LU2IN002 - Introduction à la programmation orientée-objet

Responsable de l'UE : Christophe Marsala
(email: Christophe.Marsala@lip6.fr)

Cours du lundi : Sabrina Tollari
(email: Sabrina.Tollari@lip6.fr)

(support réalisé à partir de ceux de Christophe Marsala et de Vincent Guigue)

SCIENCES SORBONNE UNIVERSITÉ

Cours 4 – 3 octobre 2022

Programme du jour

Mot clé `final`

Une variable final ne peut **pas être modifiée** après initialisation.

Exemple :

```
1 final int c=10;     // OK
2 c=15; // error: cannot assign a value to final variable c
```

△ On n'est pas obligé d'initialiser les variables `final` lors de la déclaration

```
11 final int d;      // OK pas initialisée
12 d=12;             // OK initialisation
13 d=16; // error: variable d might already have been assigned
```

Remarques :

- les attributs, les paramètres et les variables locales à une méthode peuvent être `final`
- on verra plus tard, qu'on peut écrire aussi des méthodes `final` et des classes `final`

Mot clé `final` : cas des variables d'instance

Cas particulier des variables d'instance `final`

Les variables d'instance `final` ne peuvent être initialisées que :

- lors de la déclaration
- ou dans le constructeur,

mais pas dans une méthode

```
1  public class MaClasse {
2     private final int a = 10 ;  // OK init déclaration
3     private final int b ;
4     private final int c ;
5     public MaClasse(int b) {
6        this.a = 15 ; // Faux a est déjà initialisée
7        this.b = b ;  // OK init constructeur
8     }
9     public void maMethode() {
10       this.c=24; // Faux c ne peut être initialisée
11                  // dans une méthode
12    }
13 }
```

Mot clé final : exemples d'utilisation

Exemples d'utilisation :

- un identifiant ne doit jamais être modifié
- une constante ne doit jamais être modifiée

```
1  public class Point {
2     public final String id;
3     public static final int MAX_VALUE=10;
4     private double x, y;
5     public Point(String id) {
6        this.id=id ;
7        x=Math.random()*MAX_VALUE;
8        y=Math.random()*MAX_VALUE;
9     }
10 }
```

En général, les attributs doivent être déclarés `private`

★ Pourquoi ici ces attributs ont été déclarés `public` ?

- Car comme ils sont déclarés `final`, ils ne peuvent pas être modifiés $\Longrightarrow$ pas de problème de sécurité des données
- Le client pourra connaître la valeur, mais pas la modifier

Remarque : le mot clé `static` dans la déclaration de la constante sera expliqué au cours 5

Plan du cours

Structure de données

1. Tableau à taille fixe
   - \+ Economie mémoire
   - \+ Rapidité d'accès
   - − Peu flexible (taille fixe !)

   En Java :
   - ♦ syntaxe assez similaire au langage C, sauf pour la réservation mémoire
   - ♦ un tableau a un comportement d'objet
   - ♦ tableau de type simple, mais aussi tableau d'objets
2. Tableau à taille variable
   - − Gourmand en mémoire
   - − (Un peu) moins rapide
   - \+ Très flexible

   En Java : on peut utiliser la classe `ArrayList` pour simuler des tableaux à taille variable d'objets.
   - △ Pour les types de base, il faut utiliser les classes enveloppes (`int`→`Integer`, `double`→`Double`,...)

Syntaxe des tableaux

- Déclaration d'une variable : *type* `[]` `nomVariable`

```
1 int[] tableau;
```

⚠ c'est la déclaration d'une variable, le tableau n'est pas créé

- Instanciation : `nomVariable = new` *type*`[`*taille*`];`

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

$\Longrightarrow$ Réservation de 2 cases mémoires de type `int`

- Accès à la case *i* (lecture ou écriture) : `nomVariable[i]`

```
3 tableau[0] = 1;
4 tableau[1] = 4;
5
6 int x = tableau[0];
```

- Accès à la longueur du tableau : `nomVariable.length`

⚠ un tableau a un comportement d'objet : il a un **attribut** `length`

```
7 System.out.println("Longueur : "+tableau.length);
```

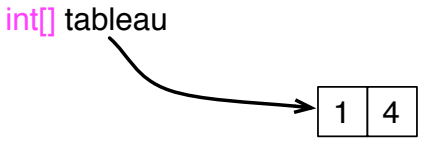
Tableau

tableau $\simeq$ **ensemble de variables**... facilement accessibles avec une boucle

```
1 int[] tableau = new int[2];
2 tableau[0] = 1;
3 tableau[1] = 4;
```

- `tableau` est une variable de type `int[]` (ie tableau d'entiers)
- `tableau[i]` : chaque case de `tableau` est de type `int`

Représentation mémoire



Attention : sur le diagramme mémoire, bien différentier variables et instances...

Variantes de syntaxe

Pour en même temps **réserver la mémoire et initialiser un tableau**, on peut aussi utiliser les syntaxes suivantes.

- Syntaxe simplifiée : `type[] maVar={value,value,...};`

```
1 boolean[] tableau={true,false,true};
```

⚠ `{value,value,...}` ne marche que lors de la déclaration

```
2 boolean[] tableau2; // OK déclaration sans initialisation
3 tableau2={true,false,true}; // Erreur à la compilation
```

- Syntaxe intermédiaire (marche partout) :

`new type[] {value,value,...}`

```
11 boolean[] tableau;
12 tableau = new boolean[]{true,false,true};
```


Tableaux et boucles

Code robuste = pas de duplication de l'information

Attention aux conditions de fin de boucles

```
1 int[] tab = {2, 3, 4, 5, 6};
```

Besoin de faire une boucle...

```
2 for(int i=0; i<5; i++) // INCORRECT dans le cadre de LU2IN002
3     ...

4 for(int i=0; i<tab.length; i++) // CORRECT
5     ...
```

A chaque fois que c'est possible, utiliser `tab.length` pour indiquer la taille du tableau.

Tableaux et boucles

Pour les tableaux, il existe aussi une boucle `for` sans indice

Syntaxe

```
for(type var : nomTableau)
    ...
```

`var` prend successivement toutes les valeurs des éléments du tableau

```
boolean[] tableau={true,false,true};
for(boolean b : tableau)
   System.out.println(b); // affiche chaque case du tableau
```

⚠ Pas d'indices : ne peut pas être utiliser avec tous les algorithmes

- Ne peut pas être utilisée quand on a besoin des indices
- Ne peut pas être utilisée quand on veut modifier le tableau (ici b est une variable locale)

Remarque : cette boucle `for` sans indice peut aussi être utilisée

- avec les tableaux d'objets
- avec certaines classes, dont la classe `ArrayList`

Tableau en variable d'instance

En général, quand on utilise un tableau en variable d'instance, il faut penser à réserver la mémoire dans le constructeur

```
1  public class MaClasse {
2    private int [] tab ; // déclaration d'une variable
3    public MaClasse(int n) {
4      tab=new int[n]; // réservation mémoire
5    }
```

Quand il y a plusieurs constructeurs, il faut faire attention que, dans tous les cas, la réservation mémoire pour le tableau soit effectuée

```
11 public class MaClasse {
12   private int [] tab ;
13   public static final int TAILLE_STANDARD=10;
14   public MaClasse(int n) {
15     tab=new int[n]; // réserve la mémoire
16   }
17   public MaClasse() {
18     this(TAILLE_STANDARD); // réserve la mémoire grâce à ligne 15
19   }
20   public MaClasse(int x, int y) {
21     this(2); // réserve la mémoire grâce à la ligne 15
22     tab[0]=x; tab[1]=y;
23   }
```


Tableau d'objets

Soit la classe `Point` (vue dans les cours précédent).
On veut faire un tableau d'objets `Point`.

- Déclaration d'une variable `tabP` de type `Point[]`

```
Point[] tabP;
```

⚠ C'est juste la déclaration d'une variable, le tableau n'existe pas encore (il n'est **pas instancié**)

- Instanciation du tableau (réservation des cases mémoires)

```
tabP = new Point[10];
```

$\Longrightarrow$ La variable `tabP` référence un tableau de 10 cases

⚠ **10 cases = 10 variables... mais aucun objet `Point`**

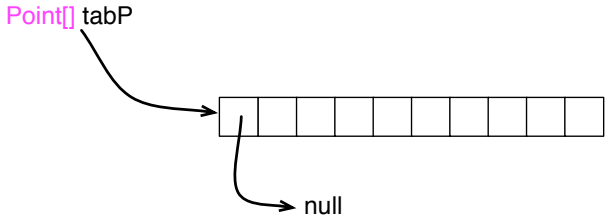


Tableau d'objets

Chaque case (=variable) peut/doit être initialisée avec un objet

- Initialisation d'une case

```
tabP[0] = new Point(1,2);
```

- Initialisation avec une boucle

```
for(int i=1;i<tabP.length;i++)
    tabP[i] = new Point(i,i);
```

- Affichage

```
for(Point pi : tabP)
    System.out.println(pi);
```

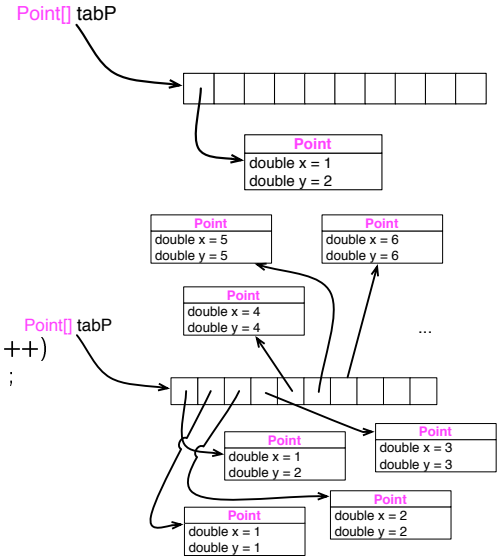


Tableau d'objets

Les cases se comportent vraiment comme des variables : on peut **jouer avec les références**

```
1 Point p = new Point(3,4);
2 Point[] tabP2 = new Point[3];
3
4 tabP2[0] = p;
5 tabP2[1] = new Point(4,5);
6 tabP2[2] = tabP2[0];
```

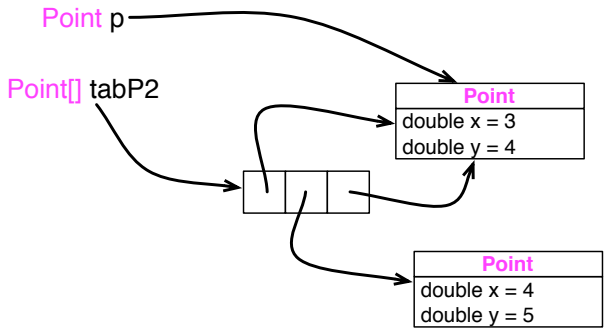


Tableau à deux dimensions

Comment gérer les matrices ?

Comme **des tableaux de tableaux**

- Déclaration des variables : `type[][]`

```
int[][] matrice;
```

- Instanciation

```
matrice = new int[2][3]; // 2 lignes, 3 colonnes
```

- Usage

```
matrice[0][0] = 0; matrice[0][1] = 1; matrice[0][2] = 2;
matrice[1][0] = 3; matrice[1][1] = 4; matrice[1][2] = 5;
```

- Syntaxe alternative d'instanciation/initialisation

```
int[][] matrice = {{0, 1, 2},{3, 4, 5}}
```

- Accès aux dimensions :

```
matrice.length // nb lignes
matrice[0].length // nb de colonnes de la première ligne
```


Tableau à deux dimensions : vision avancée

```
Point[][] matP = new Point[2][3];
```

est équivalent à la réservation d'un tableau de tableaux de `Point`

```
Point[][] matP2 = new Point[2][];
for(int i=0; i<matP2.length; i++)
  matP2[i] = new Point[3];
```

Création des objets `Point` (similaire pour `matP` et `matP2`)

```
for(int i=0; i<matP.length; i++)
   for(int j=0; j<matP[i].length; j++)
      matP[i][j] = new Point();
```

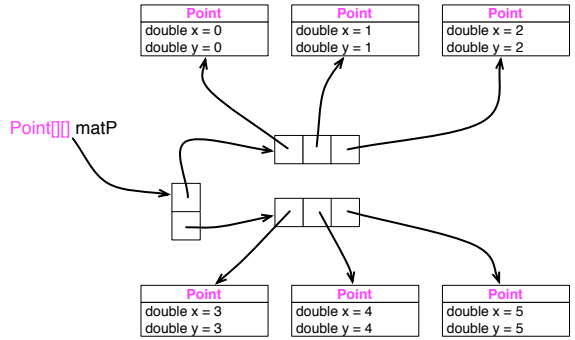
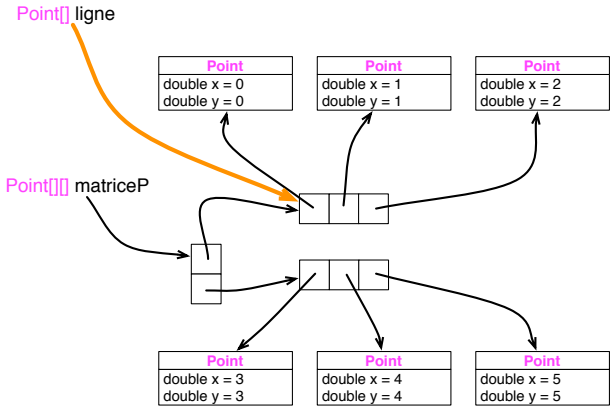


Tableau à deux dimensions : vision avancée (2)



- Possibilité de manipuler les lignes de la matrice de manière indépendante

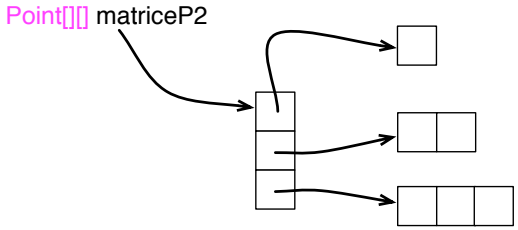
```
Point[][] matriceP = new Point[2][3];
Point[] ligne = matriceP[0];
// Affichage du premier point:
System.out.println(ligne[0]);
```


Matrice triangulaire

Pour éviter de réserver inutilement de le mémoire, on peut créer des tableaux de tableaux où les lignes ont des tailles différentes.

- La déclaration s'effectue généralement en 2 étapes :
  1. d'abord, on déclare un tableau de tableaux **sans préciser la deuxième dimension**
  2. puis, pour chaque ligne, on déclare un tableau à la bonne taille
- Par exemple, une matrice triangulaire

```
Point[][] matriceP2 = new Point[3][];  // Etape 1
for(int i=0; i<matriceP2.length; i++)
  matriceP2[i] = new Point[i+1];  // Etape 2
```



Dépassement de tableau

⚠ Tableau... ⇒ possibilité de dépasser dans un tableau

- Cas classique :
  - Mélange entre taille n et dernier indice du tableau ($n-1$)
  - Tentative d'accès à un index négatif
  - Erreur de boucle...
- Symptôme : `ArrayIndexOutOfBoundsException`
  - Echec lors de l'exécution du code (compilation OK)

```
1 Point[] tab = {new Point(), new Point()};
2 System.out.println(tab[2]); // pas de troisième case
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
at test.Point.main(Point.java:118)
```

- Attention aux `NullPointerException` : après instanciation d'un tableau, aucune instance n'est disponible :

```
1 Point[] tab = new Point[2];
2 System.out.println(tab[0].getX()); // => NullPointerException
```


Les packages

- Java est fourni avec un ensemble de classes déjà programmées
- Ces classes sont regroupées en fonction de leurs fonctionnalités dans des ensembles de classes appelés package

package ≃ bibliothèque de classes

Au démarrage, Java importe automatiquement le package `java.lang` qui contient notamment les classes :

- `String`, `Math`, `System`...

Pour utiliser une classe d'un autre package, il faut `importer la classe`. Par exemple :

- la classe `ArrayList` se trouve dans le package `java.util`
- pour utiliser la classe `ArrayList`, il faut écrire **au début de chaque fichier qui utilise cette classe** :

```
import java.util.ArrayList;
```


Plan du cours

La classe ArrayList

Usage dans 2 cas (imbriqués) :

- **Taille finale inconnue** lorsque l'on commence à utiliser le tableau (e.g. lecture d'un fichier...)
- **Taille variable** en cours d'utilisation (e.g. pile d'objets à traiter de taille variable)

- Syntaxe objet classique + approche générique (hors prog.) :
  - la variable sera de type : `ArrayList<`*type*`>`
  - *type* est forcément un objet (≠ type de base) : `Integer`, `Double`, `Point`...
- Même représentation mémoire que les tableaux de taille fixe

ArrayList : syntaxe détaillée

- Déclaration et création d'un objet de type `ArrayList<Point>`

```
ArrayList<Point> alp = new ArrayList<Point>();
```

- Ajout d'éléments

```
alp.add(new Point(1,2));
for(int i=0; i<9; i++)
    alp.add(new Point(i,i));
```

- Accés aux éléments

```
// Pour obtenir une référence sur le 1er élément,
// mais sans le supprimer de la liste
Point p1 = alp.get(0);
// Pour obtenir une référence sur le 1er élément
// ET le supprimer de la liste
Point p2 = alp.remove(0);
```

- Nombre d'éléments

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

Plus d'informations dans la javadoc (beaucoup d'autres méthodes disponibles) :
`https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html`

ArrayList : syntaxe détaillée

- Attention ! supprimer un élément avec `remove`, entraîne un décalage des indices

```
ArrayList<Double> tabArr = new ArrayList<Double>();
tabArr.add(11.);tabArr.add(12.);tabArr.add(13.);

System.out.println("Avant le remove size="+tabArr.size());
for (int index=0; index<tabArr.size(); index++)
    System.out.println("Element "+index+": "+tabArr.get(index));

tabArr.remove(1);   // On retire l'élément en 2ième position

System.out.println("Après le remove size="+tabArr.size());
for (int index=0; index<tabArr.size(); index++)
    System.out.println("Element "+index+": "+tabArr.get(index));
```

```
Avant le remove size=3
Element 0: 11.0
Element 1: 12.0
Element 2: 13.0
Après le remove size=2
Element 0: 11.0
Element 1: 13.0 // décalage : le 3ème élément
                // se retrouve en 2ième position
```


Fonctions avancées

Java est fourni avec un grand nombre de classes avec des fonctions utilitaires déjà programmées pour les tableaux, les `ArrayList` et autres classes qui gèrent un ensemble d'objets.

Pour la classe `ArrayList`

- Dans la classe `ArrayList` :

```
ArrayList<Integer> ali = new ArrayList<Integer>();
for (int i= 0; i<10; i++)
        ali.add((int)(Math.random()*10));
if (ali.contains(2))
        System.out.println("Valeur trouvée!");
```

- Dans la classe `Collections` du package `java.util`
  - Tris, min, max, mélange, renversement...

```
System.out.println(ali);
int min=Collections.min(ali);
System.out.println("Min="+min);
Collections.sort(ali);
System.out.println(ali);
Collections.reverse(ali);
System.out.println(ali);
```

Affichage :

```
[2, 3, 5, 9, 3, 6, 1, 3, 8, 3]
Min=1
// tri
[1, 2, 3, 3, 3, 3, 5, 6, 8, 9]
// inversion
[9, 8, 6, 5, 3, 3, 3, 3, 2, 1]
```


Fonctions avancées

Pour les tableaux

- La classe `Arrays` du package `java.util`
  - recherche, trie, affichage, copie, remplissage
- Exemple :

```
int [] tab={13,15,11,14,12};
int index=Arrays.binarySearch(tab, 14); // recherche l'index de 14
System.out.println("Index="+index);
System.out.println("tab="+Arrays.toString(tab));
Arrays.sort(tab); // tri du tableau
System.out.println("tab="+Arrays.toString(tab));
```

- Affichage :

```
Index=3
tab=[13, 15, 11, 14, 12]
tab=[11, 12, 13, 14, 15]
```