



Les structures de données

CSC 3101

Algorithmique et langage de programmation

Gaël Thomas





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

Les structures de données

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

Deux familles de structures de données

- Le tableau (vu au CI2)
 - Regroupe un nombre fini d'éléments **homogènes**
 - Les éléments sont indexés par un **numéro**

- Le tuple (vu dans ce CI)
(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**



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

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`

Accès aux champs d'un tuple avec « . »

```
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 !

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

| Perso |
|---|
| <code>pointsVie: int</code> <code>x: int</code> <code>y: int</code> |

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

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

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

Conventions de codage (1/2)

- 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

Conventions de codage (2/2)

- On ne définit qu'une et une seule classe par fichier source (sauf pour les classes internes et anonymes, voir CI8)
- Le fichier source définissant la classe X s'appelle X.java



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

Tuples versus tableaux : nommage

- 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

Tuples versus tableaux : allocation

- 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`

Tuples versus tableaux : accès

- 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 structures de données : objets et classes
2. Manipulation de tuples en Java
3. Tuples versus tableaux en Java
4. Objets et références
5. Objets et invocation de méthodes

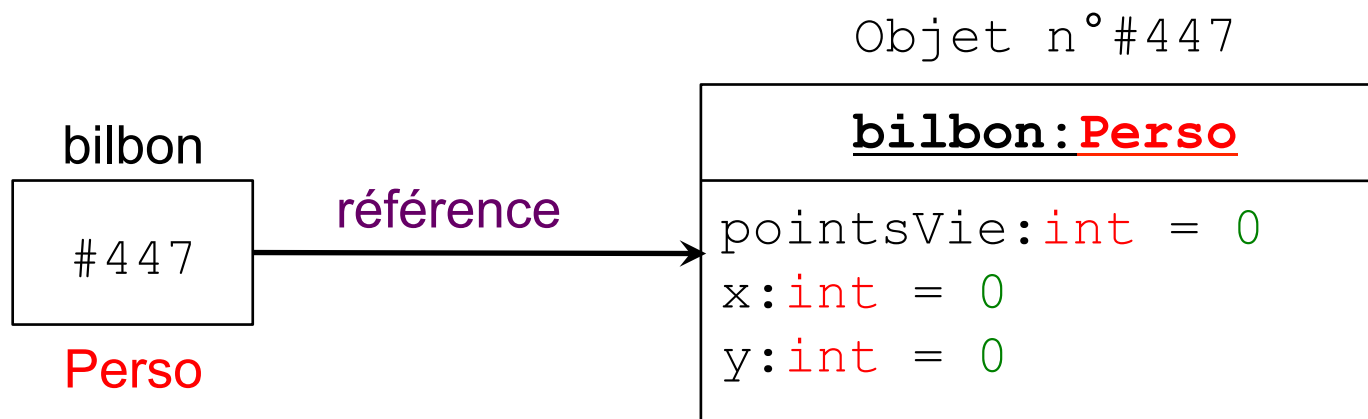
Objets et références (1/3)

■ Java définit deux entités distinctes

- Un objet est une structure de données en mémoire
- Une référence est **un identifiant unique** d'un objet

■ `Perso p` déclare une **référence** vers un objet de type `Perso`

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



Objets et références (2/3)

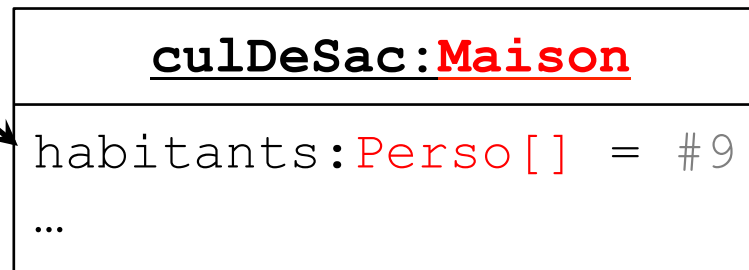
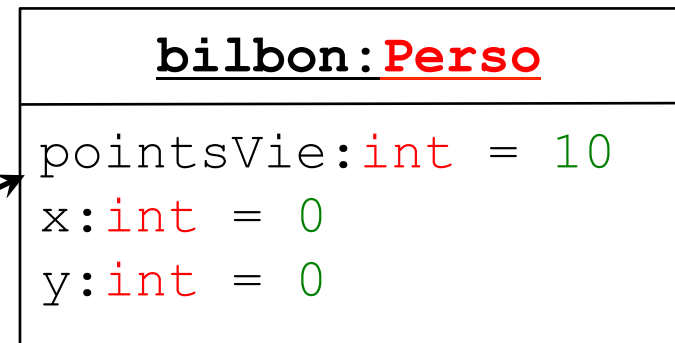
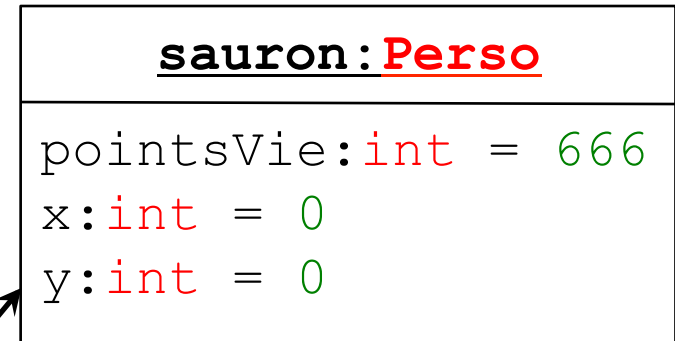
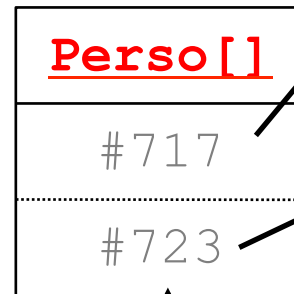
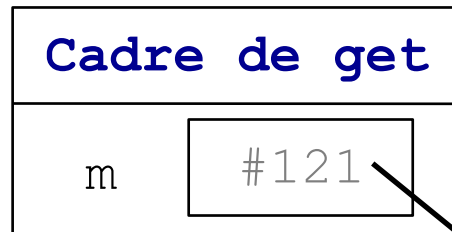
- Java définit deux entités distinctes
 - Un objet est une structure de données en mémoire
 - Une référence est **un identifiant unique** d'un objet
- `Perso p` déclare une **référence** vers un objet de type `Perso`
- De la même façon, `int[] tab` déclare une **référence** vers un objet de type `int[]`

Objets et références (3/3)

- Java définit deux entités distinctes
 - Un objet est une structure de données en mémoire
 - Une référence est **un identifiant unique** d'un objet
- `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 !

```
class Maison {  
    Perso[] habitants;  
    ...  
    static Perso[] get(Maison m) {  
        return m.habitants;  
    }  
}
```



La référence littérale `null`

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

```
Maison m = new Maison();  
Perso biblon = new Perso();
```

```
m.proprio = null; /* pas encore de propriétaire */  
...  
if(m.proprio == null)  
    m.proprio = biblon;
```

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



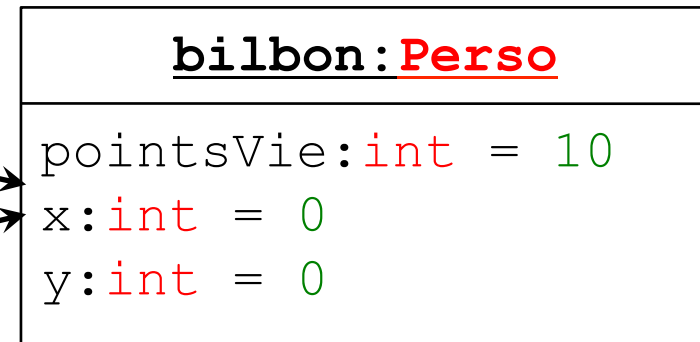
1. Les structures de données : objets et classes
2. Manipulation de tuples en Java
3. Tuples versus tableaux en Java
4. Objets et références
5. Objets et invocation de méthodes

Objets et invocation de méthodes

- On dit que les objets sont passés par **référence** en Java (puisque le code ne manipule que des références)

```
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 statiques

```
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) {  
        ...  
    }  
}
```

Notions clés

- Déclaration d'une classe définissant un tuple avec
`class Nom { type1 champs1; type2 champs2; ... }`
- 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