LU2IN002 - Introduction à la programmation orientée-objet

Christophe Marsala

SCIENCES SORBONNE UNIVERSITÉ

Cours 4 – 7 octobre 2022

Plan du cours

Le mot-clé final

Idée : protéger des variables

Donner une valeur une variable sans pouvoir la modifier ensuite.

- Exemple :

```
final int annee = 2022;  // création et initialisation
                         //  d'une variable non modifiable
annee = 2023; // ERREUR: on ne peut plus changer sa valeur !
annee++; // ERREUR: on ne peut plus changer sa valeur !
```

- Autre exemple :

```
final int annee;  // création d'une variable non modifiable
annee = 2022; // OK: première (et unique) initialisation
annee = 2023; // ERREUR: on ne peut plus changer sa valeur !
annee++; // ERREUR: on ne peut plus changer sa valeur !
```

- Remarque :
  - possible pour les attributs et variables locables

Utilisation de final pour les attributs

Idée : protéger ses objets... et ses programmes

Initialiser les valeurs des attributs sans pouvoir les modifier ensuite.

```
public class Point{
  public final double x,y;
  public Point(double x, double y){
    this.x = x; this.y = y;  // possible que dans un constructeur
  }
  // interdiction de modifier x, y dans la suite
  // (et chez le client)
}
```

- Interdiction de modifier `x` et `y` dans les méthodes
- Modification d'un `Point` $\Longrightarrow$ création d'une nouvelle instance
- De tels attributs peuvent être `public` (car non modifiable)
- Sécurité lorsqu'un objet est passé en argument de méthode

Plan du cours

Structure de données

La **structure de base** de la programmation impérative : disponible sur **les types de base et sur les objets**.

1. Tableau à taille fixe (syntaxe proche de celle du langage C)
   - \+ Economie mémoire
   - \+ Rapidité d'accès
   - – Peu flexible (taille fixe !)
2. Tableau à taille variable (classe ArrayList)
   - – Gourmand en mémoire
   - – (Un peu) moins rapide
   - \+ Très flexible

[Taille fixe] un tableau est un objet

- Déclaration : *type* [] nomVariable
- Instanciation : nomVariable = new *type* [*taille*]
- Accès à la case *i* (lecture ou écriture) : nomVariable[i]
- Accès à la longueur du tableau : nomVariable.length

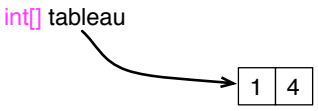
```
1  class EssaiTableau {
2    public static void main(String[] argv) {
3      int[] tableau; // déclaration d'un handler de tableau
4
5      tableau = new int[2]; // instanciation: tableau de 2 entiers
6      tableau[0] = 1; // utilisation (écriture)
7      tableau[1] = 4;
8
9      int i = tableau[0]; // utilisation en lecture
10     // Important : accès à la longueur du tableau
11     // -> nombre de valeurs qu'il peut contenir
12     System.out.println("Longueur : "+tableau.length);
13   }
14 }
```

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

Représentation mémoire

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

tableau = **ensemble de variables**...
... facilement accessibles avec une boucle.



Attention : bien différentier variables et instances...

- instanciation d'un tableau = création de variables
- créer les instances dans un second temps
- ⇒ passage aux objets (un peu) piégeux

Création et initialisation : variantes

- Créer et initialiser un tableau
  - si les valeurs sont fixées et connues dès le départ
- Syntaxe classique (marche partout) :

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

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

- Syntaxe simplifiée : {value,value,...}

  ⚠ Ne marche que sur la ligne de déclaration

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


Tableaux et boucles (1)

- Parcourir un tableau (lecture, écriture) : boucle for
- Attention ! ne pas déborder !

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

Taille du tableau : length

La taille du tableau est définie lors de la **création**
Utiliser l'attribut length pour y faire référence

```
2 // OK: modifier le tableau = modifier la boucle !
3 for(int i=0; i<tab.length; i++)
4     System.out.println(tab[i]);
```

Code robuste = pas de duplication de l'information

Toujours se baser sur length !

```
5 for(int i=0; i<5; i++) // INCORRECT en LU2IN002
6     System.out.println(tab[i]);
```


Tableaux et boucles (2)

- Parcours des éléments du tableau (sans référence d'indice) :
  for(*type* val : nomTableau) ...
  val prend successivement toutes les valeurs des éléments du tableau

```
1 for(boolean b: tableau) // affichage de tous les éléments
2   System.out.println(b);
```

⚠ Pas d'utilisation d'indices :

- utilisation intéressante, ou pas, en fonction des algorithmes
- usage possible avec les tableaux et avec tout itérable

Tableau d'objets

Soit la classe Point (vue dans les cours précédent)

Déclaration d'une variable tabP de type Point[] (tableau de points)

⚠ Le tableau n'existe pas ! (il n'est **pas instancié**)

La variable tabP référence un tableau de 10 cases

⚠ **10 cases = 10 variables**...
... 0 instances de Point !

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 :
//      création d'un tableau de tableau
//      de taille 2
Point[][] matP2 = new Point[2][];
//  création de 2 tableaux de 3 cases
for(int i=0; i<matP2.length; i++)
  matP2[i] = new Point[3];
```

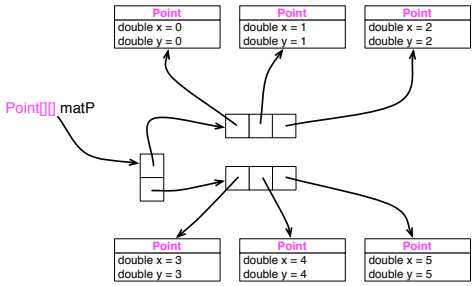
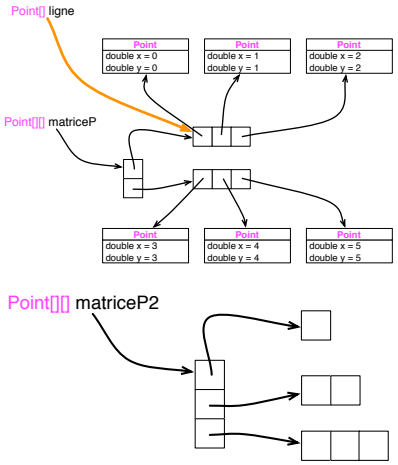


Tableau à deux dimensions : vision avancée (2)



- Possibilité de manipuler les lignes de la matrice de manière

[Tableau dynamique] 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)

- Objet JAVA à déclarer avant utilisation :

```
import java.util.ArrayList; // en entête
```

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

ArrayList : syntaxe détaillée

- **Exemple** sur un tableau de `Point`
- **Méthodes principales** : constructeur, add, get, remove, size
- Plus d'informations sur la javadoc (beaucoup d'autres méthodes disponibles) :
  `https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html`

```
// Construction comme un objet classique
ArrayList<Point> tabArr = new ArrayList<Point>();

tabArr.add(new Point(1,2)); // ajout
for(int i=0; i<9; i++)
    tabArr.add(new Point(i,i));

// accesseur sur le premier élément (index = 0)
Point p = tabArr.get(0);
// accesseur sur le deuxième élément (index = 1)
Point p = tabArr.get(1);
tabArr.remove(0); // suppression du premier élément

// accesseur sur la taille courante
System.out.println(tabArr.size());
```


ArrayList : syntaxe détaillée

- Attention ! le retrait entraîne un décalage

```
ArrayList<Double> tabArr = new ArrayList<Double>();
tabArr.add(new Double(4.2));
for (int i=0; i<5; i++)
    tabArr.add(new Double(11.38*i));

System.out.println("Taille : "+tabArr.size());

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

System.out.println("Valeur de l'element 5: "+ tabArr.get(5));
// ---> "Valeur de l'element 5: 45.52''

tabArr.remove(1);   // On retire l'élément en position 1
System.out.println("Après le remove");
for (int index=0; index<tabArr.size(); index++)
    System.out.println("Element "+index+" : "+tabArr.get(index));

System.out.println("Valeur de l'element 5: "+ tabArr.get(5));
// ERREUR: java.lang.IndexOutOfBoundsException: Index: 5, Size: 5
```

Dépassement de tableau [fixe ou dynamique]

⚠ Tableau... ⇒ erreur lors d'un dépassement

- 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]);
```

```
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
```


Fonctions avancées

- `ArrayList`
  - Dans la classe même :

```
1 ArrayList<Double> arr = new ArrayList<Double>();
2 for (int i= 0; i<10; i++)
3     arr.add((double) i);
4 if (arr.contains(2.))
5         System.out.println("Valeur trouvée!");
```

  - Dans la classe Collections
    - Tris, min, max, mélange, renversement...
- Tableau
  - Dans la classe Arrays : recherche, copie, remplissage

```
1 int[] tab = {3, 5, 1, 8, 9};
2 System.out.println(Arrays.binarySearch(tab, 1));
```