

# Introduction à la Programmation Orientée Objet

## 3 - Structures de données

**Valentin Honoré**

valentin.honore@ensiie.fr

FISA 1A

- 1 Les tableaux en Java
- 2 Les structures de données : objets et classes
- 3 Manipulation de tuples en Java
- 4 Tuples versus tableaux en Java
- 5 Objets et références
- 6 Objets et invocation de méthodes
- 7 Les *packages*

- ▶ Comme en C, un tableau est une structure de données qui contient plusieurs éléments du même type

Un tableau de 6 entiers

1	17	4	7	2	13
---	----	---	---	---	----

# Allocation d'un tableau (1/2)

- ▶ Un tableau doit être alloué dans la mémoire avec `new type[n]`  
→ Allocation d'un tableau de  $n$  éléments ayant pour type `type`
- ▶ Par exemple : `new int[6]`

Alloue un tableau de 6 entiers

0	0	0	0	0	0
---	---	---	---	---	---

## Allocation d'un tableau (2/2)

- ▶ L'opérateur `new` renvoie une référence vers un tableau (une référence est un identifiant unique d'une structure de données)
- ▶ Par exemple, `new int[6]` renvoie une référence vers ce tableau :

0	0	0	0	0	0
---	---	---	---	---	---

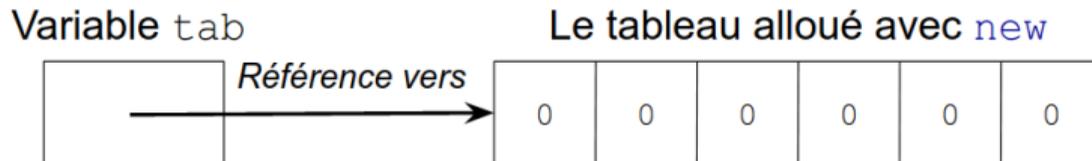
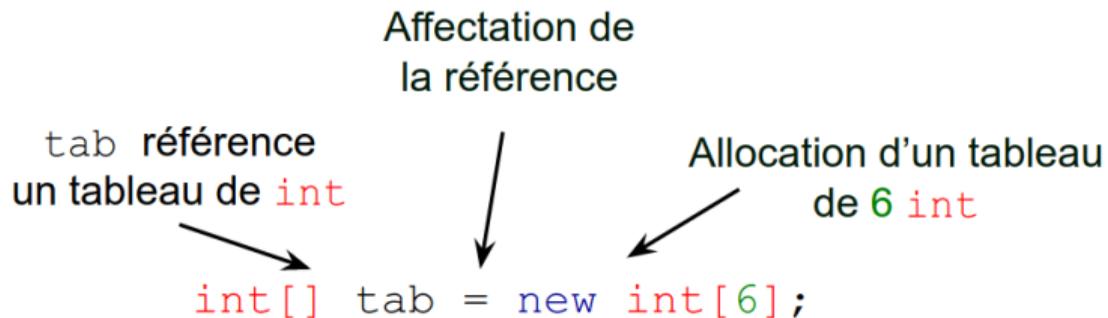
Note : Java met à 0 chaque élément lors d'une allocation

- ▶ **il n'existe pas de variable de type tableau en Java !**
- ▶ En revanche, on peut déclarer une variable de type référence vers un tableau :  
`type [] var ;`

→ var est une variable de type **référence vers un tableau**

→ var contient des éléments de type `type`

# Exemple de déclaration



- ▶ On peut aussi allouer un tableau et l'initialiser avec

```
type[] tab = {x1, ..., xn};
```

- ▶ Par exemple :

```
double[] tab = {2.3, 17.0, 3.14, 8.83, 7.26};
```

- ▶ En détails, le programme va
  - Allouer le tableau (comme avec `new`) puis initialiser les éléments
  - Renvoyer une référence vers le tableau

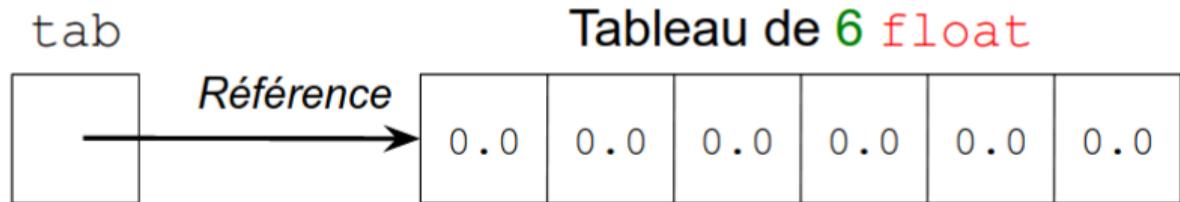
- ▶ Accès à la taille du tableau avec `tab.length`
- ▶ Accès à un élément avec `tab[indice]` comme en C  
Exemple : `tab[i] = tab[j] * 2;`

**Attention :** Comme en C, les éléments sont indexés à partir de 0

- ▶ Un accès en dehors des bornes du tableau provoque une erreur à l'exécution (`ArrayOutOfBoundsException`)

## Exemple d'accès à un tableau (1/6)

```
public static void main(String[] args) {  
    float[] tab = new float[6];  
    for(int i=0; i<tab.length; i++) {  
        tab[i] = i + 2;  
    }  
}
```



## Exemple d'accès à un tableau (2/6)

```
public static void main(String[] args) {  
    float[] tab = new float[6];  
    for (int i=0; i<tab.length; i++) {  
        tab[i] = i + 2;  
    }  
}
```



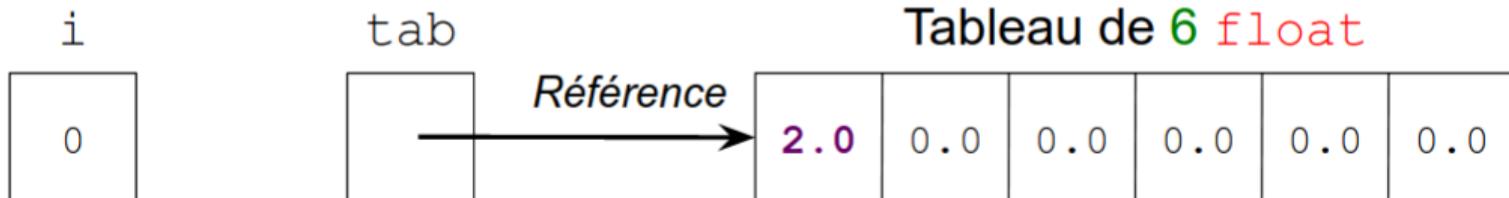
## Exemple d'accès à un tableau (3/6)

```
public static void main(String[] args) {  
    float[] tab = new float[6];  
    → for(int i=0; i<tab.length; i++) {  
        tab[i] = i + 2;  
    }  
}
```



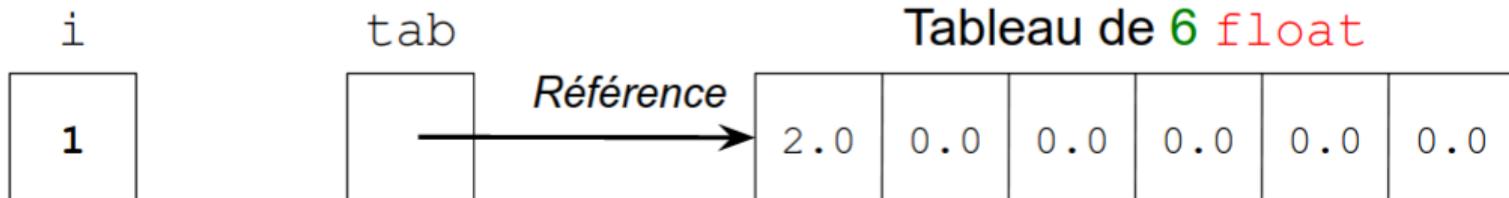
## Exemple d'accès à un tableau (4/6)

```
public static void main(String[] args) {  
    float[] tab = new float[6];  
    for(int i=0; i<tab.length; i++) {  
        tab[i] = i + 2;  
    }  
}
```



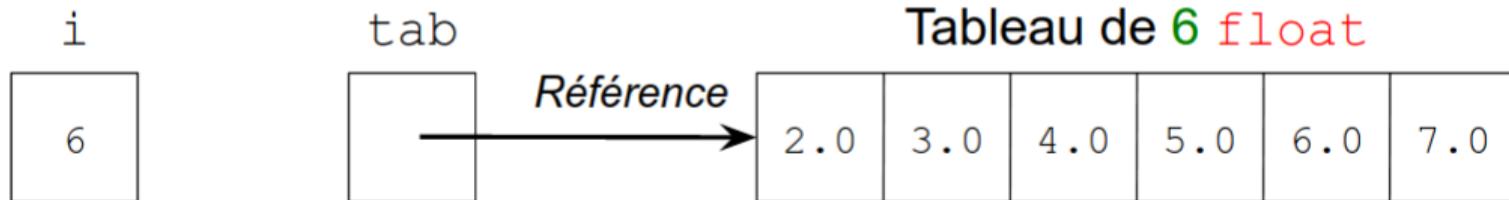
## Exemple d'accès à un tableau (5/6)

```
public static void main(String[] args) {  
    float[] tab = new float[6];  
    → for(int i=0; i<tab.length; i++)  
        tab[i] = i + 2;  
}
```



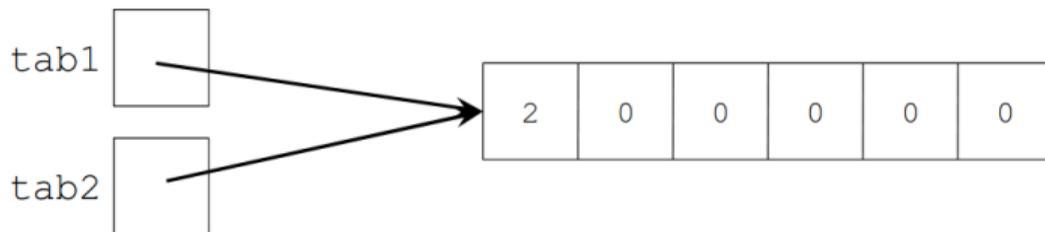
## Exemple d'accès à un tableau : fin de boucle (6/6)

```
public static void main(String[] args) {  
    float[] tab = new float[6];  
    for(int i=0; i<tab.length; i++) {  
        tab[i] = i + 2;  
    }  
}
```



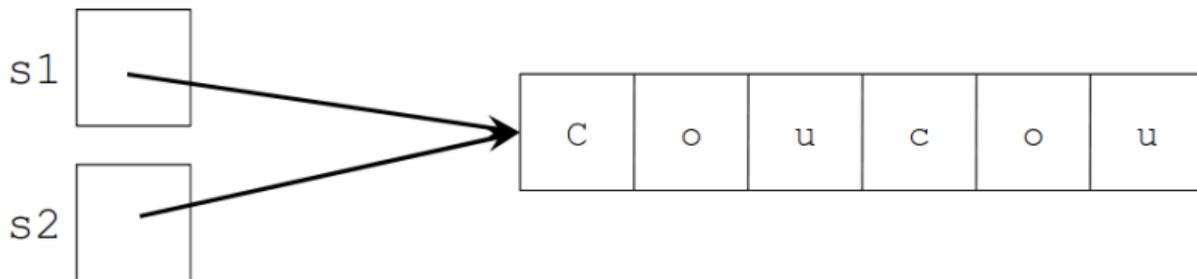
```
public static void main(String[] args) {  
    byte[] tab1 = new byte[6];  
    byte[] tab2 = tab1;  
    tab2[0] = 2;  
    System.out.println("tab1: " + tab1[0]);  
} /* affiche t
```

- ▶ Dans l'exemple précédent, tab1 et tab2 sont deux variables différentes, mais elles référencent le même tableau



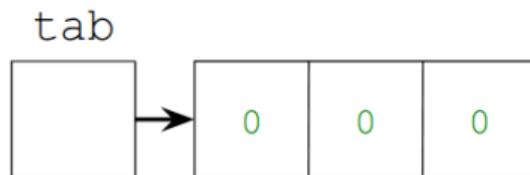
- ▶ Comme pour les tableaux, **il n'existe en fait pas de variable de type String en Java !**
- ▶ En revanche, `String` déclare une variable référençant une zone mémoire typée avec le type `String`
- ▶ Exemple :

```
String s1 = "Coucou"; String s2 = s1;
```



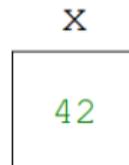
- ▶ Variable de type référence : contient une référence vers une structure de données  
→ **Tableaux** et **String**

```
int[] tab = new int[3];
```



- ▶ Variable de type primitif : contient une valeur  
→ boolean, byte, short, char, int, long, float et double

```
int x = 42;
```



- ▶ Dans `public static void main(String[] args)`,  
`args` est une **référence vers un tableau de chaînes de caractères** correspondant aux arguments du programme
  - Si aucun argument : `args.length` vaut 0
  - Sinon, `args[0]` est le premier argument, `args[1]` le second etc.

```
class CmdLine {  
    public static void main(String[] args) {  
        for(int i=0; i<args.length; i++) {  
            System.out.println(  
                "args[" + i + "]: " + args[i]);  
        }  
    }  
}
```

- ▶ Allocation d'un tableau avec  
`new type [n]`
- ▶ Déclaration d'une variable référençant un tableau avec  
`type [] var`
- ▶ Accès à un élément avec  
`var[indice]`
- ▶ L'argument du `main` est le tableau des arguments du programme

- 1 Les tableaux en Java
- 2 Les structures de données : objets et classes
- 3 Manipulation de tuples en Java
- 4 Tuples versus tableaux en Java
- 5 Objets et références
- 6 Objets et invocation de méthodes
- 7 Les *packages*

- ▶ Structure de données = regroupement de données
  - Permet de lier entre elles des données
  - Simplifie le traitement de ces données
  
- ▶ Exemple : une structure de données "Personnage" regroupant :
  - une image (l'apparence du personnage),
  - une position,
  - un nombre de points de vie...
  
- ▶ Exemple : une structure de données "ListePersonnages" regroupant :
  - un ensemble de personnages

- ▶ Le **tableau** (vu juste avant)
  - Regroupe un nombre fini d'éléments **homogènes**
  - Les éléments sont indexés par un **indice**
  
- ▶ Le **tuple** (*aussi parfois appelé enregistrement ou structure*)
  - Regroupe un nombre fini d'éléments **hétérogènes**
  - Les éléments sont indexés par un **symbole**
  - Les éléments s'appellent des **champs** ou **attributs**

En Java

- ▶ Une structure de données (tuple ou tableau) s'appelle un **objet**
- ▶ Un **objet** possède un **type**
- ▶ Le **type d'un objet** s'appelle une **classe**
- ▶ Si la classe d'un objet  $o$  est  $C$ , alors on dit que  **$o$  est une instance de  $C$**

Un objet est donc une variable qui doit être déclarée avec un type... que l'on appelle **classe**

- ▶ Type complexe (en opposition aux typages primitifs : entiers etc)
- ▶ Regroupe un ensemble de données (de type primitif ou objet !) que l'on appelle **attributs**
  - chaque attribut a un nom, un type, une valeur initiale (*facultatif*)
- ▶ Regroupe un ensemble de **méthodes** (= fonctions) de traitement :
  - 1 de ses attributs
  - 2 de données extérieures
- ▶ Principe **d'encapsulation** de données
- ▶ **Bilan** : classes représentant les objets du système + méthodes + classe exécutable

# Exemple illustratif

```
class Carre {  
  
    /* Encapsulation de 3 entiers */  
    int cote;  
    int origine_x ;  
    int origine_y ;  
  
    /* Une methode permettant de deplacer un objet.  
    * 'this' : acces direct aux donnees encapsulees */  
    void deplace(int x, int y) {  
        this.origine_x = this.origine_x + x ;  
        this.origine_y = this.origine_y + y ;  
    }  
  
    /* Une autre methode definie dans la classe */  
    int surface() {  
        return this.cote * this.cote ;  
    }  
  
}
```

En programmation objet, le concepteur du programme doit déterminer

- ▶ les objets et données appartenant à chaque objet
- ▶ les droits d'accès qu'ont les autres objets à ces données

Encapsulation permet de cacher ou non des données entre objets du programme

Une donnée peut être en accès **public** ou **privé**

On verra plus tard un descriptif plus détaillé des droits d'accès

- 1 Les tableaux en Java
- 2 Les structures de données : objets et classes
- 3 Manipulation de tuples en Java**
- 4 Tuples versus tableaux en Java
- 5 Objets et références
- 6 Objets et invocation de méthodes
- 7 Les *packages*

# Deux étapes pour créer un tuple

- ▶ Étape 1 : définition de la classe d'un tuple (i.e., de son type)
  - Donne une énumération des champs du tuple
  - Utilisation du mot clé `class` suivi d'un identifiant de `type`

```
class Perso {  
    int pointsVie;  
    int x;  
    int y;  
}
```

- ▶ Étape 2 : création d'une instance de la classe avec `new`

```
Perso bilbon = new Perso();
```

`bilbon` référence une **instance** de la classe `Perso`

```
class Perso {  
    int pointsVie;  
    int x;  
    int y;  
}
```

```
class MonProg {  
    public static void main(String[] a) {  
        Perso bilbon = new Perso();  
        bilbon.pointsVie = 10;  
        bilbon.x = 0;  
        bilbon.y = 0;  
        Perso sauron = new Perso();  
        sauron.pointsVie = 1000;  
        sauron.x = 666;  
        sauron.y = 666;  
    }  
}
```

# Ne confondez pas classe et instance! (1/2)

- ▶ Les objets utilisés dans les programmes sont des représentations dynamiques créées à partir du modèle donné par la classe. Chaque **instance** d'une classe :
  - se conforme à la description que la classe fournit
  - possède une valeur propre pour chaque attribut (qui caractérisent l'état de l'objet)
  - peut se voir appliquer toute méthode définie dans la classe
  - est référencée par une variable
  
- ▶ Une classe peut évidemment être **instanciée** plusieurs fois !

# Ne confondez pas classe et instance! (2/2)

Une **classe** est une sorte de **moule**

<b>Perso</b>
<code>pointsVie: int</code> <code>x: int</code> <code>y: int</code>

Qui permet de créer des **instances** de même type

<b><u>bilbon: Perso</u></b>
<code>pointsVie: int = 10</code> <code>x: int = 0</code> <code>y: int = 0</code>

<b><u>sauron: Perso</u></b>
<code>pointsVie: int = 1000</code> <code>x: int = 666</code> <code>y: int = 666</code>

- ▶ Quand on code en Java, on utilise les conventions suivantes :
  - Les noms de classes des tuples commencent par une majuscule
  - Les variables et champs commencent par une minuscule
  - Les méthodes commencent par une minuscule
    - Visuellement, si on voit un symbole commençant par une majuscule, on sait qu'on parle d'une classe
- ▶ On ne définit qu'une et une seule classe par fichier source
- ▶ Le fichier source définissant la classe X s'appelle X.java

- 1 Les tableaux en Java
- 2 Les structures de données : objets et classes
- 3 Manipulation de tuples en Java
- 4 Tuples versus tableaux en Java**
- 5 Objets et références
- 6 Objets et invocation de méthodes
- 7 Les *packages*

- ▶ La **classe d'un tuple** possède un nom librement défini  
→ Mot clé **class** suivi du **nom** et de l'énumération des champs

```
class Perso { int pointsVie; int x; int y; }
```

└───┘



Nom de la classe

- ▶ La **classe** d'un tableau possède un nom imposé  
→ Type des éléments suivi du symbole [ ]

```
int[]
```

└───┘



Nom de la classe

- ▶ Un objet est alloué avec le mot clé `new` suivi de la `classe`, suivi de parenthèses dans le cas des tuples, mais pas des tableaux
  - `new` renvoie une référence vers l'objet alloué
  
- ▶ Par exemple
  - `new Perso()` → alloue une instance de la classe `Perso`
  - `new int[5]` → alloue un tableau de 5 `int`

- ▶ Accès à un champ d'un tuple :  
variable suivie d'un point et du nom du champ

```
sauron.pointsVie = 1000;
```

- ▶ Accès à élément d'un tableau :  
variable suivie d'un indice entre crochets

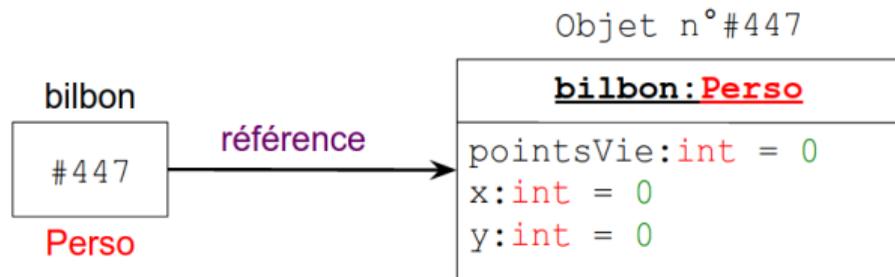
```
tab[3] = 42;
```

- 1 Les tableaux en Java
- 2 Les structures de données : objets et classes
- 3 Manipulation de tuples en Java
- 4 Tuples versus tableaux en Java
- 5 Objets et références**
- 6 Objets et invocation de méthodes
- 7 Les *packages*

# Objets et références (1/3)

- ▶ Java définit deux entités distinctes
  - Un objet est une structure de données en mémoire
  - Une référence est **un identifiant unique** d'un objet
- ▶ Une référence contient **l'adresse** d'un objet en mémoire
  - **Comme un pointeur**, contient l'adresse d'une structure
  - **Contrairement aux pointeurs**, la **seule opération autorisée sur les références** est l'affectation d'une référence de même type
- ▶ **Perso** p déclare une **référence** vers un objet de type **Perso**

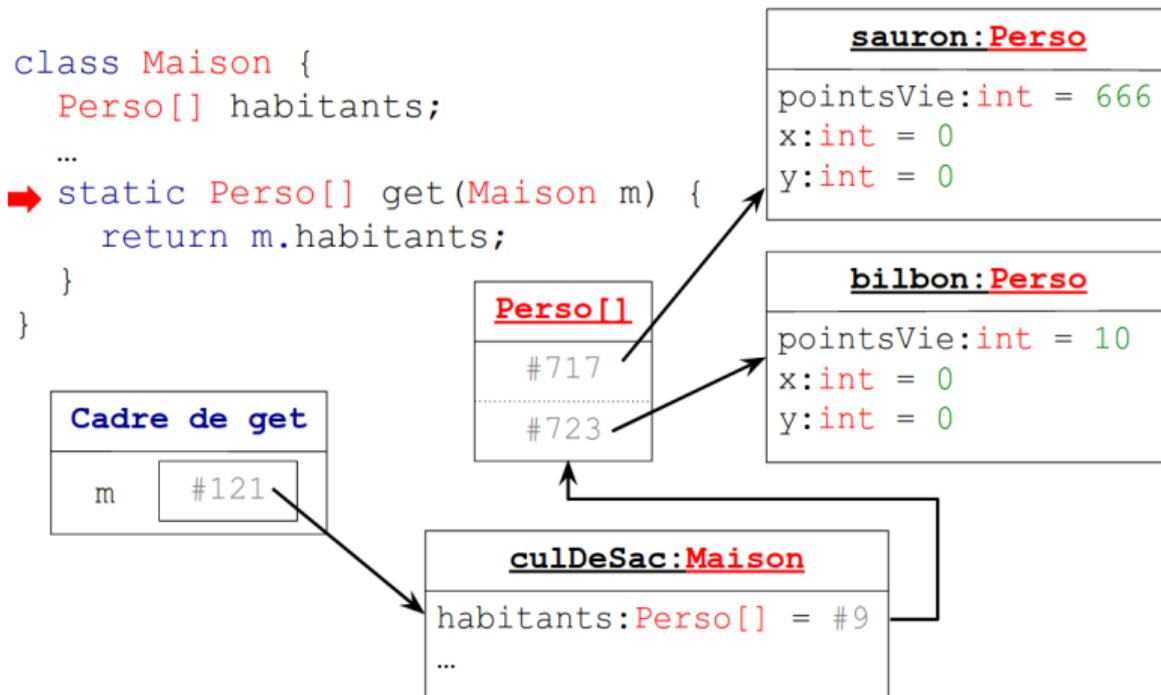
```
Perso bilbon = new Perso();
```



- ▶ Java définit deux entités distinctes
  - Un objet est une structure de données en mémoire
  - Une référence est **un identifiant unique** d'un objet
  
- ▶ Une référence contient **l'adresse** d'un objet en mémoire
  - **Comme un pointeur**, contient l'adresse d'une structure
  - **Contrairement aux pointeurs**, la **seule opération autorisée sur les références** est l'affectation d'une référence de même type
  
- ▶ `Perso p` déclare une **référence** vers un objet de type `Perso`
  
- ▶ De la même façon, `int [] tab` déclare une **référence** vers un objet de type `int []`

- ▶ Java définit deux entités distinctes
  - Un objet est une structure de données en mémoire
  - Une référence est **un identifiant unique** d'un objet
- ▶ Une référence contient **l'adresse** d'un objet en mémoire
  - **Comme un pointeur**, contient l'adresse d'une structure
  - **Contrairement aux pointeurs**, la **seule opération autorisée sur les références** est l'affectation d'une référence de même type
- ▶ `Perso p` déclare une **référence** vers un objet de type `Perso`
- ▶ De la même façon, `int [] tab` déclare une **référence** vers un objet de type `int []`
- ▶ Et `Perso [] tab` déclare donc une **référence** vers un tableau dans lequel chaque élément est une **référence** vers un `Perso`

# Java ne manipule que des références !



(on va voir plus tard ce que veut dire `static`)

- ▶ **null** : valeur littérale indiquant qu'aucun objet n'est référencé

```
Maison m = new Maison();
Perso bilbon = new Perso();
m.proprio = null; /* pas encore de proprietaire */

if(m.proprio == null)
    m.proprio = bilbon;
```

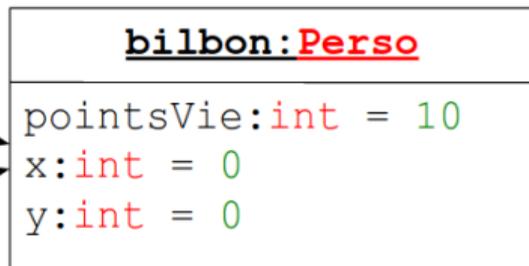
- ▶ Par défaut, les champs (resp. éléments) de type références d'un tuple (resp. tableau) sont initialisés à **null**

- 1 Les tableaux en Java
- 2 Les structures de données : objets et classes
- 3 Manipulation de tuples en Java
- 4 Tuples versus tableaux en Java
- 5 Objets et références
- 6 Objets et invocation de méthodes**
- 7 Les *packages*

- ▶ On dit que les objets sont passés par référence en Java

```
static void init(Perso p) {  
    p.pointsVie = 10;  
}
```

```
static void f() {  
    Perso bilbon = new Perso();  
    init(bilbon)  
    System.out.println(" ⇒ " + bilbon.pointsVie);  
    // affiche 10 car init reçoit une référence vers bilbon  
}
```



# Invocation inter-classe de méthode (1/2)

► Le mot clé `class` a deux rôles différents en Java

- Comme espace pour définir des classes définissant des tuples
- Comme espace pour définir des méthodes de classe (avec le mot clé `static`)

```
class Perso {  
    int pointsVie;  
    int x;  
    int y;  
}
```

```
class MonProg {  
    static void maFonction(int x) {  
        ...  
    }  
}
```

► On peut bien sûr combiner les deux rôles

```
class Perso { int pv; static void maFonc() { ... } }
```

## Invocation inter-classe de méthode (2/2)

- ▶ Par défaut, Java résout un appel de méthode dans la classe
  - Pour appeler une méthode d'une autre classe :  
préfixer le nom de la méthode avec la classe suivi d'un point

```
class MonProg {  
    static void maFonction(int x) {  
        Perso bilbon = new Perso();  
        Perso.display(bilbon);  
    }  
}
```

```
class Perso {  
    int pointsVie;  
  
    static void display(Perso p) {  
        ...  
    }  
}
```

- 1 Les tableaux en Java
- 2 Les structures de données : objets et classes
- 3 Manipulation de tuples en Java
- 4 Tuples versus tableaux en Java
- 5 Objets et références
- 6 Objets et invocation de méthodes
- 7 Les *packages*

- ▶ Un grand nombre de classes fournies par Java SE
  - Implémentent des structures de données et traitements génériques
  - Forment l'API (*Application Programmer Interface*) du langage
  - <https://docs.oracle.com/en/java/javase/21/docs/api/index.html>

Package	Description
java.awt	Interfaces et classes graphiques
java.io	Entrées/sorties
java.lang	Classes de base (importé par défaut)
java.util	Classes "boîte à outils"

Table – Quelques packages couramment utilisés

► Importer la classe ou le package dans lequel elle est définie

- une seule classe

```
import java.util.Date ;
```

- toutes les classes (même celles inutilisées)

```
import java.util.* ;
```

```
import java.util.Date ;

public class AfficherDate {
    public static void main(String[] args) {
        Date today = new Date();
        System.out.println("Nous sommes le " +
            today.toString());
    }
}
```

- ▶ Préciser avant la définition de votre classe le package auquel elle appartient
  - Exemple :

```
package ensiie.ipoo ;

import java.util.Date ;

public class AfficherDate {
    ...
}
```

- ▶ **Attention** : le chemin d'accès au fichier AfficherDate.java doit correspondre au nom de son package.
  - Celui-ci doit donc être dans ensiie/ipoo/AfficherDate.java
  - **ET** être accessible à partir des chemins d'accès à la compilation (cf Cours 2)

- ▶ Déclaration d'une classe définissant un tuple avec

```
class Nom { typeun champsun; typedeux champsdeux; ... }
```

- ▶ Allocation d'un objet avec l'opérateur `new`  
`new Nom()` si tuple ou `new type[n]` si tableau
- ▶ En Java, il n'existe que des types références, pas de type objet
- ▶ Lors d'un appel de méthode, un objet est passé par référence