

PROGRAMMATION PAR LES OBJETS EN JAVA

TRAVAUX DIRIGÉS 1

NAJIB TOUNSI

(Lien permanent: <http://www.mescours.ma/Java/TD/tdJava1.html> (.pdf))

L'objectif de ce TD est d'écrire quelques programmes simples pour se familiariser avec le langage Java. Le travail de compilation / exécution se fera en mode commande, e.g. *Terminal* de *Linux* / *MacOS*, ou *cmd* de *Windows*.

1. AVANT PROPOS

L'élève est censé(e) savoir comment créer un répertoire (commandes `md` ou `mkdir`, `cd`, etc.) et son équivalent graphique (créer / ouvrir un nouveau dossier etc.), lister son contenu (commande `dir` ou `ls`), savoir utiliser un éditeur de texte pour créer un programme et sauvegarder un programme source sous un répertoire donné. On doit aussi connaître les commandes de base *Unix* (*Linux* ou *MacOS*) et *Windows* (`cat`, `type`, `more` etc.)

2. SOMMAIRE

1. [Avant propos](#)
2. [Sommaire](#)
3. [Installation de Java et préparation du travail.](#)
4. [Compilation exécution d'un programme Java](#)
 1. [Programme hello world](#)
5. [E/S de données élémentaires \(classe scanner\)](#)
 1. [Lecture d'un simple entier avec la classe Java scanner.](#)
 2. [Lecture de plusieurs données.](#)
 3. [Lecture d'un caractère.](#)
6. [Programme avec traitement et usage de fonctions](#)
7. [Surcharge de fonctions](#)
 1. [Méthodes à nombre variable de paramètres](#)
8. [Quelques classes de bibliothèque](#)
 1. [La Classe Math du package lang.](#)
 2. [La Classe Calendar du package util.](#)
9. [Les Wrappers.](#)
10. [Objets vs Valeurs](#)
11. [Passage des paramètres en Java](#)
12. [Evaluation](#)

13. [Annexe: Préparation du TP sur PC.](#)

3. INSTALLATION DE JAVA ET PRÉPARATION DU TRAVAIL.

Sous *MacOS* ou *Linux*, Java est généralement déjà fourni. (le vérifier en tapant `javac -version`. On devrait voir `javac 1.8.0_121` ou une version supérieure). Sous *Windows*, voir l'annexe: [Préparation du TP sur PC](#)

4. COMPILATION EXÉCUTION D'UN PROGRAMME JAVA

NB. Créer un nouveau fichier d'extension `.java` pour chaque programme. Utiliser un éditeur de texte courant. Aide à la présentation syntaxique. Par exemple, Sublime (<https://www.sublimetext.com/3>).

4.1. PROGRAMME *HELLO WORLD*

Ecrire le programme suivant dans un fichier `HelloWorld.java`:

```
1  class HelloWorld {
2      static public void main(String args[]){
3          System.out.println("Hello, World");
4      }
5  };
```

- Compiler par: `javac HelloWorld.java`
- Exécuter par: `java HelloWorld`
- Constater la création du fichier `HelloWorld.class` résultat de la compilation. Commandes `ls` ou `dir`.
- Noter surtout que le nom du fichier d'extension `.class` vient du nom de la classe (première ligne).

Remarque : Le fichier source `.java` ici a le même nom que la classe contenant la fonction `main()`, i.e. l'identificateur qui suit le mot `class` dans le code source. Le nom du fichier `.class` généré pour cette classe est cet identificateur justement.

Exercice: rajouter d'autres lignes pour imprimer "bonjour" en d'autres langues (e.g. "Bonjour, ça va?", "سلام").

Petite remarque ici si vous êtes sous *Windows* en mode ligne commande :

- Avec *Bloc-notes*, enregistrer le fichier source en UTF-8 (menu Enregistrer+Encodage). En plus cet éditeur rajoute le caractère invisible *ByteOrderMark* (BOM `\uFEFF`) en début de fichier. Ce que le compilateur Java n'accepte pas. Utiliser un autre éditeur pour retirer ce caractère. Copier / coller le source vers un autre éditeur (e.g. Sublime Text.)

- Il se peut qu'à l'exécution on voit ???????? pour le mot en Arabe.
 - Taper la commande **chcp 65001** pour changer le code page vers Unicode UTF-8 et recommencer.
 - Si maintenant on voit □□□□, changer de police d'affichage (menu Propriété+Police).
 - En tout cas, il est toujours possible de rediriger le résultat d'exécution vers un fichier .txt qu'on visualisera à part ("java votreClasse > resultat.txt).

Version 2 : Créer un second programme Hello2.java. On va imprimer: Bonjour *Untel*. où le nom *Untel* est donné au lancement du programme sur la ligne commande java.

```

1 class Hello2 {
2     static public void main(String args[]){
3         if (args.length !=0)
4             System.out.println("Hello " + args[0]);
5     }
6 }

```

Exécuter avec : java Hello2 Fatima

A noter: Le mot *Fatima* constitue le premier élément args[0] du tableau args[], paramètre de la fonction *main*.

Le champ length donne la taille d'un tableau en Java.

5. E/S DE DONNÉES ÉLÉMENTAIRES (CLASSE SCANNER)

5.1. LECTURE D'UN SIMPLE ENTIER AVEC LA CLASSE JAVA SCANNER.

```

1 import java.util.*;    // Package Java qui contient la cla
2
3 class Saisie {
4     /**
5      * Lecture d'un entier, version scanner
6      */
7
8     static public void main(String args[]) {
9         Scanner clavier = new Scanner(System.in);
10        System.out.print("Donner entier: ");
11        int n = clavier.nextInt();
12        System.out.println (n*2);
13    }
14 }

```

La classe `scanner` se trouve dans le package `java.util`. Elle permet de déclarer un objet, variable `clavier` ici, sur lequel on peut lire des données. On l'a instancié ici par :

- `new Scanner(System.in);`

à partir du fichier standard d'entrée représenté par l'objet `System.in`

La méthode `nextInt()` permet de lire un entier. Pour lire un réel, `nextFloat()` ou `nextDouble()`. Pour les chaînes, `nextLine()` pour lire une ligne ou `next()` pour une chaîne, etc. (exercice: vérifier ces méthodes). Il n'y a pas `nextChar()` !

Pour en savoir plus, voir (<https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html>).

Note : Avec `scanner` on instancie normalement un objet à partir d'un fichier texte qui contient des données, e.g.

```
new Scanner(new File("monFichier.data"));
```

Il faut alors utiliser l'exception `FileNotFoundException` (de la classe `File`) pour cette instruction.

5.2. LECTURE DE PLUSIEURS DONNÉES.

```
1  import java.util.*;    // Package Java qui contient la cla
2  class Saisie2 {
3      static public void main(String args[]) {
4          // Partie Déclaration
5          Scanner clavier = new Scanner(System.in);
6          int age;
7          String nom;
8          double taille;
9
10         // Partie lecture de données
11         System.out.print("Quelle est votre nom?: ");
12         nom = clavier.nextLine();
13         System.out.print("Quelle est votre age?: ");
14         age = clavier.nextInt();
15         System.out.print("Quelle est votre taille?: ");
16         taille = clavier.nextDouble();
17
18         // Partie sortie des résultats
19         System.out.println("Bonjour "+nom);
20         System.out.print("Vous avez "+age+" ans");
21         System.out.println(" et vous mesurez "+taille+" mè
22     }
23 };
```

Remarque : On pourrait saisir l'âge et la taille sur une même ligne séparés par espace. la classe `scanner` utilise les caractères tabulation, fin de ligne et espaces comme séparateurs par défaut. Mais l'utilisateur peut en spécifier d'autres avec la méthode `useDelimiter()`. Voir la [référence](#) ci-dessus pour plus de détails.

Exercice: Créer, compiler et exécuter ce programme.

Version 2: Le même programme. Sortie des résultats avec format.

Au lieu de `println()`, on peut utiliser `format()` qui a la syntaxe de *printf()* de C.

Exercice: Remplacer les trois dernières lignes `System.out.print(ln)` par :

```
System.out.format("Bonjour %s %nVous avez %d ans", nom, age);
System.out.format(" et vous mesurez %f mètres %n", taille);
```

NB. la méthode `System.out.printf()` existe aussi et est équivalente à `System.out.format()`. `%n` est le format pour "nouvelle ligne".

5.3. LECTURE D'UN CARACTÈRE.

On lit une chaîne avec `next()`, et on prend son premier caractère avec `charAt(0)`.

```
1 import java.util.Scanner;
2 class monChar {
3     static public void main(String args[]) {
4         char monChar;
5         Scanner clavier = new Scanner(System.in);
6         String s = clavier.next();
7         monChar = s.charAt(0);
8         System.out.println(monChar);
9     }
10 }
```

On peut condenser et écrire :

```
monChar = clavier.next().charAt(0);
```

sans déclarer explicitement la chaîne `s`.

6. PROGRAMME AVEC TRAITEMENT ET USAGE DE FONCTIONS

Conversion d'une température donnée en degré *Celsius*, vers une température en degré *Fahrenheit*.

a. Sur le même modèle que le programme précédent, créer un programme Java (fichier `Celsius.java`)

```
1 class Celsius {
2     public static void main(String args[]){
3         ...
```

```

4 |         }
5 |     }

```

qui effectue cette conversion. Utiliser la formule:

$$f = 9./5 * c + 32$$

où f est la t° *Fahrenheit* et c la t° *Celsius*. Appeler la classe `Celsius`, et le source `Celsius.java`.

NB. Pour que la division $9/5$ s'effectue en réel, utiliser `9.` au lieu `9` dans le source Java.

b. Version 2: Usage de fonction en Java (*static*, dans la même classe).

Dans le source `Celsius.java`. ajouter maintenant la fonction (on dit aussi *méthode*):

```

1 |     static double c2f(int c){
2 |         double f = 9./5 * c + 32;
3 |         return f;
4 |     }

```

(à ajouter après la fonction `main()` avant l' } de fin de classe *Celsius*) et remplacer l'instruction initiale

```
f = 9./5 * c + 32;
```

par

```
f = c2f (c) ;
```

Compiler et exécuter. Discuter.

c. Version 3: Mettre maintenant la fonction `c2f()` ci-dessus dans une *nouvelle* classe que l'on appellera `Celc`.

```

1 | class Celc {
2 |     static double c2f(int c){
3 |         double f = 9./5 * c + 32;
4 |         return f;
5 |     }
6 | };

```

(à ajouter après l' }; de fin de classe *Celsius* dans le même fichier source).

et remplacer l'instruction initiale (programme *main* toujours)

```
f = c2f (c) ;
```

par

```
f = Celc.c2f (c) ;
```

Compiler et exécuter. Discuter.

d. Version 4: (Ne pas imiter cet exemple.) Maintenant, enlever le mot `static` dans le profile de la fonction `c2f()` de la classe `Celc`
(`double c2f(int c)` au lieu de `static double c2f (int c)`)

et remplacer l'instruction initiale (programme *main* toujours)

```
f = Celc.c2f (c) ;
```

par

```
f = obj.c2f (c) ;
```

où **obj** est une variable à déclarer auparavant par:

```
Celc obj = new Celc() ;
```

Compiler et exécuter. Discuter.

A Noter: Dans ce dernier cas, on doit d'abord instancier un objet *obj* de la classe *Celc* pour pouvoir appeler (donc lui appliquer) la fonction *c2f()*, dite **méthode d'instance** dans ce cas. Mais comme, il n'y a pas de données propres à chaque objet de cette classe *Celc*, il n'y a pas besoin d'instancier un objet pour utiliser la méthode *c2f()*. C'est pour cette raison qu'on peut la déclarer *static*. Comme cela on l'appelle sans instancier d'objets. On dit **méthode de classe** dans ce cas.

Remarque: Noter aussi qu'on pourrait, dans le premier cas, instancier plusieurs objets *obj₁*, *obj₂*, ... de classe *Celc*. L'appel *obj_n.c2f(c)* serait indifférent de l'objet auquel il s'applique. Ce qui explique pourquoi il y a des fonctions *static* et justifie la méthode de classe dans la Version-3 ci-dessus.

7. SURCHARGE DE FONCTIONS

En principe, une méthode a un nom unique dans une classe. Cependant Java permet à une méthode d'avoir le même nom que d'autres grâce au mécanisme de surcharge (ang. *overload*). Java utilise leur signature pour distinguer entre les différentes méthodes ayant le même nom dans une classe, c'est à dire la liste des paramètres. Ce sont le nombre et le type des paramètres qui permet de distinguer.

Soit la classe `DataArtist` :

```
1 class DataArtist {  
2     static void draw(String s) {  
3         System.out.println("Ceci est une chaîne: "+s);  
4     }  
5     static void draw(int i) {  
6         System.out.println("Ceci est un entier: "+i);  
7     }
```

```

8      static void draw(double f) {
9          System.out.println("Maintenant un double: "+f);
10     }
11     static void draw(int i, double f) {
12         System.out.format("Une entier %d et un double %f", i, f);
13     }
14 }

```

Les différents appels suivant correspondent aux bonnes fonctions:

```

DataArtist.draw ("Picasso");    // 1ère méthode, draw(String)
DataArtist.draw (1);            // 2e méthode, draw(Int)
DataArtist.draw (3.1459);       // 3e méthode, draw(double)
DataArtist.draw (2, 1.68);      // 4e méthode, draw (int, double)

```

Exercice: le vérifier.

A noter: Le paramètre retour d'une fonction ne permet pas de distinguer entre deux fonctions. `static int draw(int)` est la même signature que `static void draw(int)`. Le vérifier.

La surcharge est surtout utile pour définir plusieurs constructeurs pour un objet.

7.1. MÉTHODES À NOMBRE VARIABLE DE PARAMÈTRES

Un aspect particulier de la surcharge et la déclaration d'un nombre arbitraire de paramètres. Cela se fait par une ellipse (trois points `...`). Soit l'exemple:

```

1      public static void f(char c, int... p) {
2          System.out.println(c + " " + p.length);
3          for (int e:p) System.out.println(" " + e);
4      }

```

L'ellipse `...` doit apparaître après le dernier paramètre. Ici, on a un premier paramètre `char` et ensuite 0, 1 ou plusieurs entiers dans une variable `p`. `int...` signifie un nombre quelconque de paramètres entiers. Au fait, l'ellipse remplace favorablement un paramètre tableau dont la taille est bornée. C'est comme un tableau mais de taille illimitée (sauf par le système). D'où l'usage possible de `p.length` dans la fonction pour connaître la taille actuelle du paramètre `p`. La fonction imprime ensuite ses paramètres.

Les appels suivants sont valides :

```

char c='a';
f(c);
f(c, 1);
f(c, 2,3,4);
int[] monTableau = {5,6,7,8 };
f(c, monTableau);

```

Exercice : Vérifier l'exemple et chercher d'autres cas à vous.

NB. C'est comme l'opérateur relationnel de *Projection*. qui admet en paramètre un nom de relation et ensuite une suite de noms d'attributs de projection.

8. QUELQUES CLASSES DE BIBLIOTHÈQUE

8.1. LA CLASSE **MATH** DU PACKAGE **LANG**.

```
import java.lang.Math;
```

Cette classe contient, en plus des constantes *e* et *pi*, les méthodes pour le calcul numériques (fonctions mathématiques classiques). Ce sont des méthodes toutes `static`. Exemple `double Math.abs (double)`, `double Math.sqrt (double)` etc.

The screenshot shows the Oracle Java SE 8 API documentation for the `Math` class. The browser address bar shows `http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html`. The page is divided into two main sections: 'Fields' and 'Method Summary'.

Fields

Modifier and Type	Field and Description
static double	E The double value that is closer than any other to <i>e</i> , the base of the natural logarithms.
static double	PI The double value that is closer than any other to <i>pi</i> , the ratio of the circumference of a circle to its diameter.

Method Summary

All Methods		Static Methods	Concrete Methods
Modifier and Type	Method and Description		
static double	abs(double a) Returns the absolute value of a double value.		
static float	abs(float a) Returns the absolute value of a float value.		
static int	abs(int a) Returns the absolute value of an int value.		

(<http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>)

Exercice : Vérifier le programme suivant:

```
1 import java.lang.Math;
2 class TestMath {
3     static public void main(String args[]) {
4         System.out.println("e = " + Math.E);
5         System.out.println("pi = " + Math.PI);
6         int largeur = 3, longueur = 4;
7         double w = Math.pow(largeur,2) + Math.pow(longueur,2);
8         double hypotenuse = Math.sqrt(w);
9         System.out.println("Hypoténuse = " + hypotenuse);
10    }
```

```

11         //... il vaut mieux écrire largeur * largeur que
12     }
13 }

```

NB. L'instruction `import` n'est pas nécessaire ici. Les classes du package `java.lang` sont importées implicitement.

Exercice: Utiliser la fonction `static double random()`, qui retourne un nombre pseudo-aléatoire supérieur ou égal à 0.0 et inférieur à 1.0, pour imprimer 6 nombres entiers aléatoires compris entre 1 et 49 inclus.

Modifier le programme pour avoir 6 nombres tous différents (c.f. jeu du *Loto*).

8.2. LA CLASSE `Calendar` DU PACKAGE `UTIL`.

Cette classe abstraite contient les méthodes pour manipuler les dates dans toute ces composantes (à travers les champs `YEAR`, `MONTH`, `DAY_OF_MONTH`, `HOURL` etc.), et faire des calculs sur les dates comme déterminer le prochain jour ou semaine.

(voir <http://docs.oracle.com/javase/8/docs/api/java/util/Calendar.html> pour les détails de champs et méthodes)

Exemple-1 : L'objet `Calendar` et son contenu:

```

1  import java.util.*;
2  class TestCalendar {
3      static public void main(String args[]) {
4
5          Calendar rightNow = Calendar.getInstance();
6              // Création d'une instance date initialisée
7
8          System.out.println(rightNow);
9              // le contenu de rightNow pour les curieux!
10     }
11 }

```

Après exécution (le Fri Mar 23 10:34:04 WET 2012):

```

1  java.util.GregorianCalendar[
2
3      ... informations techniques sur le fuseau horaire,
4      la zone, l'heure d'été etc. ...
5
6      firstDayOfWeek=1,
7      minimalDaysInFirstWeek=1,
8      ERA=1,
9      YEAR=2012,
10     MONTH=2,                                <-- Mois numérotée à
11     WEEK_OF_YEAR=12,
12     WEEK_OF_MONTH=4,
13     DAY_OF_MONTH=23,

```

```

14 DAY_OF_YEAR=83,
15 DAY_OF_WEEK=6,                                <-- 6e jour de la se
16 DAY_OF_WEEK_IN_MONTH=4,
17 AM_PM=0,
18 HOUR=10,
19 HOUR_OF_DAY=10,
20 MINUTE=34,
21 SECOND=7,
22 MILLISECOND=578,
23 ZONE_OFFSET=0,                                <-- Décalage horair
24 DST_OFFSET=0
25 ]

```

Exemple-2 : Les fonctions d'accès aux champs. Les fonctions `get(...)` sont nombreuses et de la forme `get(quelqueChose)` où le paramètre est un champs Calendar (constante symbolique de type entier). Voir la [documentation officielle](#). API ci-dessus.

```

1 class TestCalendar {
2     static public void main(String args[]) {
3         Calendar rightNow = Calendar.getInstance();
4
5         // tester les get()
6         int j = rightNow.get (Calendar.DAY_OF_MONTH);
7         int m = rightNow.get (Calendar.MONTH);
8         int y = rightNow.get (Calendar.YEAR);
9
10        System.out.println("On est le:"+j+ "/" + (m+1) + "/"
11    }
12 }

```

Affiche:

On est le:19/4/2019

On a rajouté 1 pour le mois, car ils sont comptés à partir de 0. (Voir résultat exemple-1)

Exemple-3 : Calcul du jour.

Dans le même programme, rajouter en les complétant les instructions suivantes pour déterminer le nom du jour de la semaine de façon à imprimer: "On est le:**Vendredi** 19/4/2019".

```

1 String jour = "";
2 switch(rightNow.get(Calendar.DAY_OF_WEEK)) {
3     case 1: jour = "Dimanche"; break;
4     case 2: jour = "Lundi"; break;
5     // ... à compléter ...
6     case 7: jour = "Samedi"; break;
7 }
8 System.out.printf("On est le: %s %d/%d/%d\n", jour, j, m+1, y);

```

Exemple-4 : Utilisation des *set(...)* pour changer les dates. Même remarque que pour les *get(...)*. Voir l'[API](#) ci-dessus.

La fonction *set (champ , valeur)* permet de changer le champ (entier) d'une date

Exercice : Reprendre le même programme, et changer de date du mois vers *Mai*.

```
rightNow.set(Calendar.MONTH, Calendar.MAY);
```

et le jour du mois vers le 28.

```
rightNow.set(Calendar.DAY_OF_MONTH, 28);
```

pour imprimer la date du jour comme précédemment.

Utiliser la fonction *add (champ, valeur)* pour calculer la date du lendemain.

```
rightNow.add (Calendar.DAY_OF_MONTH, 1);
```

Exercice : Vérifier que le lendemain du 28 Février est 29 Février 2016, année bissextile, et que le lendemain du 28 Février est 1er Mars pour 2019.

Exercices : Faire d'autres exemples... Utiliser le champ `Hour_OF_DAY`.

9. LES WRAPPERS.

Classes `Integer`, `Float`, `Boolean` etc. du package `java.lang`. Ce sont des sous-classes de `Number`.

Ces classes détiennent principalement des méthodes pour convertir entre elles des données numériques, surtout vers (ou à partir de) leur forme chaîne de caractères.

a) On considèrera le cas de la classe `Integer`, les autres sont semblables.

- **String vers Integer**. Fonction *valueOf()*. Correspondance entre la chaîne "123" et l'entier `Integer` de valeur 123.

```
Integer I;  
I = Integer.valueOf("123");
```

Remarque: Par constructeur on peut faire la même conversion avec `new Integer ( "123" );` .

- **String vers int**. Fonction *parseInt()*. Correspondance entre "456" et l'entier `int` de valeur 456.

```
int i;  
i = Integer.parseInt("456");
```

- **Integer vers int.** Fonction `intValue()`. Correspondance entre objet `Integer` et entier `int`.

```
int i; Integer I;
i = I.intValue();
```

Remarquer que c'est une méthode d'instance ici (fonction d'accès).

- **int vers Integer.** Correspondance inverse de `int` vers `Integer`.

```
Integer I = new Integer (i);    // Par constructeur
```

- **Integer ou int vers chaîne String.** Passage réciproque de `Integer` ou `int` vers chaîne `String`.

Méthode générale `valueOf()` de la classe `String` s'appliquant (par surcharge) à tous les types primitifs.

```
int i : 34;
s = String.valueOf(i);    // s devient "34"
```

Méthode `toString()` de la classe `Integer` ici.

```
String s;
Integer I = 345;
s = I.toString();    // s devient "345"
```

Cette méthode est intéressante car elle est héritée de la classe *Object* et peut donc s'appliquer à tout objet si elle est redéfinie. On peut l'utiliser à profit pour imprimer un objet `println(objet.toString());`.

- Comparaison entre deux objets `Integer`. Résultat `int`.

```
int i = I.compareTo(J);    // 0 si I = J, négatif si I<J, p
```

On peut aussi utiliser les opérateurs de comparaison comme `I < J`.

b) Le cas des autres classes est analogue. Par exemple:

```
1 float f = 12.34f;
2 Float F = new Float (f);    // de float vers Float
3 String s;
4 F = Float.valueOf("12.34");    // de String vers Float
5 f = Float.parseFloat ("12.34");    // de String vers float
6 f = F.floatValue();    // de Float vers float
7 s = F.toString();    // de Float vers String
8 s = String.valueOf(f);    // de float vers St
```

...

Voir par exemple <https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html>

10. OBJETS VS VALEURS

Les références à objets : Une affectation $x = y$ n'est pas toujours le même effet selon que ce soit un objet primitif ou non. Selon que x et y contiennent des valeurs primitives ou sont des références vers des objets.

On va le tester sur des entiers (objets primitifs) et sur un tableau (objet java).

```

1  //
2  // Objets vs Valeurs
3  //
4  class Test {
5      static public void main(String[] args) {
6
7          int x = 1, y;
8
9          ////////// affectation de valeurs //////////
10         y = x; // deux valeurs égales mais objets diffé
11         System.out.println("Avant (x = 100): x = " + x + "
12         x = 100;
13         System.out.println("Après (x = 100): x = " + x + " mais
14         // constater que y n'as pas changé
15
16         int[] u = {4, 5}; // tableau à 2 entiers.
17         int[] v;
18
19         ////////// même chose avec objets (ici tableaux) //
20         v = u;
21         System.out.println("Avant (u[0] = 100): u[0] = "
22         u[0] = 100;
23         System.out.println("Après (u[0] = 100): u[0] = " +
24         // constater que v[0] a changé aussi
25     }
26 }
```

On obtiendra:

```

1  Avant (x = 100):  x = 1 , y = 1
2  Après (x = 100):  x = 100 mais y = 1          <-- y n
3  Avant (u[0] = 100):  u[0] = 4 , v[0] = 4
4  Après (u[0] = 100):  u[0] = 100 ET v[0] = 100    <-- v
```

où on voit que u et v désignent le même objet: une modification de u est aussi une modification de v .

Exercice:

Selon la même idée, faire un programme qui teste l'affectation $x = y$ où x et y sont des objet d'une classe C .

Indication: Soit la classe C :

```
1 class C {  
2     int x; // un champ entier  
3     public int getX(){return x;} // méthode pour co  
4     public void setX(int p){x = p;} // méthode pour modi  
5 }
```

Créer deux instances x et y :

```
C x = new C(), y = new C();
```

les initialiser et les afficher :

```
x.setX(5);  
y.setX(6);  
System.out.println(x.getX() + " et " + y.getX());
```

Constater le résultat: 5 et 6. Deux objets différents.

Faire maintenant :

```
x = y;
```

ensuite modifier x et afficher x y :

```
x.setX(4);  
System.out.println(x.getX() + " et " + y.getX());
```

Constater que y aussi a été modifié (résultat: 4 et 4).

11. PASSAGE DES PARAMÈTRES EN JAVA

On retrouve cette même caractéristique dans le passage des paramètres en Java.

Les objets en Java sont passés en paramètre par *valeur*. Mais cette valeur peut-être une donnée (objet primitif) ou une référence à un objet.

Tester sur l'exemple suivant:

```
1 //  
2 // Passage des paramètres  
3 //  
4  
5 class Test extends Object {  
6     static public void main(String args[]){
```

```

7
8     int x = 2;
9     System.out.println("Avant modif,  x = " + x);
10    modifVal(x);
11    System.out.println("Après modif,  x = " + x);
12
13    int [] t = {2, 3};
14    System.out.println("Avant modif,  t[0] = " + t[0]);
15    modifObj(t);
16    System.out.println("Après modif,  t[0] = " + t[0]);
17
18
19
20    public static void modifObj(int p[]) {
21        p[0] = p[0] + 200;    // Objet référencé p est mo
22    }
23
24    public static void modifVal(int x) {
25        x = x + 200;    // paramètre x modifié
26    }
27 }

```

On obtient:

```

1 Avant modif,  x = 2
2 Après modif,  x = 2
3 Avant modif,  t[0] = 2
4 Après modif,  t[0] = 202

```

NB: Quand on change le paramètre lui-même, c'est à dire la variable *p* ici, on change la référence mais pas le tableau.

Exercice 8.1: Vérifier maintenant qu'en modifiant la méthode `modifObj` pour changer *p* :

```

1 public static void modifObj(int p[]) {
2     p = new int [] {200, 300, 400};
3 }

```

t[0] ne va pas changer. On aura toujours *t*[0] = 2.

Exercice 8.2 : Reprendre la [classe C](#) précédente (§7) et tester le passage de paramètre d'un objet de classe C.

Déclarer une fonction:

```

1 public static void modifObj(C p) {
2     p.setX(p.getX() + 200);    // Objet référencé p e
3 }

```

et l'appeler par `modifObj(o);` (en déclarant: `C o = new C();`)

12. EVALUATION

Certains exercices seront évalués à la demande et corrigés par l'encadrant du TP..

[Prochain TD.](#)

13. ANNEXE: PRÉPARATION DU TP SUR PC.

Installer Java en téléchargeant le JDK à :

<http://www.oracle.com/technetwork/java/javase/downloads/>

(Choisir JDK et ensuite la version binaire correspondant votre OS)

Pour travailler en mode commande (c:\>): *Démarrer>Exécuter>*taper *cmd*.

On pourrait souhaiter de créer un répertoire pour y sauvegarder ses programmes java. Aller vers la racine (ou sur le Bureau si ce n'est pas autorisé)

```
cd c:\
```

et taper

```
md votreNom
```

```
cd votreNom
```

Pour taper ses programmes utiliser *blocNote* simple ou *Sublime* pour une meilleure assistance syntaxique . Ouvrez d'abord votre répertoire dans une fenêtre *Windows*.

Les commandes **javac** et **java** se trouvent normalement dans le répertoire **c:\jdk...\bin**. Pour éviter de taper le chemin complet

```
c:\jdk...\bin\javac pgme.java
```

et taper simplement

```
javac pgme.java
```

rajouter le répertoire **c:\jdk...\bin** dans la variable d'environnement *Path*. Exemple :

Avant : Path=C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem

Après : Path=C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;
c:\jdk...\bin

Aller dans *panneau de configuration*, dossier *System* ensuite l'onglet *Avancé*. Modifier alors la variable d'environnement *Path*.