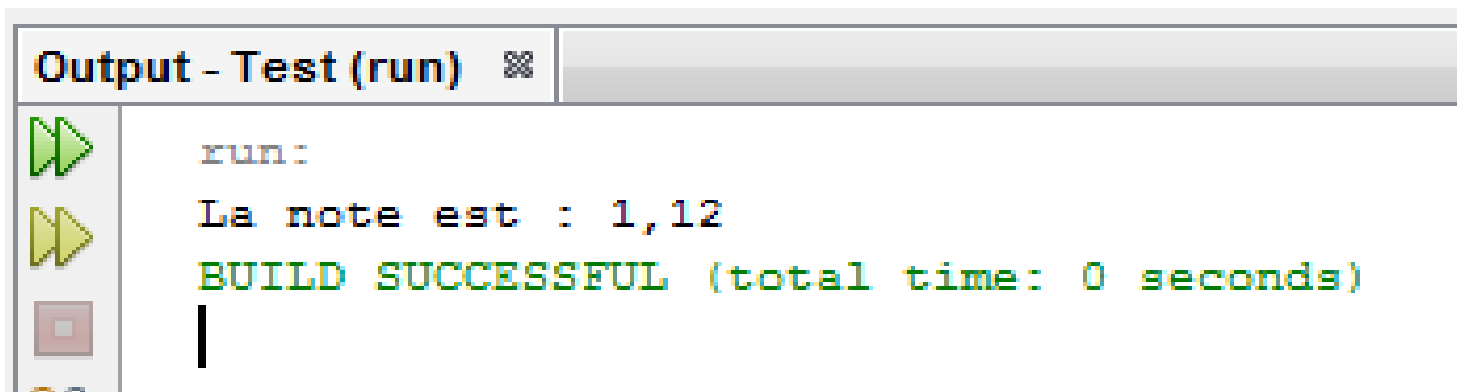
Commande d'affichage formaté

```
package test;

public class Test {

    public static void main(String[] args) {
        float n=1.1234F;
        System.out.printf("La note est : %.2f\n", n);
    }

}
```



```
Output - Test (run)  ⌵
run:
La note est : 1,12
BUILD SUCCESSFUL (total time: 0 seconds)
```

Lecture en mode console

Il existe plusieurs méthodes parmi ces méthodes, on a:
(nécessite d'importer le package java.util.*)

```
Scanner e = new Scanner(System.in);
```

Si on veut lire un entier:

```
int A; A= e.nextInt();
```

Si on veut lire un float:

```
float B; B= e.nextFloat();
```

Si on veut lire une chaîne sans espace:

```
String C; C= e.next();
```

Si on veut lire une chaîne avec espace:

```
String C; C= e.nextLine();
```

Introduction au type String

Déclaration

```
String chaineSalutation = "bonjour";
```

Une chaîne de caractère constante se déclare toujours entre guillemets "...".

Connaître la longueur d'une chaîne

```
int l = chaineSalutation.length();
```

Introduction au type String –suite–

Accès à un caractère

```
char cara1 = chaineSalutation.charAt(0);  
char cara1 = chaineSalutation.charAt(2);
```

La variable `cara1` contient le caractère *b*, la variable `cara2` contient le caractère *n*.

Concaténation : l'opérateur +

```
String ch1 = "Bonjour";  
String ch2 = " à tous";  
String ch = ch1 + ch2;
```

La variable `ch` contient la chaîne *"Bonjour à tous"*.

Introduction au type String –suite–

Impression d'une chaîne de caractères

```
System.out.println(chaineSalutation);  
System.out.println(ch1+ch2);  
System.out.println(ch);
```

Comparaison de chaînes La méthode `equals` qui teste l'égalité de deux chaînes de caractères : `ch1.equals(ch2)` ou `ch1.equals("Bonjour")`.

La méthode `compareTo` pour comparer deux chaînes de caractères dans l'ordre lexicographique (alphabétique) : `ch1.compareTo(ch2)`

- renvoie un entier strictement **négatif** si `ch1` est située avant `ch2` dans l'ordre lexicographique
- renvoie un entier strictement **positif** si `ch1` est située après `ch2` dans l'ordre lexicographique
- **0** si `ch1` contient la même chaîne que `ch2`.

Méthodes de classes

Une méthode de **classe** ou méthode **static** est une méthode qui peut être appelée par sa classe et elle peut manipuler que les membres static de sa classe.

```
public class Test{  
    public static int lireEntier(){.....}  
    public static void afficher() {lireEntier();.....}  
}
```

Dans une autre classe, on peut appeler la fonction afficher : **Test.afficher();**

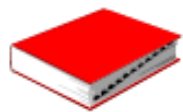
Classe et Objet

2

Objet

- Approche procédurale :
"Que doit faire mon programme ?"
- Approche orientée-objet :
"De quoi doit être composé mon programme ?"
- Cette composition est conséquence d'un choix de modélisation fait pendant la conception

Exemple: Gestion d'une bibliothèque



Germinal
E. Zola



Le Monde



Le seigneur des anneaux
J.R.R.Tolkien



Michel Martin
Bibliothécaire



Alice Dupont
Directrice



Arsène Deschamps
Lecteur

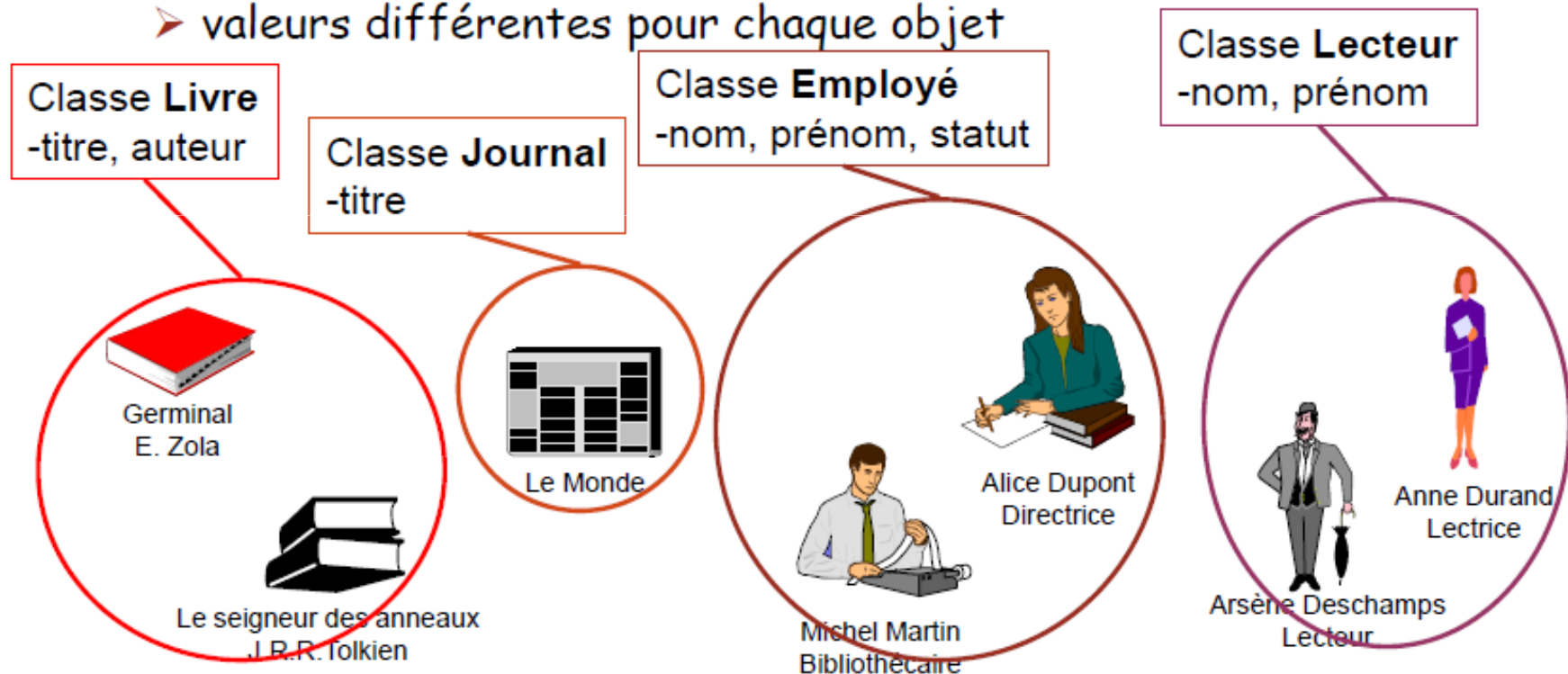


Anne Durand
Lectrice

Classe

Des objets similaires peuvent être informatiquement décrits par une même abstraction : une **classe**

- même **structure de données** et **méthodes de traitement**
- valeurs différentes pour chaque objet



Encapsulation

L'encapsulation des données est le premier et le plus important des concepts de la programmation objet. Il stipule que les données et les fonctions qui opèrent sur elles sont encapsulées dans des objets. Les seules fonctions à être autorisées à modifier les données d'un objet sont les fonctions appartenant à cet objet. Depuis l'extérieur d'un objet, on ne peut le modifier que par des fonctions faisant office d'interface. Ainsi, il n'est plus à craindre que des fonctions modifient indûment des données.

Héritage

Les objets, comme les données, possèdent un type que l'on appelle classe. Les classes peuvent être organisées en hiérarchies, chaque classe héritant de sa classe mère ou super-classe.

L'héritage est ainsi source d'économie de code, une classe peut être définie comme la descendante d'une classe (ou sous-classe).

Polymorphisme

Le troisième principe de base de la programmation objet est le polymorphisme. Il passe plus inaperçu que les précédents.

Des fonctions différentes dans les classes différentes peuvent prendre le même nom. Ainsi, dans une hiérarchie de classes d'éléments graphiques la fonction *dessiner()* aura le même nom pour un polygone ou un cercle, cependant les techniques utilisées pour dessiner ces éléments sont différentes. Le polymorphisme est beaucoup plus puissant qu'il n'y paraît à première vue. Il fait économiser des identificateurs de fonctions et rend les notations plus lisibles.

Définition d'une classe

La notion de classe est un enrichissement de la notion usuelle de structure en C (*struct*) en C. Une classe permet de définir un type d'objet, éventuellement compliqué, associant des données et des procédures qui accèdent à ces données. Les données se présentent sous forme de champ désignés par identificateurs et dotés d'un type. A une donnée, on associe un mot clé qui désigne l'accessibilité de cette donnée.

Les procédures, également appelées *méthodes*, définissent les opérations possibles sur ces composants. A une méthode, on associe un mot clé désignant l'accessibilité de cette méthode.

Exemple

```
public class Point {  
    private int _x;  
    private int _y;  
    public Point() { _x = 0 ; _y = 0 ; }  
    public Point(int x, int y)  
    {    _x = x ; _y = y ; }  
    public void avancer(int dx, int dy)  
    {  
        _x = _x + dx ; _y = _y + dy ;  
    }  
}
```

Exemple - suite

La classe Point possède deux attribut : `_x`, `_y` et trois méthodes `Point()`, `Point(int, int)`, `avancer(int, int)`.

Le mot clé `private` indique que l'attribut est privé il ne sera accessible que par les méthodes appartenant à la classe. Un attribut public est accessible par une méthode appartenant à une autre classe

Constructeur

Une classe peut définir une ou plusieurs procédures d'initialisation appelées *constructeurs*. Un constructeur est une méthode avec aucune valeur de retour et qui porte le même nom que la classe. S'il existe plusieurs méthodes dans une classe, elles se distinguent par le type ou le nombre de Paramètres (signature) : c'est la surdéfinition

Point() et *Point(int, int)*

Constructeur -suite

Le constructeur permet de créer une instance de la classe. Une instance de classe est dite *objet*.

Pour créer un Point on va :

Déclarer un *handle* sur un Point.

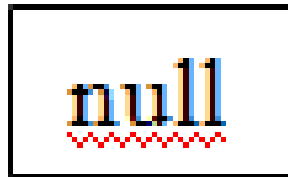
Point p ; (p est un handle sur un point).

- Remarque

Un handle est un numéro. Par exemple, quand une application alloue de la mémoire, le compilateur ne rend pas un pointeur mais un numéro. Ce numéro est associé à une zone mémoire.

Constructeur -suite

La déclaration seule ne crée pas d'objet. Elle associe l'identificateur à une référence appelée *null* qui ne fait référence à rien.



Constructeur -suite

Pour créer une instance de la classe point (on dit objet de la classe Point), il faut exécuter **new** en citant un constructeur de la classe Point, avec ses paramètres.

La primitive new rend en résultat la référence sur l'objet crée, qu'il suffit alors d'affecter à la variable déclarée :

```
new Point(2,0)
```

```
Point p = new Point(2,0) ;
```

destruction des objets: *destructeur*

Avec Java, plus besoin d'écrire une fonction destructeur. En effet, il existe un programme appelé 'garbage collector' (ramasseur d'ordures) qui est exécuté automatiquement dès que la mémoire disponible devient inférieure à un certain seuil.

Mot final

❑ Variable final

Une variable déclarée *final* ne peut plus voir sa valeur modifiée (final int x=1 ;).

❑ Méthodes final

Les méthodes final ne peuvent pas être redéfinies dans les classes dérivées.

❑ Classes final

Une classe final ne peut être étendue pour créer des sous-classes.

Mot static

❑ **Attribut static**

Un attribut statique (déclaré avec le mot clé **static** : *static int x=3*) est un attribut partagé par tout les objets de la classe (tout les objet ont la même valeur pour cet attribut).

❑ **Méthode static**

Une méthode statique (déclarée avec le mot clé **static** : *public static void f()*) est une méthode qui n'a accès qu'aux membres static de la classe.

Objet this

this désigne une référence sur l'objet courant.

Exemple

```
public Point(int _x, int _y)
{
    this._x = _x ;
    this._y = _y ;
}
```

Pour distinguer le paramètre du constructeur à l'attribut, on utilise le mot clé **this**

Méthode this()

Un constructeur peut appeler un autre **via cette méthode**, elle doit figurer en **première ligne**

Exemple

```
public class Point{  
    public Point(int _x, int _y)  
    {  
        this._x = _x ;  
        this._y = _y ;  
    }  
    public Point()  
    {  
        this(0,0);  
    }  
    .....  
}
```



Package et accessibilité

La portée d'une classe est définie comme la zone dans laquelle elle est visible. Seules les classes publiques sont accessibles depuis l'extérieur du package.

<i>Modificateur de portée</i>	<i>Portée</i>
aucun	Le paquetage
public	Partout

Portée des membres d'une classe

****protected*** : Les membres d'une classe peuvent être déclarés *protected*. Dans ce cas, l'accès en est réservé aux méthodes des classes dérivées, des classes appartenant au même package ainsi qu'aux classes appartenant au même package que les classes dérivées.

****private*** : Accès au niveau de la classe

****public*** : accès partout

- **Remarque**

Un attribut sans modificateur (modificateur par défaut) n'est pas accessible depuis l'extérieur du package de la classe.

Les tableaux

En programmation, un tableau désigne un ensemble d'éléments de même type identifiés par un nom unique.

Chacun des éléments étant ensuite repéré par un indice précisant sa position au sein de l'ensemble.

En Java, les tableaux sont considérés comme des objets, les tableaux à plusieurs indices s'obtiennent par composition de tableaux.

Déclaration d'un tableau

Imaginons que nous voulions créer un tableau d'entiers, deux déclarations sont possibles :

```
int t[];
```

```
int []t;
```

La différence entre les 2 déclarations a une importance lorsque l'on souhaite déclarer plusieurs tableaux :

```
int [] t1,t2; //déclaration de 2 tableaux d'entiers
```

```
int t1[], t2[]; //même chose
```

```
int t1[], n, t2[]; //2 tableaux t1 et t2, n est un entier
```

```
Point a, tp[], b; //a et b sont de type Point, tp est un tableau  
d'objets Point
```

Création d'un tableau

Pour créer un tableau on utilise l'opérateur **new** comme pour des objets classiques.

Voici un exemple montrant comment créer un tableau de 10 entiers :

```
int t[] = new int[10];
```

Il est également possible de fournir une liste d'expression entre accolades. Voici un exemple de création d'un tableau d'entiers à 5 éléments.

```
int t[] = {1, 2, 7, 10, 0};
```


Création d'un tableau (suite)

L'instruction précédente équivaut aux instructions suivantes:

```
int [] t = new int[5];
```

```
t[0] = 1; t[1] = 2; t[2] = 7; t[3] = 10; t[4] = 0;
```

Attention: Une fois un tableau créé, on ne peut plus modifier sa taille. Par contre, on peut créer un nouveau tableau et -si besoin - recopier les valeurs du tableau initial. La taille d'un tableau est toujours positive.

Taille d'un tableau

Il est possible d'avoir accès à la taille d'un tableau à l'aide du champ `length`.

```
int t[] = new int[6];
```

```
System.out.println("taille de t "+t.length);
```

```
//affiche 6
```

```
t = new int[2];
```

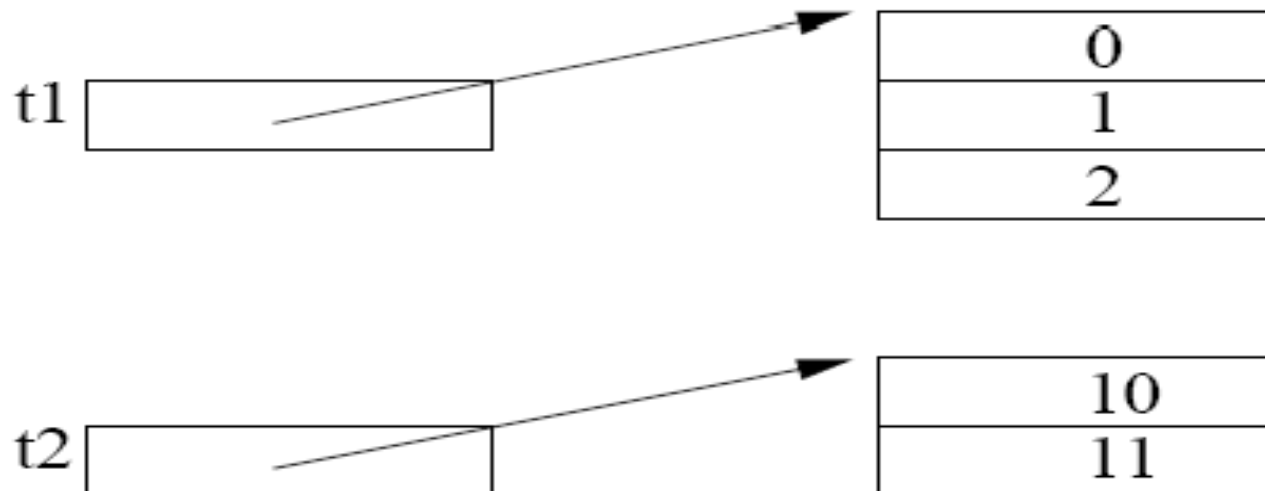
```
System.out.println("taille de t "+t.length);
```

```
//affiche 2 Ici length est vu comme un champ et
```

```
//non comme une méthode.
```

Affectation des tableaux

```
int [] t1 = new int[3];  
for(int i=0; i<t1.length; i++)  
    t1[i] = i;  
int [] t2 = new int[2];  
for(int i=0; i<t2.length; i++)  
    t2[i] = 10+ i;
```

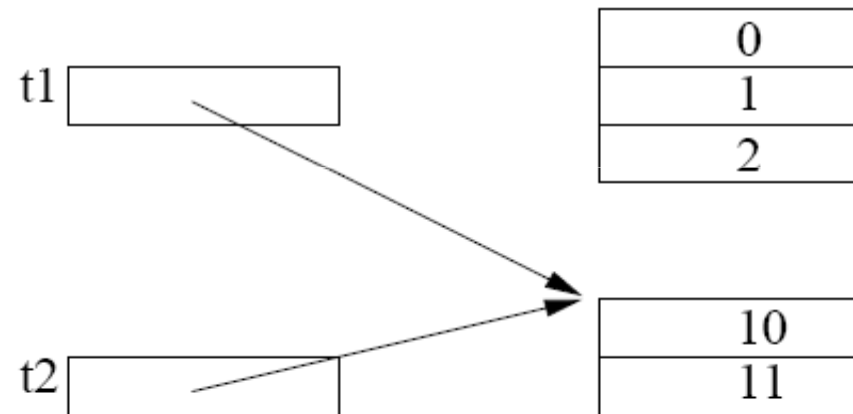


Affectation des tableaux(suite)

Si maintenant on exécute :

```
t1 = t2; //la référence contenue dans t2 est copiée dans t1
```

Nous aboutissons alors à cette situation :



Maintenant si l'on exécute les instructions suivantes :

```
t1[1] = 5;  
System.out.println(t2[1]); //affiche 5 !!!
```

L'ancien objet référencé par `t1` devient candidat au ramasse-miettes. Lors de l'affectation de références de tableau, il n'y a aucune copie des valeurs des éléments du tableau.

Tableau d'objets

Exemple de tableau d'objets

```
public class Point
{
    private int x,y;

    public Point(int x,int y)
    {
        this.x = x;
        this.y = y;
    }

    public void affiche()
    {
        System.out.println("Point : "+x+", "+y);
    }
}
```

Tableau d'objets (suite)

```
public class TabPoint
{
    public static void main(String [] args)
    {
        Point [] tp;
        tp = new Point[3];
        tp[0] = new Point(1,2);
        tp[1] = new Point(4,5);
        tp[2] = new Point(8,9);
        for(int i=0; i<tp.length; i++)
            tp[i].affiche();
    }
}
```

Résultat :

```
Point : 1, 2
Point : 4, 5
Point : 8, 9
```

Tableau en argument ou en retour d'une méthode

Lorsque l'on transmet un nom de tableau en argument d'une méthode, on transmet en fait une copie de la référence au tableau. La méthode agit directement sur le tableau concerné et non sur une copie. Lorsqu'une méthode retourne un tableau, elle retourne sa référence et non pas une copie de ce tableau.

Tableau en argument ou en retour d'une méthode (suite)

```
public class Util {  
    public static void raz(int t[]) { .....}  
    public static void affiche(int t[]) {.....}  
    public static int[] creerTab(int n) {.....}  
    public static int  occ(int t[], int val) {.....}  
    public static void tri(int t[]) {.....}  
    public static int[] inverse(int t[]) {.....}  
}
```


Tableau à plusieurs indices

Premier exemple. Ces trois déclarations sont équivalentes pour un tableau à 2 dimensions :

```
int t [] [];  
int [] t [];  
int [] [] t;
```

```
int t [] [] = { new int [3], new int [2] };
```

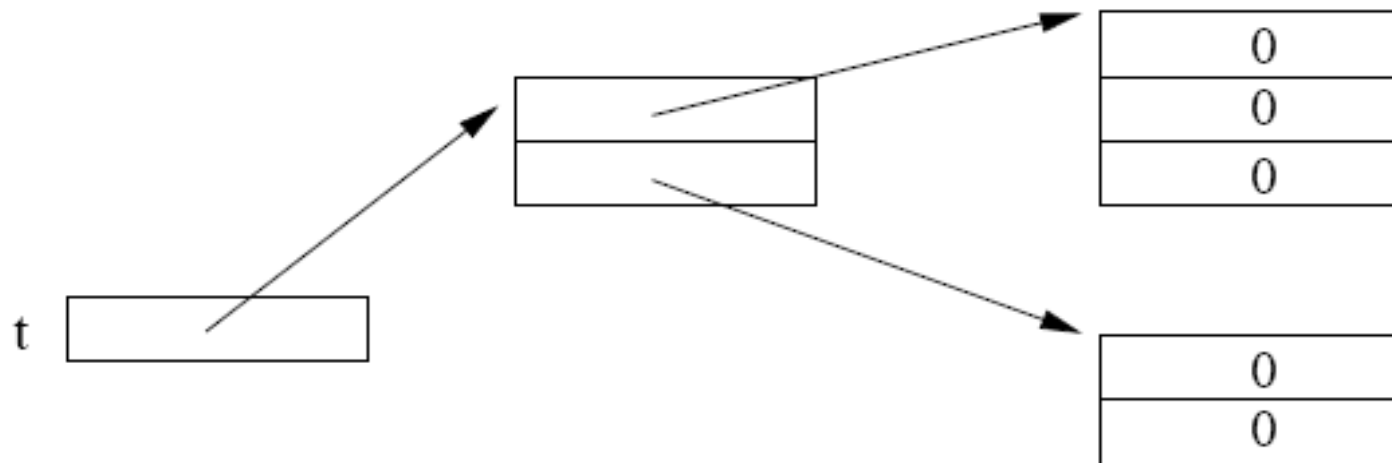


Tableau à plusieurs indices (suite)

On constate alors que :

- la notation `t[0]` désigne la référence au premier tableau de 3 entiers,
- la notation `t[0][1]` désigne le deuxième élément de ce tableau (pour rappel les indices commencent à 0),
- la notation `t[1]` désigne la référence au deuxième tableau de 2 entiers,
- la notation `t[0][i-1]` désigne le *i*ème élément de ce tableau pour *i* compris entre 1 et 2,
- l'expression `t.length` vaut 2,
- l'expression `t[0].length` vaut 3,
- l'expression `t[1].length` vaut 2.

Tableau à plusieurs indices (suite)

Second exemple.

```
int t [] [];  
t = new int [2] [];  
int [] t1 = new int [3];  
int [] t2 = new int [2];  
t[0] = t1;  
t[1] = t2;
```

La situation peut s'illustrer comme ceci :

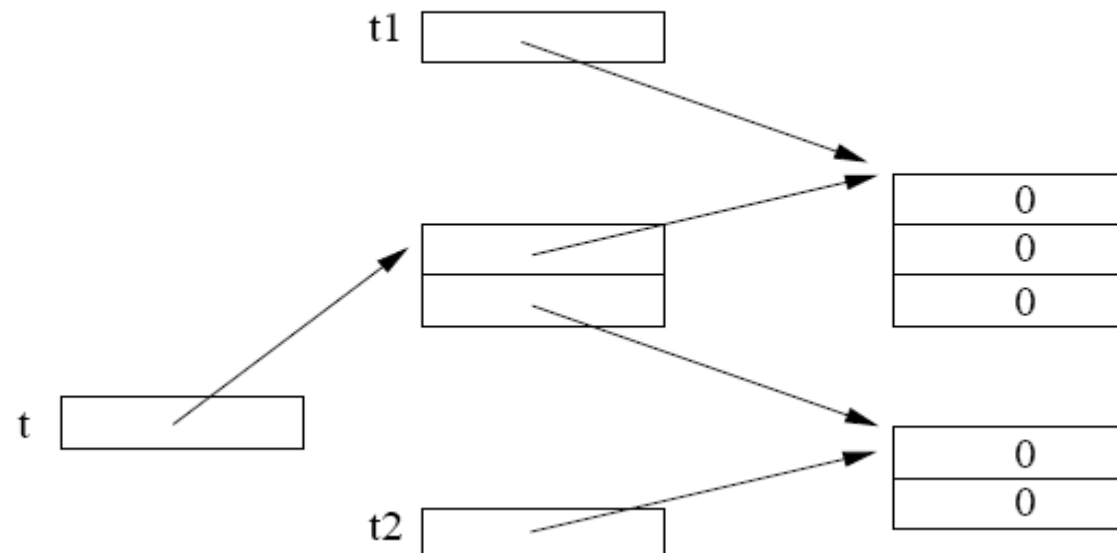


Tableau à plusieurs indices (suite)

```
class Util {  
    static void raz(int t[] []) { int i,j;  
        for(i=0;i<t.length;i++)  
            for(j=0;j<t[i].length;j++) t[i][j]=0; }  
    static void affiche(int t[] []) { int i,j;  
        for(i=0;i<t.length;i++) {  
            System.out.println("ligne de rang "+i+" =");  
            for(j=0;j<t[i].length;j++)  
                System.out.println(t[i][j] + " ");  
            System.out.println(); } } }
```

Tableau à plusieurs indices (suite)

```
public class TestTab2
{
    public static void main(String arg[])
    {
        int t [][] = {{1,2,3}, {11, 12}, {21, 22, 23, 24} }
        System.out.println("t avant raz :");
        Util.affiche(t);
        Util.raz(t);
        System.out.println("t apres raz :");
        Util.affiche(t);
    }
}
```

Résultat.

```
t avant raz
ligne de rang 0= 1 2 3
ligne de rang 1= 11 12
ligne de rang 2= 21 22 23 24
t apres raz
ligne de rang 0= 0 0 0
ligne de rang 1= 0 0
ligne de rang 2= 0 0 0 0
```

Tableaux réguliers

Cas des tableaux réguliers. Si on souhaite créer une matrice de NL lignes et NC colonnes on peut toujours procéder comme ceci :

```
int t[] [] = new int [NL] [];  
int i;  
for(i=0; i<NL; i++)  
    t[i] = new int [NC];
```

On peut également écrire en Java :

```
int t[] [] = new int [NL] [NC];
```

Exercices sur les tableaux

Utilitaire Ecrire une classe disposant des méthodes statiques suivantes :

- une méthode **genere** et qui fournit en retour un tableau des n premiers nombres impairs, la valeur **n** étant donnée en argument.
- une méthode **somme** qui reçoit en argument 2 tableaux d'entiers de même taille et qui fournit en retour un tableau représentant la somme des deux vecteurs.

Matrice Ecrire une classe définissant le concept de matrice carré avec un constructeur à un seul argument. Cette classe contiendra les méthodes suivantes :

- **multiplie** qui prend en argument un entier et qui multiplie tous les éléments de la matrice par cet entier
- **somme** qui prend en argument une matrice de même dimension telle qu'à la fin de la méthode on obtienne la somme des 2 matrices.
- une méthode **affiche** sans argument qui affiche tous les éléments de la matrice à l'écran.

Moyenne

Ecrire un programme qui calcule la moyenne d'un ensemble de notes rentrées au clavier. Le programme demandera d'abord le nombre de notes à lire, puis demandera et récupérera les différentes notes rentrées par l'utilisateur. Le programme affichera la moyenne des notes ainsi que la proportion de notes supérieures à la moyenne et la proportion de notes inférieures à la moyenne.