

Des Objets et des Classes



Qu'est-ce qu'un objet ?

- Les logiciels modernes organisent leurs données sous la forme d'**objets** appartenant à des **classes**. *Un peu comme les mathématiciens organisent leurs données sous la forme d'**éléments** appartenant à des **ensembles**.*

- Mais tout de suite une remarque. **En Java, certaines données sont des objets, et d'autres pas !** Oui, c'est un peu bizarre...

int = {entiers} n'est pas une classe mais un *type primitif*.

Un entier n'est pas un objet, mais une *donnée primitive*.

float = {nombres approchés} n'est pas une classe mais un *type primitif*.

Un nombre approché n'est pas un objet, mais une *donnée primitive*.

boolean = {true, false} n'est pas une classe mais un *type primitif*.

Un booléen n'est pas un objet, mais une *donnée primitive*.

Alors, où sont ces fameuses classes ?

- Jusqu'à présent, nous avons surtout travaillé avec des données primitives (nombres, booléens).
- Les classes pourront être :
 - soit des classes déjà fournies par le langage Java, qui dispose d'une impressionnante bibliothèque de classes toutes prêtes à être utilisées par le programmeur. C'est l'**API** de Java.

↓
Application Programming Interface

<http://java.sun.com/j2se/1.5.0/docs/api/>

- soit des classes que nous allons rédiger nous-mêmes pour un programme spécifique, parce que nous aurons besoin d'objets particuliers.

Exemple 1 : la classe String

- Les objets de la classe String de l'API sont les textes, ou chaînes de caractères.

```
String str = "Un exemple de chaîne !";
```

- L'opérateur + permet d'additionner des nombres mais aussi des chaînes, ou les deux à la fois, en convertissant les nombres en chaînes :

```
println("Je vous dis " + "bonjour !");
```

 → Je vous dis bonjour !

```
int x = 20, y = 40;  
println("Je place " + x + " euros.");
```

 → Je place 20 euros.

```
println(x + y + " sont placés.");
```

 → 60 sont placés.

```
println("Je place " + x + y);
```

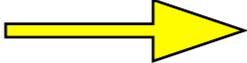
 → Je place 2040

→
de gauche à droite !

- **IMPORTANT** : un objet a du **savoir-faire**. Ce savoir-faire est décrit dans le texte (ou la documentation) de sa classe.

- Par exemple, on peut demander à une chaîne de nous donner sa longueur, en lui envoyant un **message** :

```
int n = str.Length();  
println("Longueur de str : " + n);
```



Longueur de str : 22

- Cette manière de parler aux objets doit être bien assimilée. Ce n'est pas le style mathématique usuel qui aurait écrit `length(str)` au lieu de `str.Length()`.

- Bien entendu, `length()` est bien une fonction, mais comme il s'agit d'un message adressé à un objet, on parle plutôt de **méthode** (une partie de son savoir-faire).

`length : void → int` est une **méthode** de la classe `String`.
C'est un **message** qui doit être envoyé à une chaîne.

- Il y a beaucoup d'autres méthodes dans la classe String. Pour les connaître, on ouvre l'API Java.

The screenshot shows a web browser window displaying the Java API documentation for the `String` class. The browser's address bar shows the URL `http://java.sun.com/j2se/1.5.0/docs/api/`. The page title is "String (Java 2 Platform SE 5.0)".

The left sidebar contains a navigation menu for "Java™ 2 Platform Standard Ed. 5.0". It lists "All Classes" and "Packages" including `java.applet`, `java.awt`, `java.awt.color`, and `java.awt.datatransfer`. A scrollable list of classes is shown, with `String` highlighted.

The main content area displays the "Class String" page. It includes navigation links for "Overview", "Package", "Class", "Use Tree", "Deprecated", "Index", and "Help". Below these are links for "PREV CLASS", "NEXT CLASS", "FRAMES", and "NO FRAMES". The "SUMMARY" section includes links for "NESTED", "FIELD", "CONSTR", and "METHOD".

The class hierarchy is shown as `java.lang.Object` extending `java.lang.String`. The "All Implemented Interfaces" section lists `Serializable`, `CharSequence`, and `Comparable<String>`.

The class declaration is shown as `public final class String` extending `Object` and implementing `Serializable`, `Comparable<String>`, and `CharSequence`.

The text explains that the `String` class represents character strings and that strings are constant and immutable. It provides an example of string creation: `String str = "abc";`.

- Le nom complet de la classe String de l'API est java.lang.String
- Ce qui précède le nom de la classe dans son nom complet se nomme un **paquetage** (*package* en anglais). La classe String se trouve donc dans le package java.lang
- Quelques autres méthodes de la classe String :

```
String str = "Prix 32 Euros";  
println("str : " + str);  
String strUp = str.toUpperCase();  
println("strUp : " + strUp);  
String strLo = str.toLowerCase();  
println("strLo : " + strLo);  
int n = str.length();  
println("longueur : " + n);  
String sub = str.substring(3,10);  
println("sub : " + sub);  
int comp1 = str.compareTo("Prix 32 Euros");  
println("comp1 : " + comp1);  
int comp2 = "libres".compareTo("livre");  
println("comp2 : " + comp2);  
int comp3 = "gump".compareTo("forrest");  
println("comp3 : " + comp3);
```

Run
→

```
str : Prix 32 Euros  
strUp : PRIX 32 EUROS  
strLo : prix 32 euros  
longueur : 13  
sub : x 32 Eu  
comp1 : 0  
comp2 : -20  
comp3 : 1
```

"libres" est avant "livre"
dans un dictionnaire !

Exemple 2 : la classe Point

- Le package `java.awt` contient des classes spécialisées dans le graphisme Java. Par exemple :

`java.awt.Point`

`java.awt.Rectangle`

- Contrairement au package `java.lang` qui est **automatiquement disponible**, le package `java.awt` ne l'est pas. Il faut donc au début d'un programme **importer les classes** qui nous intéressent :

```
import java.awt.Point;
```

- Dès lors, les méthodes de la classe `Point` sont accessibles sans avoir besoin d'écrire leur nom complet précédé de `java.awt`.
- Ouvrons la documentation de l'API à la classe `java.awt.Point` !

Field Summary

int	x	The x coordinate.
int	y	The y coordinate.

Constructor Summary

Point ()	Constructs and initializes a point at the origin (0, 0) of the coordinate space.
Point (int x, int y)	Constructs and initializes a point at the specified (x, y) location in the coordinate space.
Point (Point p)	Constructs and initializes a point with the same location as the specified <code>Point</code> object.

Method Summary

boolean	equals (Object obj)	Determines whether or not two points are equal.
Point	getLocation ()	Returns the location of this point.
double	getX ()	Returns the X coordinate of the point in double precision.
double	getY ()	Returns the Y coordinate of the point in double precision.
void	move (int x, int y)	Moves this point to the specified location in the (x, y) coordinate plane.

- Un point (objet de type Point) a deux **champs** (*field* : propriété, attribut) entiers nommés x et y. Donc si p est un point, alors p.x est l'abscisse de p, et p.y est l'ordonnée de p.

- Comment **construire un nouveau point** ? Avec un **constructeur** (*constructor*). La classe Point en propose trois. Utilisons le second :

```
Point p = new Point(5,-2);
```

Soit p le nouveau point de coordonnées (5 ; -2)

- Comment **parler à un point** ? Avec une **méthode** (*method*). La classe Point en propose une dizaine. Par exemple :

- la méthode getX() qui retourne l'abscisse d'un point :

```
int xp = p.getX();
```

- la méthode toString() qui retourne une représentation du point sous la forme d'une chaîne :

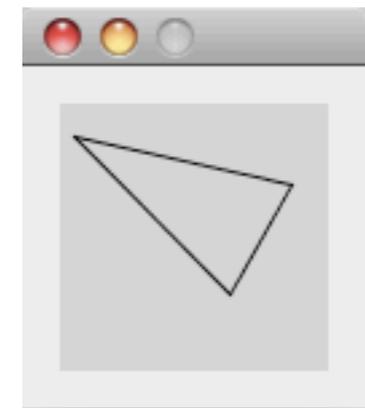
```
String chp = p.toString();
```

- Exemple : je construis trois points p1, p2, p3 du plan :

```
Point p1 = new Point(5, 12);  
Point p2 = new Point(86, 30);  
Point p3 = new Point(63, 71);
```

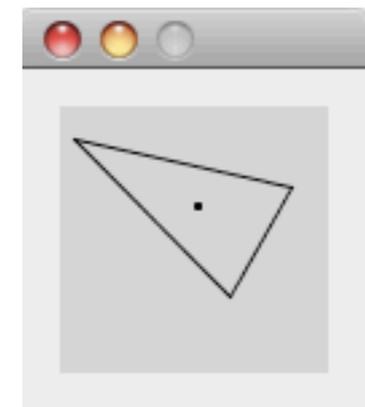
- Je les dessine sous la forme d'un triangle :

```
stroke(2);  
smooth();  
line(p1.x, p1.y, p2.x, p2.y);  
line(p2.x, p2.y, p3.x, p3.y);  
line(p3.x, p3.y, p1.x, p1.y);
```



- Je calcule le centre de gravité g du triangle et je le dessine :

```
Point g = new Point((p1.x+p2.x+p3.x)/3, (p1.y+p2.y+p3.y)/3);  
strokeWeight(2);  
point(g.x, g.y);
```



- Enfin, j'affiche g sous forme textuelle :

```
println("g = " + g);
```



```
g = java.awt.Point[x=51,y=37]
```

Le rôle de la méthode toString()

- La dernière ligne de la page précédente aurait dû s'écrire :

```
println("g = " + g.toString());
```

La règle du + dit que dans l'expression "g = " + g, la variable g doit être convertie en chaîne. Puisque g est un objet, Java va chercher si dans la classe de cet objet se trouve une méthode toString(). Si oui, il l'utilise pour transformer g en chaîne. Sinon, allez savoir...

- Dans la classe java.awt.Point, la représentation externe d'un Point prend la forme java.awt.Point[x=..., y=...].
- Vous ne pouvez pas changer la méthode toString() d'une classe Java déjà présente dans l'API.
- Lorsque vous programmerez vos propres classes, pensez à y mettre une méthode toString() pour que Java s'en serve dans le println !

Exemple 3 : la classe Random

- Le **générateur de nombres aléatoires** de Processing est très orienté vers les nombres *approchés*, ce qui est pénible pour obtenir des entiers !
- Utilisons plutôt la classe `java.util.Random` de l'API Java, qu'il faut importer :

```
import java.util.Random;
```

- La documentation propose un constructeur simple sans paramètre :

```
Random rand = new Random();
```

- La variable `rand` représente un objet "*générateur aléatoire*", capable de produire des nombres entiers ou des nombres approchés, suivant le message qu'on lui envoie. On trouve par exemple les deux méthodes `nextInt` et `nextFloat` :

```
int n = rand.nextInt(10);           // dans [0,9]  
float f = rand.nextFloat();         // dans [0,1[
```

Class Random

[java.lang.Object](#)└─ [java.util.Random](#)

Constructor Summary

[Random\(\)](#)

Creates a new random number generator.

[Random\(long seed\)](#)Creates a new random number generator using a single `long` seed:

Method Summary

protected int	next(int bits) Generates the next pseudorandom number.
boolean	nextBoolean() Returns the next pseudorandom, uniformly distributed <code>boolean</code> value from this random number generator's sequence.
void	nextBytes(byte[] bytes) Generates random bytes and places them into a user-supplied byte array.
double	nextDouble() Returns the next pseudorandom, uniformly distributed <code>double</code> value between 0.0 and 1.0 from this random number generator's sequence.
float	nextFloat() Returns the next pseudorandom, uniformly distributed <code>float</code> value between 0.0 and 1.0 from this random number generator's sequence.
double	nextGaussian() Returns the next pseudorandom, Gaussian ("normally") distributed <code>double</code> value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence.
int	nextInt() Returns the next pseudorandom, uniformly distributed <code>int</code> value from this random number generator's sequence.
int	nextInt(int n) Returns a pseudorandom, uniformly distributed <code>int</code> value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

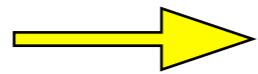
Attention à initialiser vos objets !

- Si vous déclarez une variable de type objet :

```
Point p;
```

sans lui donner de valeur, le compilateur refuse d'y accéder :

```
println("p = " + p);
```



The variable "p" may be accessed here before having been definitely assigned a value.

The variable "p" might not have been initialized.

- Il faudra donc l'**initialiser** quelque part avant de s'en servir :

```
p = new Point(5,-20);
```

- L'opérateur **new** crée un objet. Il est **suivi du constructeur** qui remplit les champs `x` et `y` de l'objet `p`, de sorte que `p.x == 5` et `p.y == -20`.

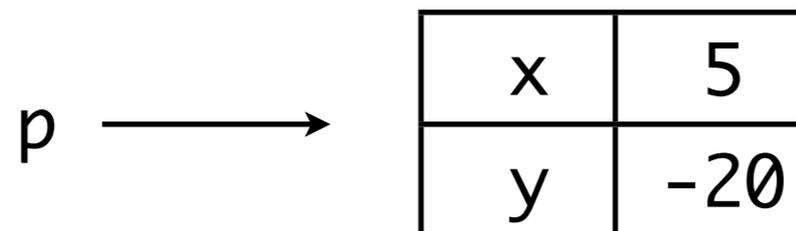
Les variables de type objet : des références

- Un **objet** est un élément (on dit une **instance**) d'une **classe**. Il possède des **champs** et un **savoir-faire** (des **méthodes**).
- L'ensemble des valeurs des champs à un instant donné constitue l'**état** de l'objet. Cet état est stocké à une certaine adresse dans la mémoire de l'ordinateur, sous forme compacte.

```
p = new Point(5, -20);
```

x	5
y	-20

- La variable *p* n'est en réalité pas autre chose que cette adresse. On dit que *p* est une **référence vers l'objet** (en réalité vers son état).



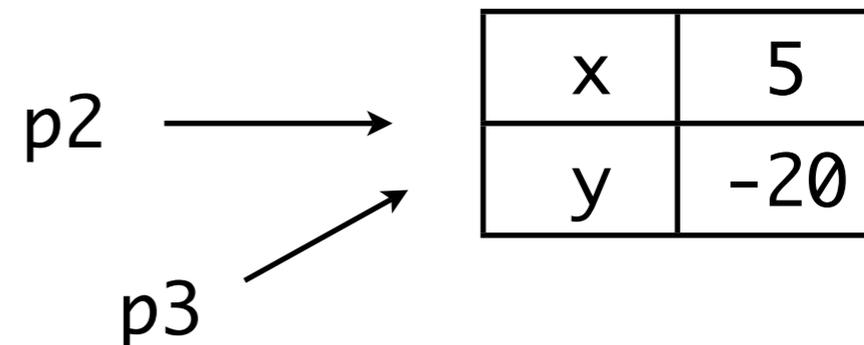
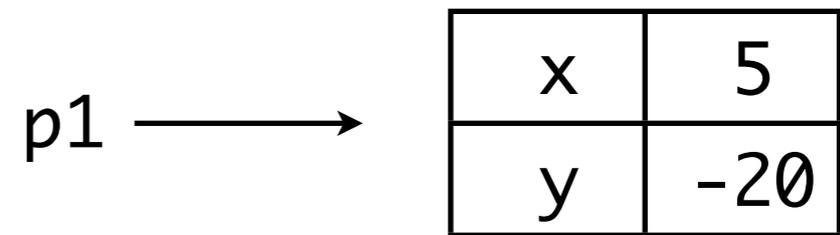
- Dans d'autres langages, on dirait que *p* est un *pointeur* vers l'état.

Affectation et égalité entre références

- Une variable de type **objet** est donc une référence (un pointeur).
- On dit indifféremment que *p* est un point, ou une référence vers un point. Attention à ces abus de langage !
- L'égalité `==` entre objets signifie en fait une comparaison de références ! Est-ce que deux objets sont un seul et unique objet ?

```
Point p1 = new Point(5, -20);  
Point p2 = new Point(5, -20);  
Point p3 = p2;
```

```
p1 == p2    → false  
p2 == p3    → true
```



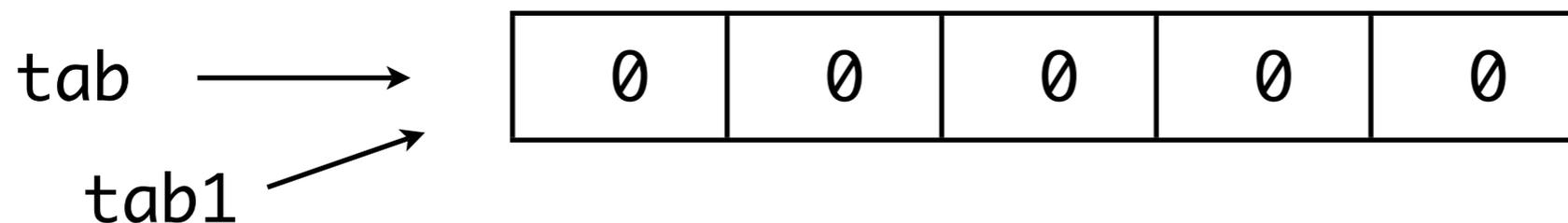
- Deux objets sont **égaux** s'ils sont situés au même endroit, pas s'ils ont le même état !!

Les tableaux sont-ils des objets ?

- OUI, mais des objets un peu *bizarres* car leur constructeur a une forme *bizarre* :

```
int[] tab = new int[5];
```

- Même le nom `int[]` de la classe est bizarre, puisque les classes s'écrivent avec un mot débutant par une **Majuscule** (Point, String ...).
- Il reste qu'une variable de type tableau est donc une **référence** vers le bloc compact où sont stockés les éléments du tableau.



- Donc l'instruction suivante ne crée pas de nouveau tableau :

```
int[] tab1 = tab;
```

D'ailleurs, il n'y a pas le mot new...

Travailler avec des tableaux d'objets !

- Exemple : je veux fabriquer un tableau de 3 points aléatoires dans un canvas 300 x 300 :

1. Je déclare un tableau de 3 points :

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

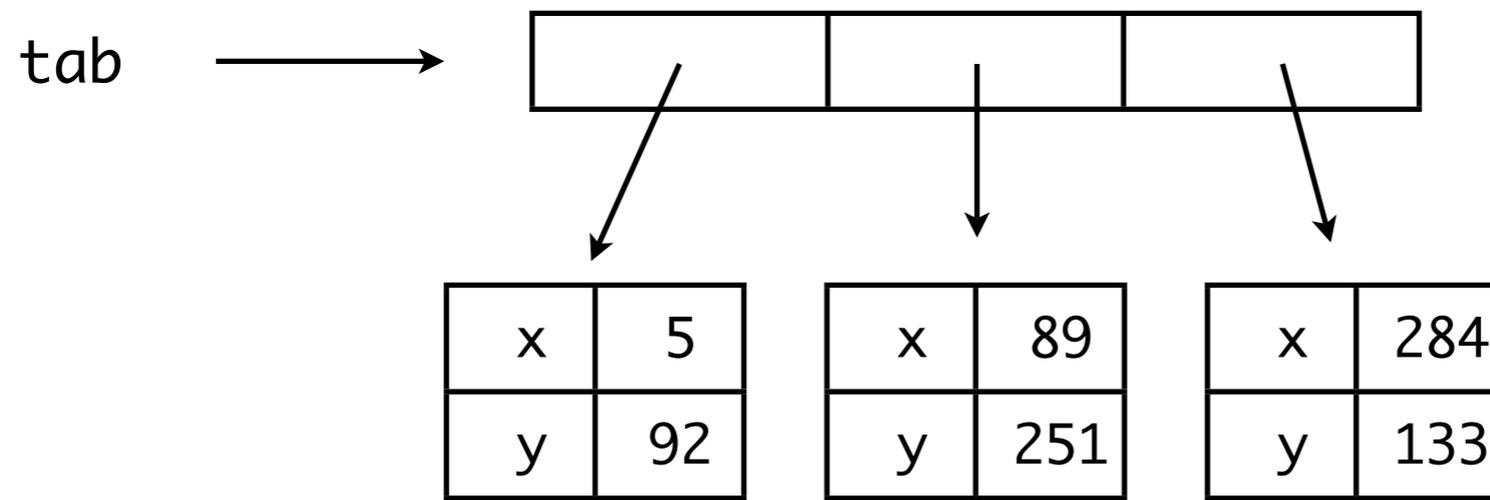
*ATTENTION : ce tableau existe mais ses cases sont des références non encore initialisées ! Java remplit alors le tableau avec une valeur spéciale nommée **null**.*

```
println(tab[0]);
```

 **null**

2. Je crée les 3 points aléatoires du tableau :

```
Random rand = new Random();  
for (int i = 0; i < tab.length; i = i + 1) {  
    tab[i] = new Point(rand.nextInt(300), rand.nextInt(300));  
}
```



3. Je calcule l'indice `iMin` du point le plus bas :

```
int yMin = tab[0].y, iMin = 0;      // iMin est l'indice du minimum
for (int i = 1; i < tab.length; i = i + 1) {
    if (tab[i].y > yMin) {
        yMin = tab[i].y;
        iMin = i;
    }
}
```

4. Je fais afficher le point le plus bas :

```
println("Point le plus bas : " + tab[iMin]);
```



```
java.awt.Point[x=89,y=251]
```