Langage Java Chapitre 2

La syntaxe java .
Mots réservés.
Types de données de base, variables et constantes.
Transtypage.
Opérateurs et expressions.
Contrôle de programmes, tests et boucles.
Les tableaux, l'interface Comparable

1° Mots réservés (mots clés)

abstract	boolean	break	byte	case	cast	catch
char	class	const	continue	default	do	double
else	extends	final	finally	float	for	
	goto	if	implements	import	instanceof	
int	interface	long	native	new		
	package	private	protected	public	return	
short	static	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while	

2° Types et variables de base (primitifs)

type boolean

Intervalle de valeurs : true/false

Opérations : == != ! & ^ | && ||

type byte (sur 8bits)

Intervalle de valeurs : de -128 à 127

Opérations : == != < > <= >= + - * / %
++ -- << >> >>> & | ^ ?

type short (sur 16 bits)

Intervalle de valeurs : de -32768 à 32767

Opérations : == != < > <= >= + - * / %
++ -- << >> >>> & | ^ ?

type int (sur 32 bits)

Intervalle de valeurs : de -2^{31} à $+2^{31} – 1$

Opérations : == != < > <= >= + - * / %
++ -- << >> >>> & | ^ ?

type long (sur 64 bits)

Intervalle de valeurs : de -2^{63} à $+2^{63} – 1$

Opérations : == != < > <= >= + - * / %
++ -- << >> >>> & | ^ ?

type char (sur 16 bits)

Les caractères en unicode sont sur 16 bits : de 0x0000 à 0xFFFF.
En simplifiant pour les caractères ASCII : Octet de poids fort à 0
octet de poids faible: code ASCII

type float (sur 32 bits)

Valeur positive la plus grande : $(2-2^{-23})*2^{127} = 3.40282347e38$
Valeur positive la plus petite : $2^{-149} = 1,40129846e-45$
Opérations : == != < > <= >= + - * / % ++ --
<< >> >>> & | ^ ?

type double (sur 64 bits)
Valeur positive la plus grande : $(2-2^{-52})*2^{1023} = 1.79769313486231570e308$
Valeur positive la plus petite : $2^{-1022} = 2.22507385850720139e-308$

Opérations : == != < > <= >= + - * / % ++ --
<< >> >>> & | ^ ?

3° Déclarations des variables et des constantes.

Toute donnée déclarée à l'intérieur d'une méthode est une variable.
Exemples : Déclarations de variables.

boolean test=true ; byte registre8 ; short nombre1; int nombre2 ;

long facto ; char car='A' ; float f1 ; double dl ;

Une variable peut être initialisée ou non.

Un caractère est initialisé entre apostrophes ‘A’ .

Pour déclarer des constantes ajouter le mot réservé **final**. Exemple :

final int VALEURMAX=200 ; // une constante ne peut plus changer de valeur

REMARQUE : Java propose des classes pour tous les types de base.

Exemples :
Des objets sont instanciés et créés avec l’opérateur new :

```
Integer n= new Integer(0);          // un objet de type Integer
Float f= new Float(0.0);            // un objet de type Float
Double d = new Double(0.0);         // un objet de type Double
```

4° Le transtypage ou cast.

Lorsqu’une variable doit changer de type, il faut la transtyper.
Exemple

```
public static double sinus(float f1)
      {
      double resultat;
      resultat = Math.sin((double)f1);
      return resultat;
      }
```

La méthode sin de la classe Math n’accepte qu’un double comme argument, alors que la variable f1 passée en paramètre à la fonction est un float.
Le float f1 est transtypé en double par la syntaxe (double)f1.

5° Les opérateurs et les expressions

Une expression est une affectation, un test, un calcul, etc …. .
Une instruction est expression qui se termine par un point-virgule ;

Exemples :

```
a= b*c ;
for(i=0 ; i<5 ; i++)
        {
          x-- ;
          g+=25 ;
        }
```

Une classe, une fonction, un bloc de traitement etc ….sont délimités par des accollades :

```
{
  ……..
  ……..
}
```

Un commentaires est placé entre /* …….. commentaire …….. */ sur une ou plusieurs lignes, ou bien derrière // ……. commentaire sur une seule ligne.

Signification des opérateurs.

Opérateur	Catégorie	Signification	Exemple
++	Arithmétique	Post ou pré incrémentation	i++ ; --a ;
--		Post ou pré décrémentation	g-- ; --f ;
+		Addition	a=b+c ;
-		Soustraction	a=b-c ;
+		Plus unaire	+a ;
-		Moins unaire	-a ;
*		Multiplication	a=b*c ;
/		Division	a=b/c ;
%		Modulo (reste de la division)	a=c%b
		Expressions condensées Valables pour les opérations Arithmétique et logiques	a+= 5 ; pareil que a=a+5 ; c*=10 ; pareil que c=c*10 ; b &=3 ; pareil que b=b&3 ;
(type)		Transtypage ou cast	a=(char)(c+b) ; a est un char c et b ne sont pas des char
+	Chaînes de caractères	Concaténations	S= « oui »+ « et » + « non »
<<	Logique	Décalage à gauche	
>>>		Décalage à droite avec 0 à gauche	
>>		Décalage à droite	
!		Complément logique	
&		ET logique (bit à bit)	
|		OU logique (bit à bit)	
^		OU logique exclusif (bit à bit)	
=	Affectation	Affectation	A=25 ;
==	Tests	Test d'égalité	if(a == b)
!=		Test de différence	if(a != b)
&&		ET sur des expressions	if(a==b && c !=d)
||		OU sur des expressions	if(a==b || c !=d)
>		Supérieur	
<		Inférieur	
>=		Supérieur ou égal	
<=		Inférieur ou égal	

Les priorités des opérateurs sont de gauche à droite en cas d'égalité.
Les parenthèses sont plus prioritaires.
Pour les opérateurs arithmétiques, division et multiplication sont plus prioritaires que addition et soustraction.

Tableau de précédence (priorité) des opérateurs. Les opérateurs sur une même ligne ont la même priorité, et sont rangés ligne par ligne du plus prioritaire au moins prioritaire.

OPERATEUR	ASSOCIATIVITE	DESCRIPTION
`. ( ) [ ] new`	de gauche à droite	Opérateurs primaires
`! ~ ++ -- + - (cast)`	de droite à gauche	Opérateurs unaires
`* / %`	de gauche à droite	Multiplication, division, reste
`+ -`	de gauche à droite	Addition, soustraction
`<< >> >>>`	de gauche à droite	Décalages
`< <= > >= instanceof`	de gauche à droite	Comparaisons
`== !=`	de gauche à droite	Egalité, différence
`&`	de gauche à droite	Opérateur ET bit à bit
`^`	de gauche à droite	Opérateur OU EXCLUSIF bit à bit
`|`	de gauche à droite	Opérateur OU bit à bit
`&&`	de gauche à droite	Opérateur ET
`||`	de gauche à droite	Opérateur OU
`? :`	de gauche à droite	Condition
`= *= /= %= += -= <<= >>= >>>= &= ^= |=`	de droite à gauche	Affectation

L'opérateur `new` a la même priorité que l'opérateur `.` , ce qui permet d'écrire directement

```
new Classe1().methode1 ()
```

6° Contrôles de programme : Tests et boucles.

Tests :

if if ….. else switch …. case

Exemple 1 :

```
if ( c == 25 )
 {
   System.out.println("C vaut  25") ;
 }
```

Exemple 2 :

```
if ( c == 25 )
 {
   System.out.println("C vaut 25") ;
 }
else
 {
   System.out.println("C ne vaut  pas 25") ;
 }
```

Exemple 3 :

```
  int c ;
  c= ….. ;
  switch ( c )  {
            case 125 : System.out.println("C vaut 125") ;break ;
            case 250 : System.out.println("C vaut 250") ; break ;
            case 375 : System.out.println("C vaut 375") ; break ;
             default : break ;
             }
```

Boucles

for while do ….. while .

Avec while le test est fait à l'entrée de la boucle, alors qu'il est fait à la sortie de la boucle avec do …. while.

Une boucle for peut très souvent être remplacée par une boucle while ou do … while.

Exemple 1 :

```
for(i=0 ; i<5 ; i++)
{
  System.out.println("bonjour") ;
}
```

Exemple 2 :

```
i=0 ;
do
{
  System.out.println("bonjour") ;
   i++ ;
}while(i<5 );
```

Exemple 3 :

```
i=0 ;
while( i < 5)
{
  System.out.println("bonjour") ;
   i++ ;
}
```

Dans les 3 exemples, le résultat sera le même.

7° La classe String pour les chaînes de caractères

Il faut étudier la classe String pour connaître les principales méthodes utiles pour manipuler les chaînes de caractères.

```
String  s1 = "bonjour tout le monde" ;
String s2 = new String("bonjour tout le monde") ;
String s3 ;
s3 = "bonjour tout le monde" ;
```

Tous les objets String précédents contiennent la même chaîne.

La classe String contient de nombreuses méthodes qui méritent d'être étudiées.

Comparaison de 2 String

Il faut utiliser la méthode **compareTo()** :

```
if  (s1.compareTo(s2) ==0)
        System.out.println("Chaines identiques") ;
else    System.out.println("Chaines différentes") ;
```

Longueur d'une chaîne de caractères

Il faut utiliser la méthode **length()** :

```
int  nb = s1.length() ;
```

La conversion d'un String en un autre type

La méthode **String.valueOf()** est surchargée, elle peut reçoivoir un argument de type int, ou float… et retourne l'objet String correspondant.

```
int  i  = 10 ;
String s = String.valueOf(i) ;
```

La concaténation de 2 String s'effectue avec l'opérateur +

```
String  s1 = "bonjour monsieur " ;
String s2 = new String("javanais") ;
String s3 = s1 + s2;
```

Il est possible de rechercher dans un String un caractère particulier, une sous chaîne…

8° Les tableaux.

Ils représentent un ensemble de données de même type, à une ou plusieurs dimensions. On peut se les figurer comme des cases adjacentes dans lesquelles se trouvent des données, chaque case étant repérée par son indice.

En Java, les tableaux doivent être déclarés.
La création d'un tableau s'effectue
– à sa déclaration s'il est initialisé en même temps,
– ou bien ensuite créé avec l'opérateur **new**, c'est alors un objet dynamique, dont la taille

n'est précisée qu'au moment de la création, ce qui est avantageux.

La numérotation des cases des tableaux par des indices commence par l'indice 0.

Exemple de syntaxe
Les expressions
int tab[] ;
ou
int [] tab ; sont équivalentes pour déclarer un tableau d'entiers.

Compteur lescompteurs[] ; déclare un tableau de compteurs.

Le tableau doit ensuite être créé avec l'opérateur new :
tab = new int [10] ;
Lescompteurs = new Compteur[5] ;

Exemple

```
class tableaux1
  {

   static int t1[]={ 20,30,80,10,50,100,90,40,60,200 };
   static int []t2;

      public static  void main(String[] args)
      {

        /* On pourrait ici utiliser des tableaux locaux à la
méthode main comme indiqué ci-dessous

       int t1[]={20,30,80,10,50,100,90,40,60,200};
       int []t2;
       */

       int i;
       // Le tableau t1 est déjà initialisé à sa création
       for( i=0;i<10;i++)
         System.out.println(t1[i]);

        /*Le tableau t2 est créé avec la taille qu'il faut et
initialisé à 0*/

       t2 = new int[10];
       for( i=0;i<10;i++)
         System.out.println(t2[i]);

      // suite de la méthode main
      // ..............

      }
 }
```

Les tableaux t1 et t2 sont des tableaux d'entiers à une dimension.

Le tableau t1 est initialisé aussitôt déclaré, le tableau t2 est d'abord déclaré sans taille fixe puis créé avec une taille de 10 par l'opérateur **new**.

Deux syntaxes sont possibles pour déclarer des tableaux : **int t1[] ; ou int []t1 ;**
int t3[][][] ; ou int [][][] t3 ; tableaux d'entiers à 3 dimensions .

Taille d'un tableau
Un tableau est un objet, sa propriété **length** contient sa taille. On récupère sa taille de la manière suivante :

```
int lg ;
---
lg = t2.length ;
```

9° Exercices

1. **Une première classe Tableau.**
   On demande de créer une classe TableauX dont le squelette est le suivant :

```
public class TableauX {
        int []Tab ;                                  // Le tableau
        int N ;                                      // la taille du tableau
        public Tableau(int N) { /* A compléter */)   // le constructeur
        public void afficheTab() { /* A compléter */) //affiche le tableau
        public void saisieTab()  { /* A compléter */) //pour initialiser le tableau
        public int  getPlusGrand()  { /* A compléter */)
        public int  getPlusPetit()  { /* A compléter */)
        public bool rechecheNombre(int n)  { /* A compléter */)
        }
```

   Le constructeur reçoit comme argument la taille du tableau à créer.
   La méthode **void afficheTab()** affiche tous les éléments du tableau.
   La méthode **void saisieTab()** demande et saisi au clavier les entiers à ranger dans le tableau.
   La méthode **int getPlusGrand()** retourne le plus grand nombre du tableau.
   La méthode **int getPlusPetit()** retourne le plus petit nombre du tableau.
   La méthode **bool rechecheNombre(int n)** recherche si l'argument n est présent dans le tableau, elle retourne true dans ce cas, false si n n'est pas trouvé dans le tableau.

   Coder la classe TableauX.
   Coder une classe TestTableauX qui ne contient que la méthode main(). Cette méthode main contiendra la création d'un objet de type TableauX et le test de toutes les méthodes de la classe TableauX.

2. **Compléter la classe TableauX avec une méthode permettant de remplir le tableau Tab avc des valeurs aléatoires.**

- Ecrire la méthode remplirTableau() qui remplit le tableau Tab avec des **valeurs aléatoires** comprises entre 0 et 200.

On utilisera pour cela la méthode statique **random()** de la classe **Math**, **Math.random()** renvoie un double compris entre 0.0 et 1.0. Il faut trouver la solution permettant d'obtenir un entier dans l'intervalle [0 200] à partir d'un double compris entre 0.0 et 1.0.
Il est conseillé de visualiser l'aide HTML sur la classe Math de la documentation Java.

- Ajouter une méthode **void triBulle()** à la classe TableauxX, pour qu'elle trie les valeurs du tableau Tab , dans l'ordre croissant.

Principe du tri à bulle : On part d'un tableau avec 10 entiers, les indicces allant de 0 à 9. En partant du bas du tableau non ordonné, indice 9, on compare un élément du tableau avec celui d'indice inférieur. Si le nombre à l'indice supérieur est plus petit que le nombre à l'indice inférieur avec lequel on le compare, on fait « remonter » le plus petit des deux en permutant les 2 nombres. Si l'ordre est déjà bon entre les deux nombres que l'on compare, on ne change rien. Ainsi, au premier passage, on aura fait 9 comparaisons, et le plus petit des 10 nombres sera « remonté à la surface », c'est à dire à l'indice 0.
Il faudra recommencer une deuxième série de comparaisons en repartant du bas du tableau, à l'indice 9. Cette deuxième série sera plus courte que la première, car le nombre à l'indice 0 est déjà à sa place. A l'issue de la deuxième série de comparaisons, les 2 plus petits nombres seront classés dans l'ordre croissant à l'indice 0, et à l'indice 1. On continue ainsi jusqu'au classement des 10 éléments du tableau.

Que se passe-t-il si on tente d'accéder à un élément extérieur au tableau, par exemple Tab[11] ?

3. Créer un tableau de 10 chaînes de caractères non ordonnées.

Deux méthodes de tribulle sur des chaînes de caractères seront membres de la classe :

static void croissant( ) et static void decroissant( )

Afin de trier les chaînes par ordre croissant et par ordre décroissant de l'ordre alphabétique.
On utilisera les méthodes de la classe String. Il est conseillé d'utiliser l'aide HTML de la documentation Java pour étudier la classe **String**. Etudier notamment la méthode **compareTo()** utile pour comparer 2 chaînes de caractères.

Le tableau de String sur lequel porte le tri peut être déclaré en tant qu'attribut de la classe.

4. Créer un tableau d'entiers à 3 dimensions.

On s'imaginera 3 plans en Z de dimensions: 5 en X sur 4 en Y. Le 1er plan est obtenu avec Z=0, le 2ème avec Z=1 et le 3ème avec Z=2.

Il ne faut exécuter qu'un seul « random » dans le programme.

On remplira ce tableau avec des valeurs aléatoires comprises entre 0 et 300 de la façon suivante : Le premier plan du tableau ne contiendra que des nombres inférieurs à 100, le deuxième plan du tableau contiendra des nombres >= 100 et < 200, et le troisième plan du tableau ne contiendra que des nombres supérieurs à 200.

L'algorithme de base du traitement d'une valeur est le suivant :

Tirer une valeur N aléatoire
Si N>=0 && N<100
Ranger N dans la 1ère case vide du 1er plan
Si N>=100 && N<200
Ranger N dans la 1ère case vide du 2ème plan
Si N>=200 && N<300
Ranger N dans la 1ère case vide du 3ème plan

10° La boucle foreach

La boucle **foreach** se traduit en Java de la façon suivante :

for (**Type** NomVar **:** TableauDeType) {
//Utilisation de la variable Nomvar
}
L'itération sur le tableau est faite automatiquement.
Exemple :

```
public class TableauForeach {
  public static void main(String args[]) {
    String[] datas = { "Paris", "Stockholm","Rome","Berlin" };
    for (String s : datas) {
      System.out.println(s);
    }
  }
}
```

11° Le mot réservé final

Définir une constante : on utilise les modificateurs **static** et **final**.
Exemple :

```
        static final double PI = 3.141592653589793;
```

Définir une méthode final :
Une méthode **final** ne peut être redéfinie, voir le chapitre 4 pour la redéfinition, c'est-à-dire qu'elle exécutera toujours le même code.

```
class ChessAlgorithm {
    enum ChessPlayer { WHITE, BLACK }
    ...
    final ChessPlayer getFirstPlayer() {
        return ChessPlayer.WHITE;
    }
```

```
    ...
}
```

12° L'opérateur instanceof

L'opérateur **instanceof** permet de savoir si un objet donné est une instance d'une classe donnée.
Il retourne true si l'objet considéré est une instance de la classe donnée.
Exemple :

```
Parent obj1 = new Parent();
System.out.println("obj1 instanceof Parent: " + (obj1 instanceof Parent));
System.out.println("obj1 instanceof Abonne: " + (obj1 instanceof Abonne));
```

Affichage obtenu :

```
"obj1 instanceof Parent: true"
"obj1 instanceof Abonne: false"
```

13° La class java.util.Arrays

La classe **java.util.Arrays** est très pratique pour les tableaux car elle propose des méthodes statiques pour trier les tableaux, rechercher une valeur particulière….

Exemple :

```
import java.util.Arrays;

public class Array1 {

        public static void affiche(int []t){
                for( int i=0;i<10;i++)
                                System.out.print(t[i]+ "  ");
                        System.out.println();
        }
        public static void main(String[] args) {
                int []t = null;
                t = new int[10] ;

                for (int i=0 ; i<10; i++) t[i] = (int)(50*Math.random()) ;

                affiche(t) ;
                Arrays.sort(t);
                affiche(t) ;

                int res = Arrays.binarySearch(t, 5);

                System.out.println(res);
        }
}
```

Résultat d'une 1ère exécution:

```
27  13  16  44  12  23  31  5  41  0
0  5  12  13  16  23  27  31  41  44
2
```

1ère ligne : affichage du tableau non trié.
2ème ligne: affichage du tableau trié.
3ème ligne : la valeur recherchée « 5 » est en 2ème position dans le tableau, res vaut 2.

Résultat d'une 2ème exécution:

```
1  26  32  26  2  49  30  33  32  36
1  2  26  26  30  32  32  33  36  49
-3
```

La valeur recherchée n'est pas trouvée dans le tableau. Si elle était présente cette valeur devrait être insérée en 3ème position (après le « 2 »). res vaut ici -3, la valeur absolue du résultat donne le rang qu'elle devrait avoir, la valeur négative indiquant qu'elle n'est pas présente dans le tableau.

La méthode **Arrays.sort()** est surchargée pour être utilisée pour tous les types de base.

Il est possible de l'utiliser pour trier un tableau d'objets mais il faut que cette classe d'objet implémente l'interface comparable et définisse la méthode **compareTo()**, seule méthode de cette interface.

```
import java.util.Arrays;
public class Compteur implements Comparable
{
public int valeur ;

/* 1er constructeur de la classe  Compteur */
public Compteur() {
        valeur = 0 ;
System.out.println("Je suis le constructeur sans argument") ;
        }
/* 2ème constructeur de la classe  Compteur */
public Compteur(int n) {
        valeur = n ;
System.out.println("Je suis le constructeur avec un argument") ;
        }
public int compareTo(Object o) {
        Compteur c = (Compteur)o;
        return  (valeur - c.valeur )  ;
}
public void affiche() {
        System.out.println("Valeur du compteur = "+valeur) ;
        }
public static void main(String argv[])
        {
        Compteur lescompteurs[] = new Compteur[10] ;
```

```
        for (int i = 0; i<10; i++){
                int n = (int)(50*Math.random()) ;
                lescompteurs[i] = new Compteur(n);
        }
        System.out.println("Avant le tri:") ;
        for (int i = 0; i<10; i++)lescompteurs[i].affiche() ;

        Arrays.sort(lescompteurs);

        System.out.println("\nAprès le tri:") ;
        for (int i = 0; i<10; i++)lescompteurs[i].affiche() ;
        }
}
```

Cela donne l'affichage suivant :

```
Avant le tri:
Valeur du compteur = 43
Valeur du compteur = 47
Valeur du compteur = 37
Valeur du compteur = 8
Valeur du compteur = 39
Valeur du compteur = 19
Valeur du compteur = 30
Valeur du compteur = 19
Valeur du compteur = 31
Valeur du compteur = 12

Après le tri:
Valeur du compteur = 8
Valeur du compteur = 12
Valeur du compteur = 19
Valeur du compteur = 19
Valeur du compteur = 30
Valeur du compteur = 31
Valeur du compteur = 37
Valeur du compteur = 39
Valeur du compteur = 43
Valeur du compteur = 47
```