Introduction à la programmation en Java

UFR Sciences de Nice

Licence Math-Info 2006-2007

Module L1I1

Frédéric MALLET

Jean-Paul ROY

Où en sommes-nous ?

- Nous savons rédiger le texte d'une classe d'objets, avec dans l'ordre : ses champs, ses constructeurs, ses méthodes.
- Nous pouvons exprimer qu'une méthode a ou n'a pas de résultat.
- L'utilisation d'une conditionnelle if n'a plus de secrets.
- Nous commençons à acquérir des capacités d'abstraction et de modularité.
- Nous savons faire la différence entre une méthode d'instance et une méthode de classe (statique).
- Nous avons fait connaissance avec les 3 boucles :

`for, while, do…while`

qui nous permettent de programmer des répétitions.
- Nous savons piloter une tortue pour faire du graphisme.

Que nous manque-t-il ?

- Beaucoup de choses en vérité !
- Pour l'instant :

  · construire de vastes **collections d'objets**

  · **parcourir** et **analyser** les collections créées.

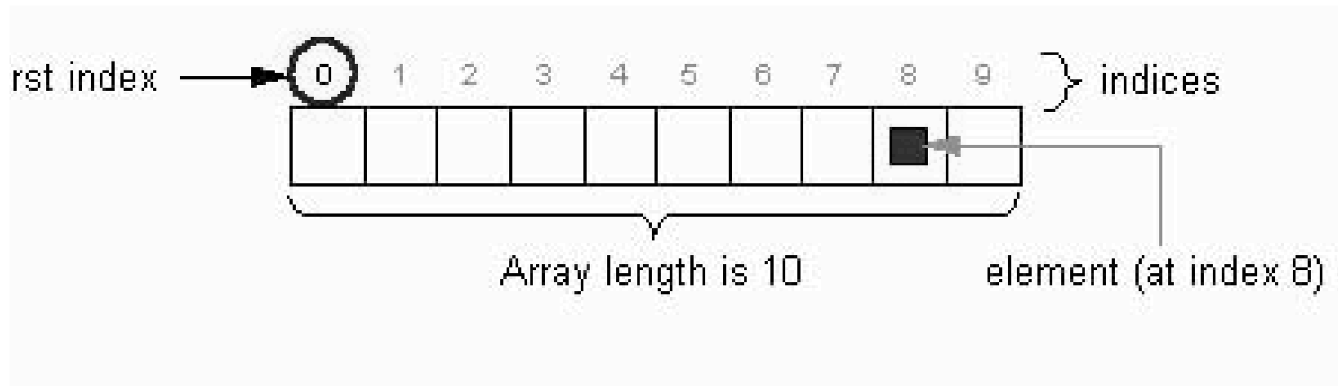
- Mais :

  · Les collections seront-elles *ordonnées* ou *en vrac* ?

  · Leur nombre d'éléments sera-t-il *fixe* ou *variable* ?

Collections d'objets

(de taille fixe)



Pourquoi des collections ?

- Il est souvent nécessaire de stocker de nombreuses données dans des *collections* : bibliothèques, sécurité sociale, cartes d'étudiants, albums de photos, relevés météo, etc.
    - Souvent des milliers d'éléments, parfois des millions !
    - Il est hors de question de créer une variable par élément.
- La taille (nombre d'éléments) d'une collection peut être fixe (les lettres de l'alphabet) ou variable (ma collection de DVD).
- Nous allons étudier les tableaux qui sont des collections :
    - de taille fixe : on choisit un nombre d'éléments puis on s'y tient ;
    - dont les éléments sont tous de même type, et numérotés.

Les collections de type tableau

• Comme la plupart des langages, Java propose des **tableaux** : collections d'éléments *numérotés de même type* (objets ou pas), en nombre fixé une fois pour toutes !

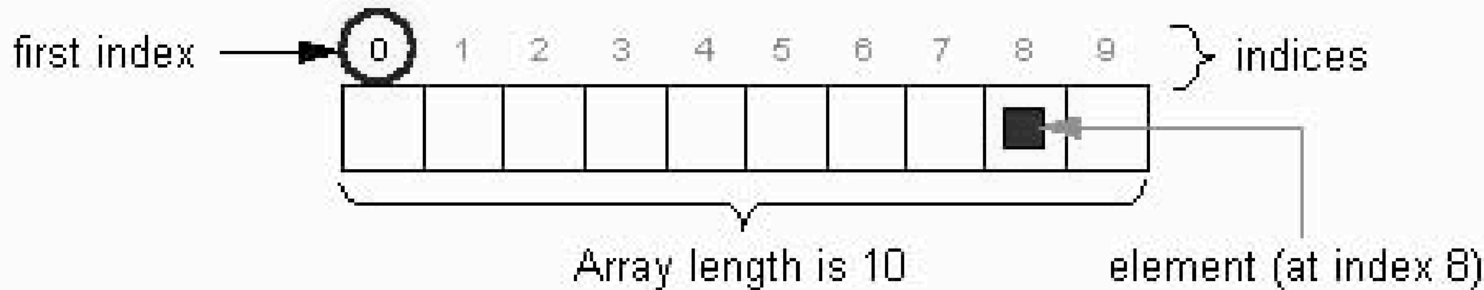
• On pourra donc seulement :

- consulter ou modifier directement l'élément numéro k

• Il est garanti que le temps d'accès à l'élément numéro k ne dépend pas de k (il a lieu *en temps constant*).

• Le souhait de compatibilité avec le langage C va entraîner quelques bizarreries. Les tableaux sont des objets, mais avec une syntaxe bien spéciale…

Construction d’un tableau d’entiers



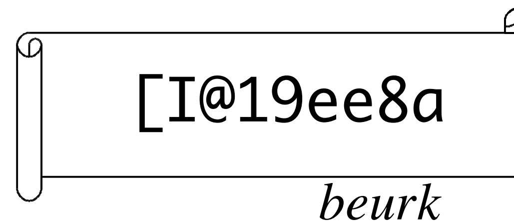

```
int[] tab = new int[10];
```

Soit **`tab`** un nouveau tableau de 10 entiers.

`tab.length` ⟹ **`10`**

`length` est un champ public non modifiable du tableau, pas une méthode comme dans ***`String`*** !

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



beurk

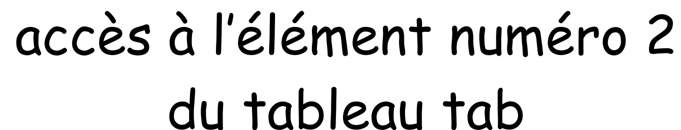
Notation indexée des éléments

• Un tableau n'est autre qu'une suite finie de variables indexées.
On aurait pu opter pour 10 variables **`tab0, tab1,`**...

• En Math : $(\mathbf{tab}_0 , \mathbf{tab}_1 , \mathbf{tab}_2 , \dots , \mathbf{tab}_9)$

• En Java : **`tab[0]`**, **`tab[1]`**, **`tab[2]`**, ... , **`tab[9]`**

• Ne pas confondre **`tab[2]`** et **`tab(2)`** !!!



Appel de la fonction tab
sur l'argument 2

• En Java, l'accès est sécurisé (pas en C) : seuls **`tab[i]`** existent
pour $0 \leq i \leq 9$:

```
System.out.println(tab[10]);
```

java.lang.ArrayIndexOutOfBoundsException

Modification d'un élément

• Puisque les composantes d'un tableau sont des variables (indexées), on peut les *affecter* avec l'opérateur **=**

```
tab[2] = 2004;
```

L'élément numéro 2 de **tab** devient égal à 2004.

```
tab[2] = tab[2] + 2 * tab[3];

tab[2] += 2 * tab[3];
```

```
tab[2] + 1 = 0;
```
Pas de variable à gauche du signe =

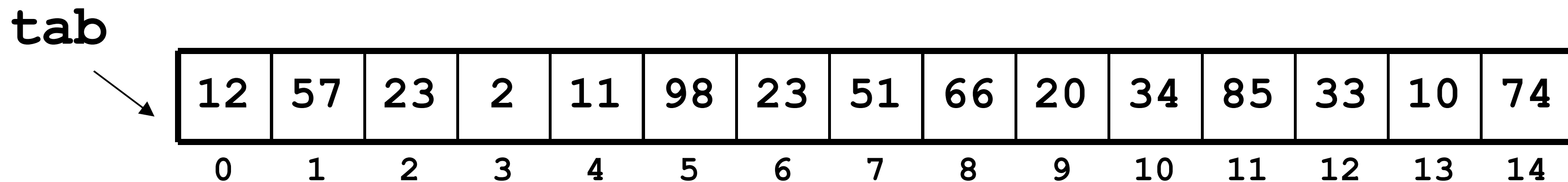
```
tab[2] = "bonjour";
```
Bad types in assignment

```
tab[10] = 0;
```
IndexOutOfBoundsException

Initialisation d'un tableau

- Il suffit de modifier en séquence tous les éléments du tableau.
- La boucle **`for`** s'utilise beaucoup avec les tableaux.
- Construisons un tableau **`tab`** d'entiers aléatoires de **`[0,99]`**:

```
int[] tab = new int[15];                // 15 éléments
Random rand = new Random();
for(int i=0; i<tab.length; i++) {
  tab[i] = rand.nextInt(100);           // tab[i] ∈ [0,99]
}
```



• Il est possible d'**initialiser le tableau <u>en même temps</u> qu'on le déclare** si l'on connaît tous ses éléments :

```
int[] tab1 = {8, 3, -5, 3, 6} ;
```

• Tous les éléments doivent être de même type !

```
int[] tab2 = {8, 3.14, -5, 3, 6} ;

int[] tab3 = {5, 3, "hello", 8} ;

int y = 3 ;

int[] tab4 = {y, y+1} ;
```

• La longueur sera calculée automatiquement :

• Attention, ceci serait *illégal* :

```
int[] tab1 ;
tab1 = {8, 3, -5, 3, 6} ;
```

Affichage d'un tableau *à l'horizontale :*

```
for(int i = 0; i < tab.length; i++) {
  System.out.print(tab[i] + " ");
}
System.out.println();
```

un espace pour séparer les éléments…

pour aller à la ligne à la fin !

• Si l'on affiche plusieurs fois le tableau en changeant l'ordre des éléments, les colonnes sont en zigzag !

```
245 6 5674 33 786
5674 33 6 786 245
```

• Le choix de l'espace `" "` était maladroit. Mieux vaut utiliser une ou plusieurs *tabulations* `"\t"`

```
245     6       5674    33      786
5674    33      6       786     245
```

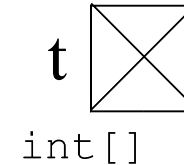
• Traduction d’une boucle **`for`** en boucle **`while`** :

```
for(int i = 0; i < tab.length; i++) {
  tab[i] = rand.nextInt(100);
}
```

```
int i = 0;
while (i < tab.length) {
  tab[i] = rand.nextInt(100);
  i++;
}
```

Les tableaux sont des objets !

- Un tableau doit être construit AVANT d'être utilisé :
  - Déclaration : **`int[] t;`**



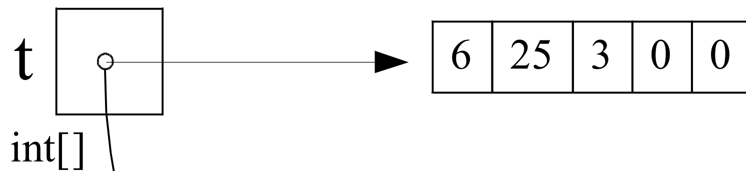
`t.length` ⇒ *NullPointerException*

`t[0]` ⇒ *NullPointerException*

  - Construction / initialisation : on choisit de manière <u>définitive</u> le nombre d'éléments !

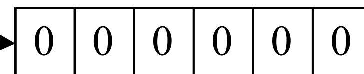
```
t[0]=6;
t[1]=25;
t[2]=3;
```

`t = new int[5];`

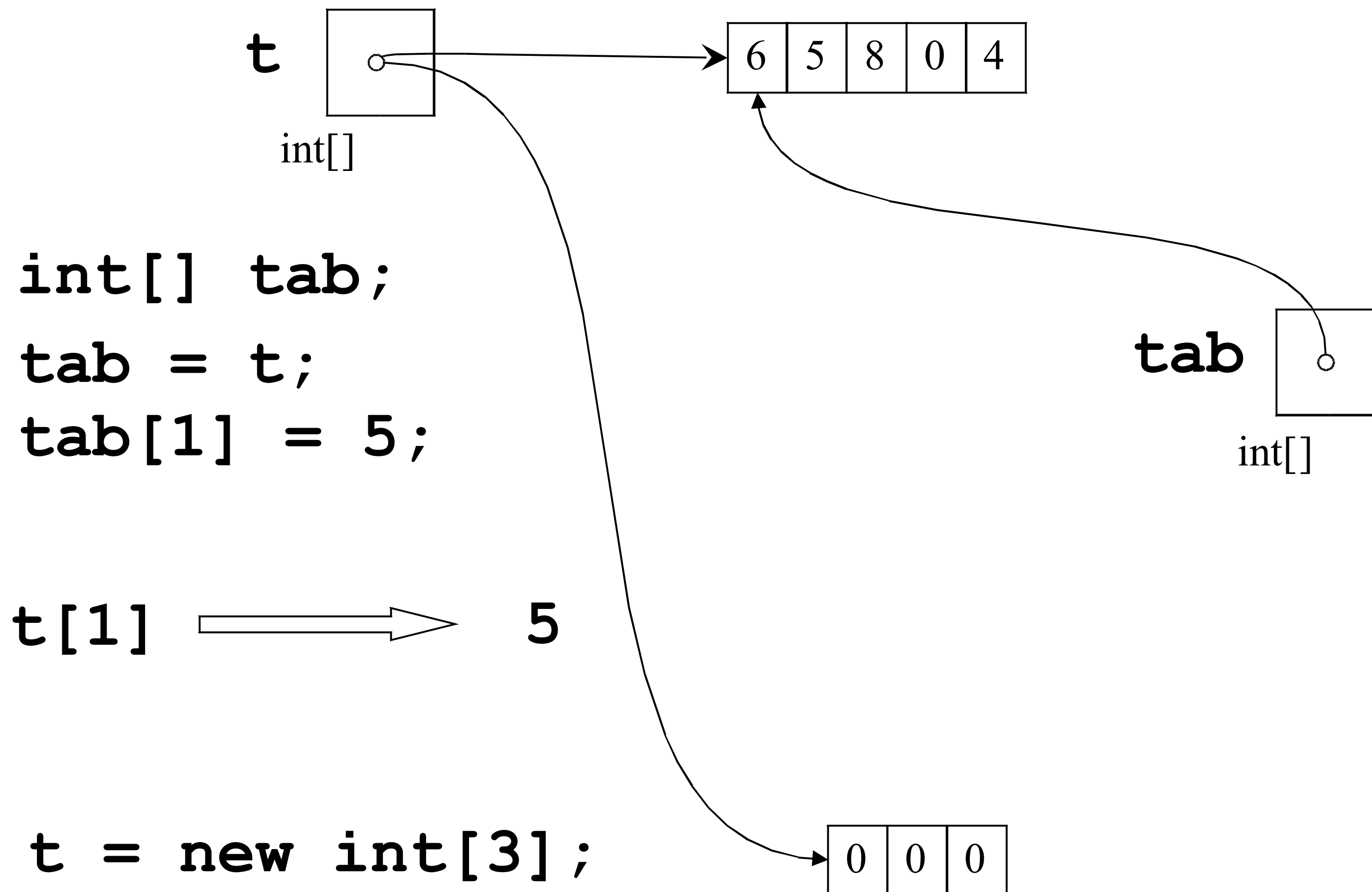


`t.length` → 5

`t = new int[6] ;`



Affectation de tableaux



Ne pas confondre les références et les objets !

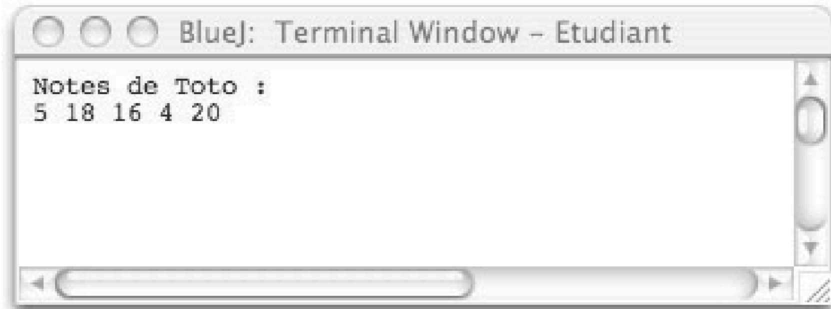
Modélisation des notes d'un étudiant

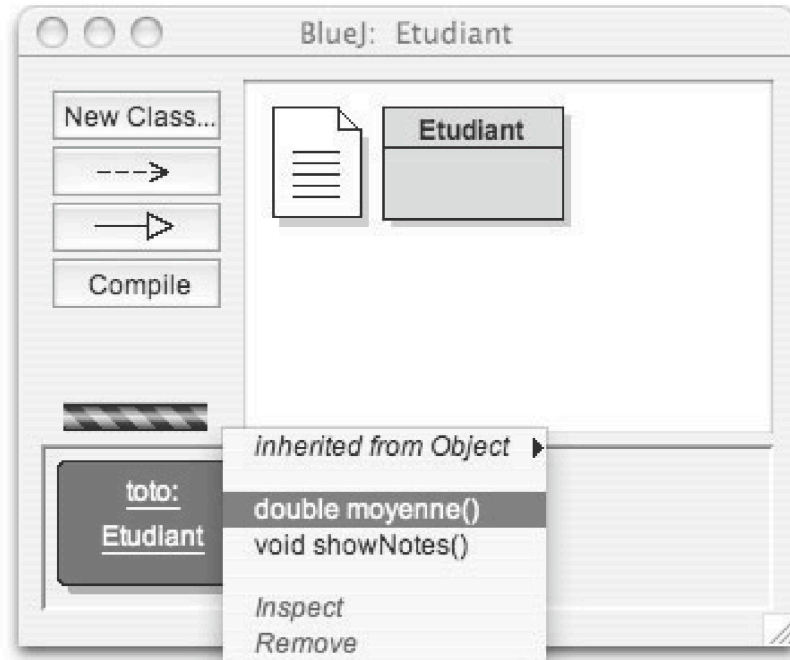
```
import java.util.Random;
public class Etudiant {
   private String nom;
   private int[] notes;
   private Random rand = new Random();

   /** le constructeur */
   public Etudiant(String nom, int n) {   //  n notes
      this.nom = nom;
      notes = new int[n];
      for(int i = 0; i < n; i++) {
        notes[i] = rand.nextInt(21);        // au hasard
      }
   }
```

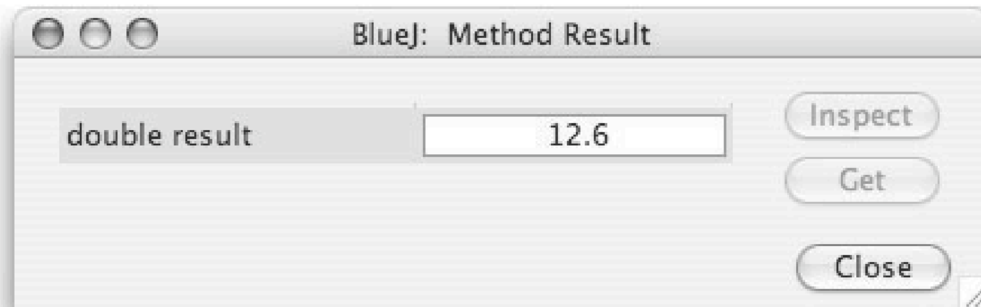
```
Etudiant e = new Etudiant("Toto",6);
```

Soit e un étudiant de nom « Toto » ayant 6 notes.

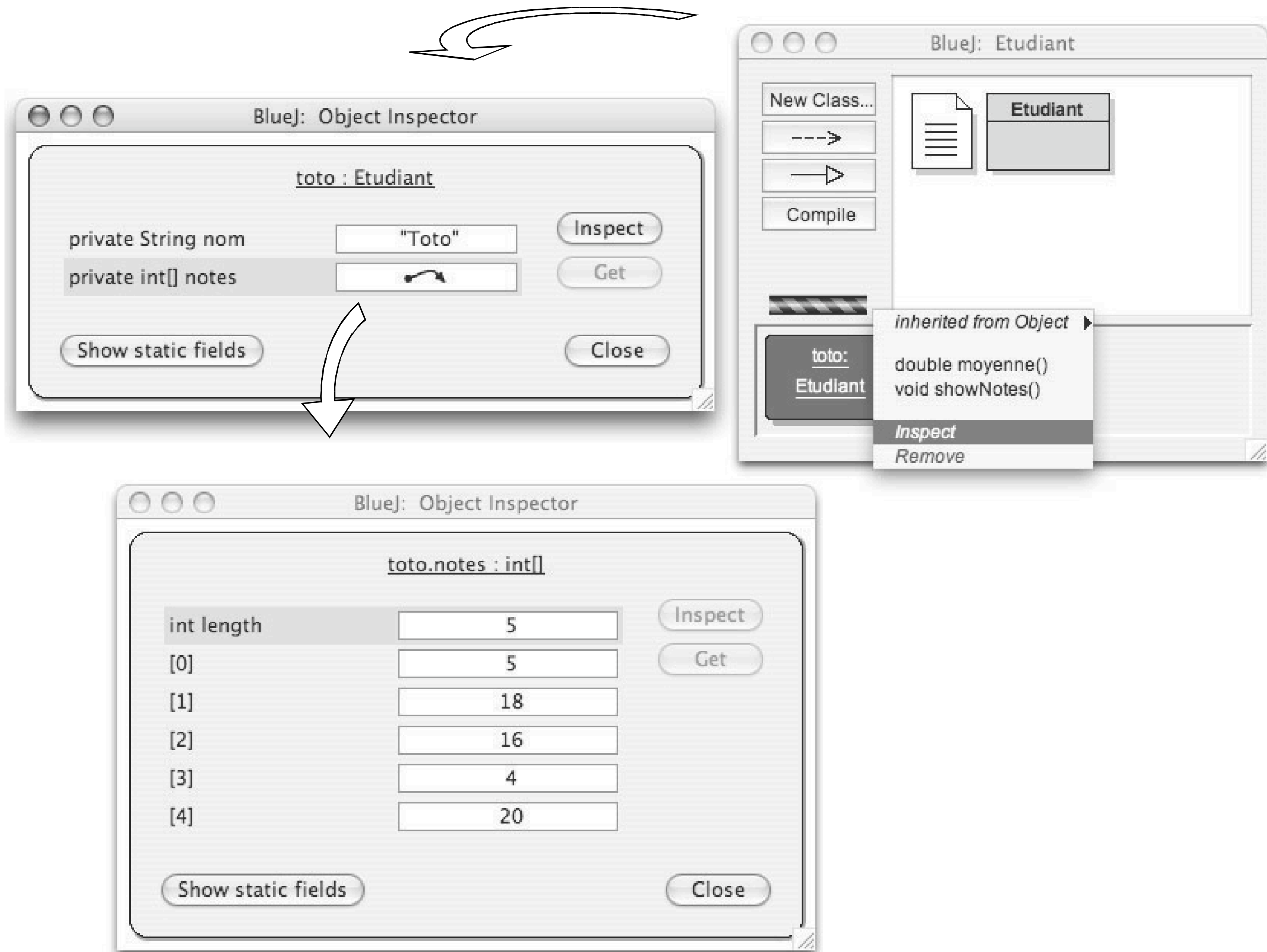
```
/** affichage des notes à l'horizontale */
public void showNotes() {
  System.out.println("Notes de " + nom + " :");
  for(int i = 0; i < notes.length; i++) {
     System.out.print(notes[i] +  "\t");
  }
  System.out.println();
}

/** calcul de la moyenne approchée */
public double moyenne() {
  double somme = 0;
  for(int i = 0; i< notes.length; i++) {
    somme = somme + notes[i];
  }
  return somme / notes.length;
}
}
```


BlueJ: Terminal Window - Etudiant
Notes de Toto :
5 18 16 4 20


BlueJ: Etudiant
New Class...
Compile
Etudiant
toto:
Etudiant
inherited from Object
double moyenne()
void showNotes()
Inspect
Remove


BlueJ: Method Result
double result
12.6
Inspect
Get
Close

• Visualisation du champ `notes` avec **l'inspecteur** :



Recherche séquentielle d'une note donnée

- Ex : l'étudiant a-t-il une note donnée ? Explorons le tableau jusqu'à trouver la note cherchée ou sortir du tableau…

```
public boolean existeNote(int note) {
   int i = 0;
   while(i < notes.length && notes[i] != note) {
      i++;
   }

   // état : i == notes.length || notes[i] == note
   if (i == notes.length) {
      return false;
   } else {
      return true;
   }
}
```

```
return i != notes.length;
```

• Modifions la méthode pour qu'elle rende le rang ≥ 0 de la note trouvée (ou bien **-1** si elle n'est pas présente) :

```
public int rangNote(int note) {
    int i = 0;
    while(i < notes.length && notes[i] != note) {
        i++;
    }
    if (i != notes.length) {
        return i;
    } else {
        return -1;
    }
}
```

Encore un while ?

Et pourquoi pas une boucle for ?…

- Pourquoi n'avons-nous pas opté pour une boucle `for` ?

Parce que l'on ne sait pas a priori combien de fois la boucle va tourner !

- Mais on peut néanmoins adopter le point de vue suivant :

Je décide de parcourir a priori <u>tout</u> le tableau, mais si je peux <u>m'échapper</u> avant, alors je m'échappe !

```
public int rangNoteReturn(int note) {
  for(int i = 0; i < notes.length; i++) {
     if (notes[i] == note) {
        return i;                           //  échappement!
     }
  }
  return -1;          // si aucun « return » n'a été exécuté !
}
```

• Dès qu'elle est invoquée, l'instruction « **`return r;`** » provoque l'abandon de la méthode en cours, avec le résultat **`r`** sous le bras…

• Si la méthode est sans résultat, on peut utiliser « **`return;`** » sans valeur de retour.

Recherche du maximum d’un tableau

• Par exemple d’un tableau d’entiers. Cherchons la meilleure note d’un étudiant :

```
public int meilleureNote() {
  int mNote = 0;
  for(int i = 0; i < notes.length; i++) {
    if (notes[i] > mNote) {
      mNote = notes[i];
    }

  }
  return mNote;
}
```

mNote = la meilleure note trouvée jusqu'à présent...

• Par exemple si **`notes`** est **`{12,5,8,15,9}`** ?

- Par exemple si **`notes`** est **`{-12,-5,-8,-15,-9}`** ?…
- Impossible ? Et dans des QCM à points négatifs :-)
- *Initialiser au premier élément* et boucler sur le reste !

```
public int meilleureNote() {
    int mNote = notes[0];
    for(int i = 1; i < notes.length; i++) {
      if (notes[i] > mNote) {
        mNote = notes[i];
      }

    }
    return mNote;
}
```

Un tableau en paramètre d'une méthode !

```
public void copierNotesDans(int[] tableau) {
  for(int i = 0; i < notes.length; i++) {
    tableau[i] = this.notes[i];
  }
}
```

- Ce qui est passé à la méthode, c'est une *référence* ou *pointeur* sur le tableau.

- Le contenu effectif du tableau pourra donc être modifié !

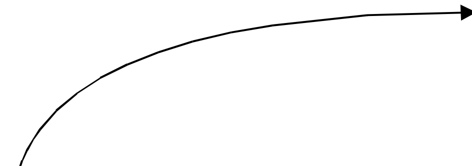


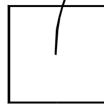
tableau 

tableau: int[]	
length	5
tableau[Ø]	5
tableau[1]	18
tableau[2]	16
tableau[3]	4
tableau[4]	20

• Bien entendu c'est à l'appelant d'avoir pris la précaution de passer une référence sur un tableau existant !

• Exemple typique d'utilisation dans une méthode d'instance de la classe **Etudiant** (cf p. 16) :

```
int[] t2 = new int[notes.length];
copierNotesDans(t2);
for (int i=0; i<t2.length; i++) {
  if (notes[i] != t2[i]) {
    System.out.println(″Il y a un pb !!!″);
  }
}
```

• Notez qu'il suffit en fait que la taille du tableau receveur (ici **t2**) soit au moins égale à celle du tableau **notes** !

• L'utilisation de tableaux non initialisés conduit souvent à l'erreur ***NullPointerException***...

Un tableau comme résultat d'une méthode

• Ex: une méthode qui rend une *copie* du tableau des notes d'un étudiant, mais avec des notes sur 10 au lieu de 20 :

```
public double[] notesSur10() {
    int n = notes.length;
    double[] res = new double[n];
    for(int i = 0; i<n; i++) {
      res[i] = notes[i] / 2.0;
    }
    return res;
}
```

Allocation d'un nouveau tableau !

Copie du tableau **notes** dans le tableau **res**

et non 2

On rend (une *référence* sur) le nouveau tableau

Exemple: calcul additif du triangle de Pascal

• Pour $0 \leq p \leq n$, on note **binomial(n,p)** le nombre de parties à p éléments d'un ensemble à n éléments (combinaisons de p éléments parmi n).

• Avec 10 cartes, il y a binomial(10,3) = 120 paquets de 3 cartes.

• Deux manières de calculer :

- par une formule directe : $\text{binomial}(n,p) = \dfrac{n!}{p!\,(n-p)!}$
- par une relation de récurrence :
  binomial(n,p) = binomial(n-1,p) + binomial(n-1,p-1)

• On se propose de calculer la ligne numéro n du triangle de Pascal, donc le tableau dont les éléments sont les entiers binomial(n,p) avec $0 \leq p \leq n$.

• Le résultat est donc un tableau d'entiers, de longueur n+1.

p →, n ↓

n \ p	0	1	2	3	4	5	6	7
0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0
2	1	2	1	0	0	0	0	0
3	1	3	3	1	0	0	0	0
4	1	4	6	4	1	0	0	0
5	1	5	10	10	5	1	0	0
6	1	6	15	20	15	6	1	0
7	1	7	21	**35**	35	21	7	1

n = 7

Chaque tableau est calculable *sur place* à partir du précédent !

sur place ⇔ sans tableau auxilliaire !

`binomial(7,3) = 35`

$$C_7^3 = 35$$

Etudiez ce code de très près !!!

```
public class Pascal {
   private int[] ligne;

   public Pascal(int numLigne) {
      ligne = new int[numLigne+1];
      ligne[0] = 1;
      for(int p=1; p<=numLigne; p++) {
        for(int n=p; n>=1; n--) {
          ligne[n] = ligne[n-1] + ligne[n];
        }
      }
    }

   public String toString() {     // affichage d'un tableau…
     ...
   }
}
```

```
Pascal ligne7 = new Pascal(7);

System.out.println(ligne7);
```

• Le langage Java permet aussi de construire des **tableaux à 2 dimensions**, ou *matrices*.

p →
n ↓

	0	1	2	3	4	5	6	7
0	1							
1	1	1						
2	1	2	1					
3	1	3	3	1				
4	1	4	6	4	1			
5	1	5	10	10	5	1		
6	1	6	15	20	15	6	1	
7	1	7	21	35	35	21	7	1

n = 7

`tab[4][2] == 6`

• L'élément situé en ligne **`i`** et colonne **`j`** se noterait alors :

`tab[i][j]`

• Mais peu importe puisque ceci est HORS-PROGRAMME !