Le langage Java - Syntaxe

LES BASES: litéraux, types, expressions, instructions

LE LANGAGE JAVA - SYNTAXE

LES COMMENTAIRES

LES IDENTIFICATEURS

LISTES DES MOTS RÉSERVÉS:

LES LITÉRAUX

BOOLÉENS

LA DÉCLARATION DES VARIABLES

TYPE SIMPLE VS COMPOSÉ

VECTEURS ET MATRICES

VISIBILITÉ DES VARIABLES

OPÉRATEURS

TABLE DE PRÉCÉDENCE

L'ASSIGNATION

EXPRESSION...

OPÉRATEUR D'ALLOCATION

CONVERSION DE TYPE

Les commentaires

Trois manières de créer des commentaires en JAVA.

/* ... */.

```
/* ceci est un commentaire
   sur plusieurs lignes
... qui se termine ici */
```

//...

```
int i // ceci est une variable entière
```

/** et */.

Ce commentaire est à réserver dans les déclarations en vue d'une documentation automatique

bonnes habitudes

Il faut absolument commenter les programmes.

- écrit une seule fois
- relu des dizaines de fois,
- une reformulation de la spécification du programme

mauvais commentaire

```
int zoom=2 // zoom à 2
```

aucun renseignement sur le rôle de zoom et pourquoi 2.

commentaire OK

```
int zoom=2 // valeur par défaut du zoom au démarrage
```

Les identificateurs

nommer:

- les variables,
- les classes
- les méthodes
- les packages, ... des programmes JAVA.

identifier =
"dans {a..z, A..Z, $, _}"
< " dans {a..z,$,_,0..9,unicode character over 00C0}" > .

-
- des caractères a..z ou A..Z,
- des chiffres de 0 à 9
- les _ et $
- et les caractères Unicode supérieure à 0X00C0. caractères nationaux tels que: Ç, ü, ...

exemples d'identificateur valides:

```
$valeur_system
dateDeNaissance
ISO9000
```

exemples d'identificateur non-valides:

```
ça          // Ç comme premier caractère
9neuf       // 9 comme premier caractère
note#       // # pas au dessus de 0X00C0
long        // OK mais c'est un mot réservé!
```

listes des mots réservés:

Abstractboolean	break	byte	byvalue[1]	casec
atchchar	class	const[2]	continue	default
dodouble	else	extends	false	finally
floatfor	goto[2]	if	implementsimport	
instanceofint	interface	long	native	newnull
packageprivate	protected	public	retrun	short
staticsuper	switch	synchronized	this	
threadsafethrow	transient	true	try	void
while				

Donc un identificateur ne doit pas être un mot de cette liste.

1. réservé, mais pas actuellement utilisé

bonnes habitudes

ne pas utiliser le $ et le _ si vous devez utiliser des librairies en C;

de ne pas utiliser le $ en première position;

de séparer les noms composés en capitalisant la première lettre des noms à partir du deuxième mot.

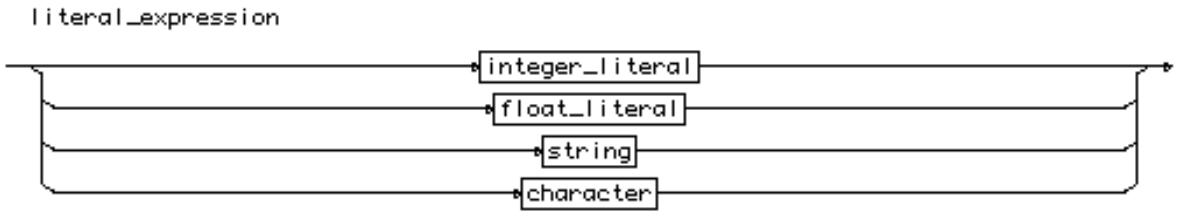
conseillé	acceptable	déconseillé
`nomDeMethode`	`nom_de _methode`	`$init`
`ceciEstUneVariable`	`ceci_est_une_variable`	`_type`

Les litéraux

Les litéraux définissent explicitement les valeurs sur lesquelles travaillent les programmes JAVA.

trois catégories de litéraux:

- les booléens,
- les nombres,
- les caractères.



booléens

- false
- true

initialiser des variables booléennes

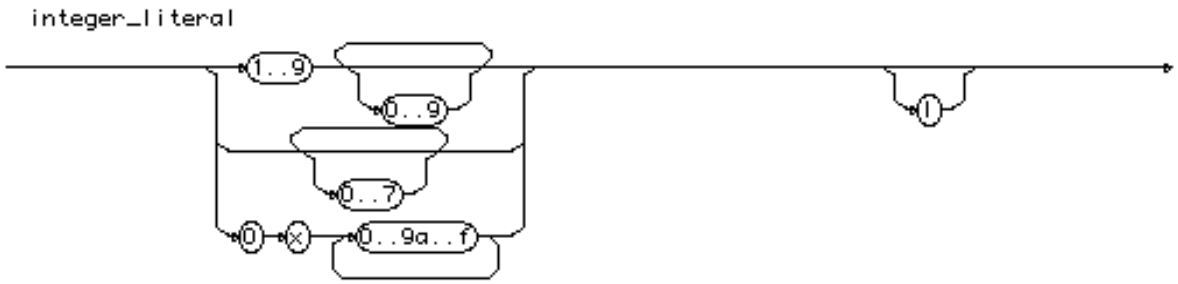
(Attention: ne peut être assimilée à un 0 ou 1 comme dans le cas des langages C ou C++)

```
boolean resultatAtteint = false ;
boolean continuer = true ;
```

entiers

trois formats:

- en décimal,
- en octal,
- hexadécimal.
-



en décimal ne commence jamais par un zéro.

```
int nbrDeMois = 12;
```

Pour en octal, précédé d'un zéro.

```
int nbrDeDoigts = 012;    // =10 en décimal
```

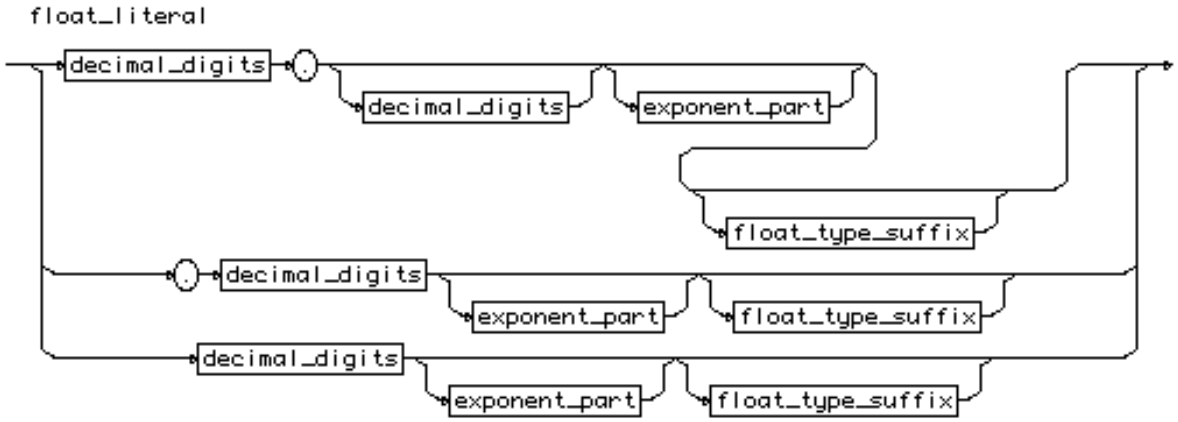
en hexadécimal, précédé d'un 0x ou 0X.

```
int dixHuit= 0x12;
```

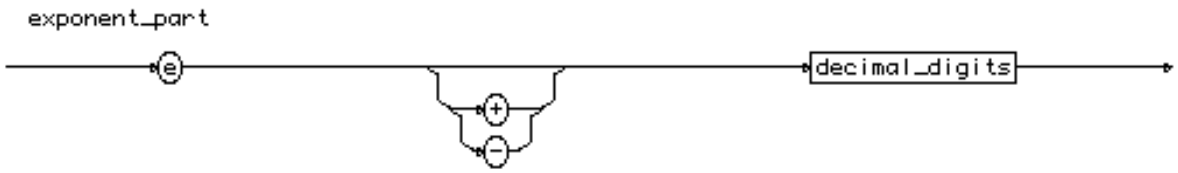
entiers déclarés littéralement = *int* sur 32 bits.
entier *long* en lui faisant succéder un L (64 bits).

flottants

- une mantisse
- éventuellement exposant en puissance de 10.
- obligatoirement un point décimal ou un exposant



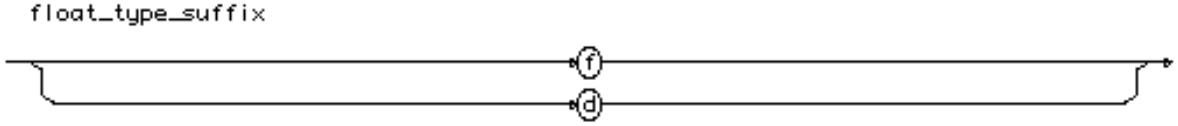

```
Exemple de flottants sans partie exposant:
  2..52.52.0  .001
Exemple de flottants avec partie exposant:
  2E02E32E-3.2E32.5E-3
```



Les nombres flottant déclarés = *float* sur 32 bits.
flottant *double* précision, succéder un D (64 bits)

flottant *float* , succéder un F (32 bits)

```
3.14F3.14159D
```



caractères

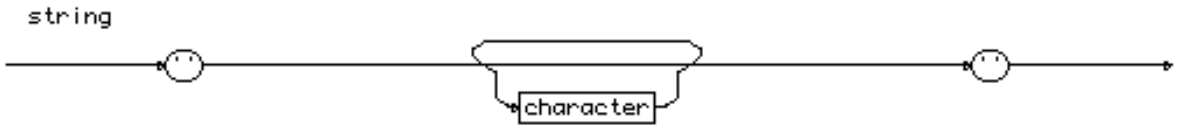
caractère / deux apostrophes (quotation simple):
‘x’‘a’‘4’

l’ensemble des caractères Unicode (16bits)

caractère	abréviation	séquence
continuation	<nouvelle ligne>	\
Nouvelle ligne	N	\n
tabulation	HT	\t
retour arrière	BS	\b
retour chariot	CR	\r
saut de page	FF	\f
backslash	\	\\
apostrophe	‘	\’
guillemet	"	\"
caractère octal	0377	\377
caractère hexadécimal	0xFF	\xFF
caractère Unicode	0xFFFF	\uFFFF

chaînes de caractères

- suite de caractères entourée de guillemets,
- type *String*.
- constitue une classe
- ≠ un vecteur de caractères.
-



la chaine de caractère	le résultat si on l'imprime
""	
"\""	"
"texte sur 1 ligne"	texte sur 1 ligne
"texte sur 1 ligne \défini sur 2 lignes"	texte sur 1 ligne défini sur 2 lignes
"texte sur 1 ligne \n écrit sur 2 lignes"	"texte sur 1 ligne écrit sur 2 lignes"

La déclaration des variables

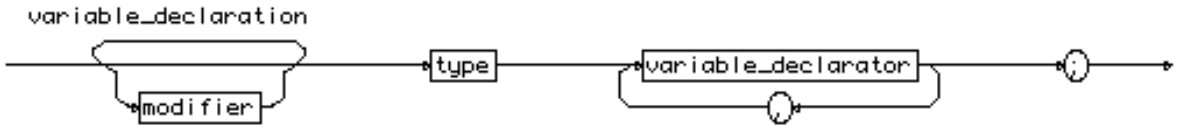
obligatoirement une déclaration.

Le compilateur

- vérifications de compatibilité de type dans les expressions, visibilité de la variable, etc.

augmenter la qualité des programmes

- détecter des erreurs à la compilation
- détecter au moment de l'exécution.



préfixer le nom de la variable par son type

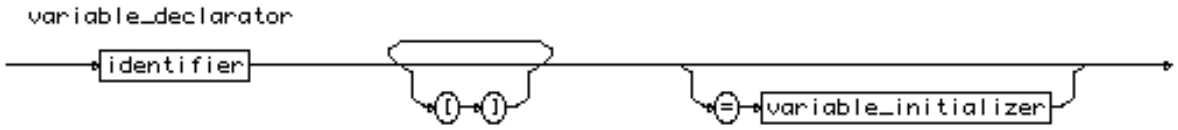
```
int i;
```

```
int i,j,k;
```

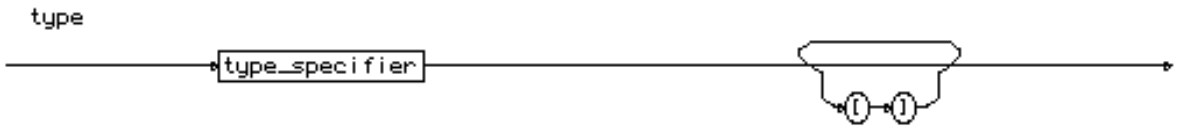
modifier la définition de la variable

final = une constante.

```
final int NbreDeRoues=4;
final float pi=3.14159;
```



Type simple vs composé

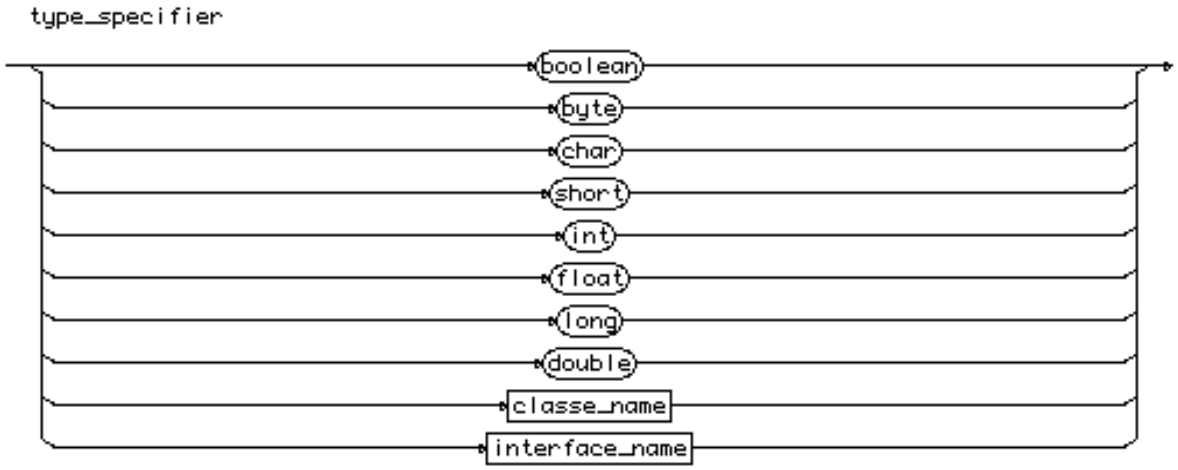


Les *types simples*

- les booléens,
- les entiers
- ...
-

types composés, construits à partir d'autres types

- les vecteurs,
- matrices,
- classes,
- interfaces.
-



booléens

définit une variable dont le contenue sera vrai ou faux.

assignée

- *false*
- *true*
- résultat d’une expression logique

```
boolean voitureArretee=true;
```

entiers

quatre types d'entier:
byte,
short,
int
long.

type entier	nombre de bits	exemples
byte	8	byte NbrEnfant;
short	16	short VolumeSon;
int	32	int i,j,k;
long	64	long detteSecu;

assignée

- les littéraux entiers
- le résultat d'une expression entière.

flottants

deux types de flottant:

- *float*
- *double*

type flottant	nombre de bits	exemples
float	32	float ageMoyen;
double	64	double pi;

assignée

- les littéraux flottant
- résultat d’une expression flottante.

caractères

une valeur de l’ensemble Unicode de caractères

assignée

- les littéraux caractère
- le résultat d’une expression caractère.

```
// le séparateur est un tabulateur
char separateur="\t";
```

Le caractère en JAVA est représenté sur 16 bits.

Il ne sert qu’à conserver la valeur d’un seul caractère.

Les chaînes sont représentées par la classe *String*.

vecteurs et matrices

post-fixer le type ou la variable par [ ].:

```
int i[ ];   // vecteur d'entiers
int[ ] j;   // j à le même type que i
char motsCroises[ ][ ];   // une matrice de caractères
```

pas contraints au moment de la déclaration.

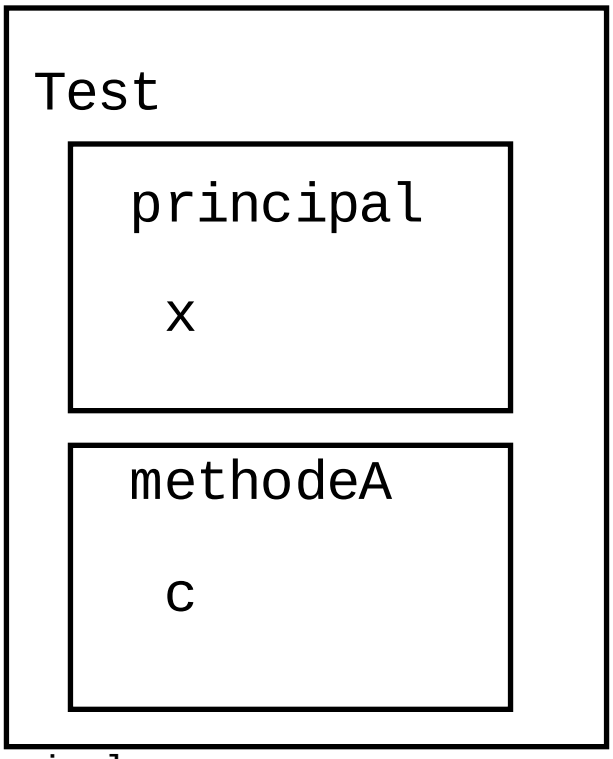
L'allocation au moyen d'une méthode *new*

visibilité des variables

La variable est visible à l'intérieur du bloc où elle est définie.

un bloc est définit comme l'ensemble des instructions comprises entre deux accolades { } .

```
class Test { // debut de test
   public static void principal(String args[])
      { // debut de principal
        float x
        ...
       } // fin de principal
   public void methodeA ()
      { // debut de methodeA
        char c
        ...
       } // fin de methodeA
```

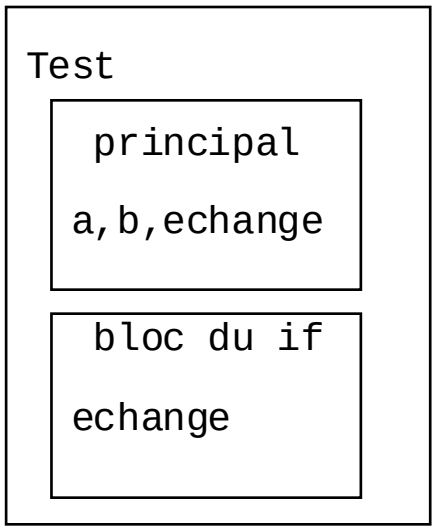



```
} // fin de principal
```

redéfinition des variables

si dans un bloc on redéfinit une variable existant dans un bloc supérieur, cette nouvelle variable masque la variable supérieur à l'intérieur de ce bloc (donc aussi pour ses sous-blocs).

```
class Test { // debut de test
   public static void principal(String args[])
      { // debut de principal
        float echange; int a,b;
        ...
        if a < b then
          { int echange;
            echange=a;a=b;b=echange
          };
       } // fin de principal
   public void methodeA ()
} // fin de principal
```



- réservée à des variables temporaires
- indice d’une boucle.

Opérateurs

regroupés par type d'opération:

- numérique,
- de comparaison,
- logique,
- sur les chaînes de caractères,
- de manipulations binaires.

le nombre d'opérandes :

- unaire,
- binaire
- ternaire.

évaluées de la gauche vers la droite,
une table de précédence ,la priorité entre les opérations.

```
x=z+w-y/(3*y^2)
```

on a = + / () par ordre de lecture
La table de précédence => () / + - =.

Dans la (), on a * ^ par odre de lecture,
l'ordre d'exécution est aussi *^.

table de précédence

(de la plus haute à la plus basse).
Les opérateurs de même niveau depuis la gauche.

.	[]	()		
++	--	!	~	instanceof
*	/	%		
+	-			
<<	>>	>>>		
<	>	<=	>=	
==	!=			
&				
^				
&&				
||				
?:				
=	op=			
,				

l’assignation

L’assignation assigne l’expression de droite, à l’expression de gauche (une variable).

opérateur binaire qui modifie son opérande gauche.

```
j=2; // expression d’assignation d’un litéral
i=j*3; // expression d’assignation d’une expression
i=j*3(j=2); // expression valide combinant les deux
```

Cette dernière expression a pour résultat i=6!

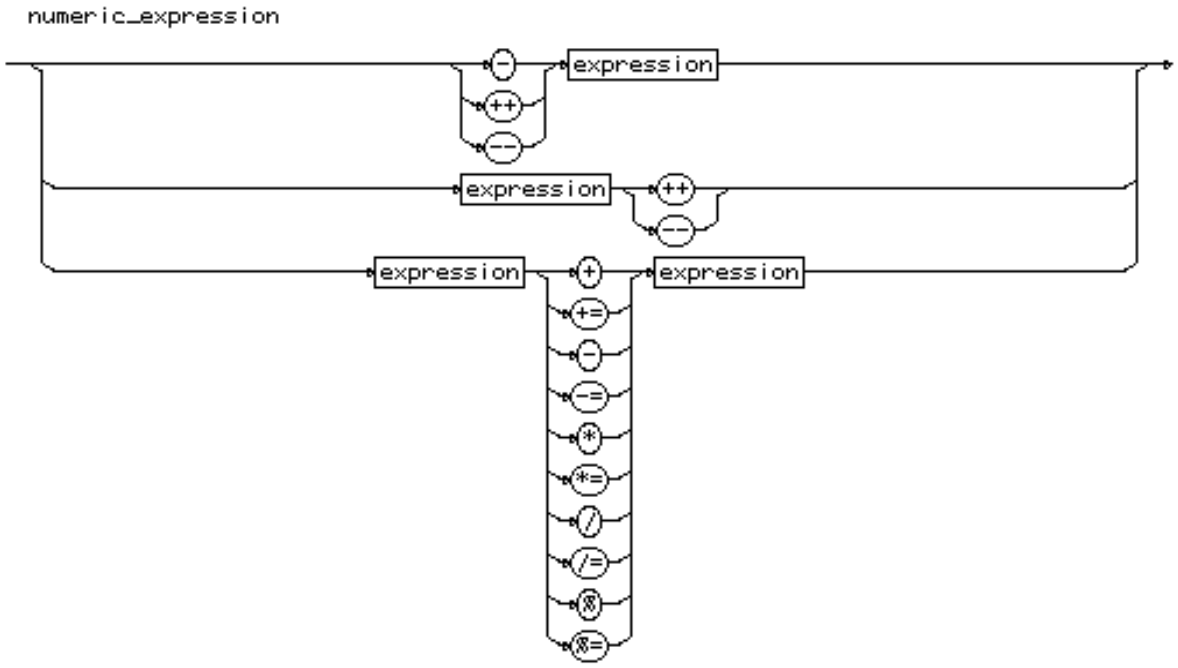
effets de bord: ~, ++ , --

dans une même expression rend confus la lecture.

décomposer les expressions un

seul opérateur avec effet de bord.

expression numérique



les opérateurs unaires	action	exemple
-	négation	`i=-j;`
++	incrémentation de 1	`i++;`
--	décrémentation de 1	`i--;`

++ et -- peuvent préfixer ou postfixer la variable.

```
++i;  // est équivalent à i++;
```

expression numérique ...

les opérateurs binaire	action	exemple
+	addition	i=j+k
+=		i+=2; // i=i+2
-	soustraction	i=j-k;
-=		i-=j; // i=i-j
*	multiplication	x=2*y
*=		x*=x; // x=x*x
/	division (tronque si les arguments sont entiers)	i=j/k;
/=		x/=10; // x=x/10
%	modulo	i=j%k;
%=		i%=2; // i=i%2

la division par 0 et le modulo par 0 génère une exception à l'exécution.

Les opérations dont la représentation dépasse le maximun du type de l'entier débordent dans les entiers négatifs.

Pour les flottant, les opérations même sémantique.

++ et -- on ajoute 1.0.

Le % modulo appliqué au flottant prend le sens de la division dans les entiers.

la division par 0 et le modulo par 0 génère la valeur *inf*. Les opérations dont la représentation dépasse le maximun du type génère la valeur inf.

Les débordements vers l'infinement petit génère 0.

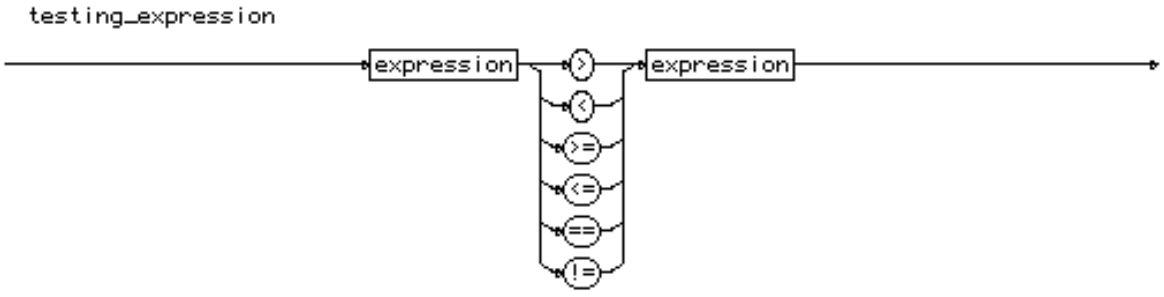
Le programme TestNumber

```
class TestNumber{
    public static void main (String args[]) {
        int i=1000, j=1000;
        float x=1, y=1;
        for (int k=0; k<100;k++) {
           i*=10;
           j/=10;
           x*=10000;
           y/=10000;
      System.out.println("\ni="+i+" j="+j+" x="+x+" y="+y);
        }
    }
}
```

```
i=10000 j=100 x=10000 y=0.0001
i=100000 j=10 x=1e+08 y=1e-08
i=1000000 j=1 x=1e+12 y=1e-12
i=10000000 j=0 x=1e+16 y=1e-16
i=100000000 j=0 x=1e+20 y=1e-20
i=1000000000 j=0 x=1e+24 y=1e-24
i=1410065408 j=0 x=1e+28 y=1e-28
i=1215752192 j=0 x=1e+32 y=1e-32
i=-727379968 j=0 x=1e+36 y=1e-36
i=1316134912 j=0 x=Inf y=9.99995e-41
i=276447232 j=0 x=Inf y=9.80909e-45
i=-1530494976 j=0 x=Inf y=0
i=1874919424 j=0 x=Inf y=0
...
```

Opérateurs relationnels

tester tous les types avec une relation d'ordre:
entiers,
réels,
caractères, ...



les opérateurs relationnels	action	exemple
<	plus petit que	`x<i;`
>	plus grand que	`i>100;`
<=	plus petit ou égal que	`j<=k;`
>=	plus grand ou égal que	`c>='a';`
==	égal à	`i==20;`
!=	différent de	`c!='z';`

ATTENTION relation d'ordre sur les flottants,
x==y peut être vrai
y<x || y>x soit forcement vrai.

l'opérateur == confondu avec =

opérateurs logiques

opérandes booléennes.

Les quatres opérateurs de base sont:

- ! la négation,
- & le ET,
- | le OU
- ^ le OU exclusif.

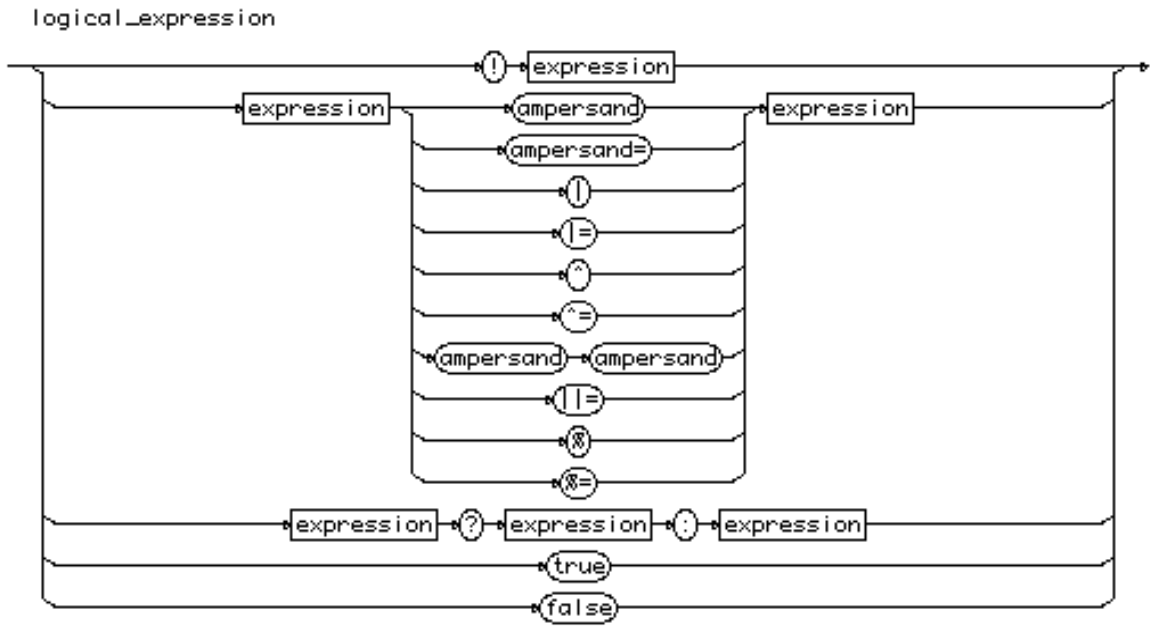


Table de vérité des opérateurs logiques

p	q	!p	p&q	p|q	p^q
v	v	f	v	v	f
v	f	f	f	v	v
f	v	v	f	v	v
f	f	v	f	f	f

opérateurs logiques ...

forme assignée (sauf pour la négation) donc:

```
p&=(i=10); // est équivalent à p=p&(i<10)
```

évaluation raccourcie pour le ET et le OU: && et ||.

L'évaluation de l'expression logique est stoppée dès que sa valeur de vérité peut être assurée.

```
p1 && p2 && ... & pn
```

l'évaluation est stoppée si *pi* est évaluée à faux. Dans le cas:

```
p1 || p2 || ... || pn
```

l'évaluation est stoppée si *pi* est évaluée à vrai.

permet d'éviter une erreur

```
i!=0 && x > ( y/i);// y/i n'est pas évalué si i égale 0
```

JAVA possède aussi une expression ternaire de la forme:

```
p?e1:e2;
```

Si p est vrai alors l'expression e1 est évaluée sinon e2 est évaluée.

l'assignation étant une expression, il est possible de l'utiliser comme une instruction *if*.

```
if (p) e1; else e2; // est équivalent à p?e1:e2;
```

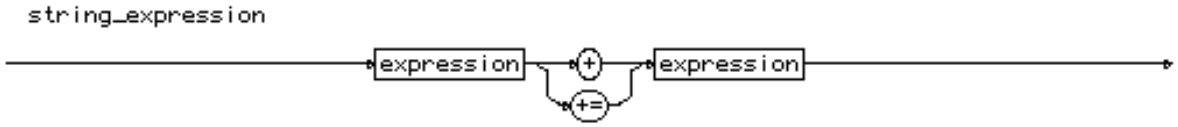
opérateurs logiques ...

les opérateurs logiques	action	exemple
!	négation	`!p;`
&	ET	`p & (i<10)`
|	OU	`p | q`
^	OU exclusif	`p ^ false`
&&	ET évalué	`p && q && r`
||	OU évalué	`p || q || r`
!=	négation assignée	`p!=p;`
&=	ET assigné	`p&=q  // p= p & q`
|=	OU assigné	`p|=q  // p= p | q`
?:	Si alors sinon	`(i<10)?(j=2):(j=3)`

Les opérateurs !, &, |, et ^ sont étendus à des opérations de manipulation binaire sur les bits des opérandes entières.

Pour chaque bit des entiers p et q, on utilisera la table de vérité des opérations, en substituant à vrai 0 et à faux 1.

opérateurs sur les chaînes de caractères



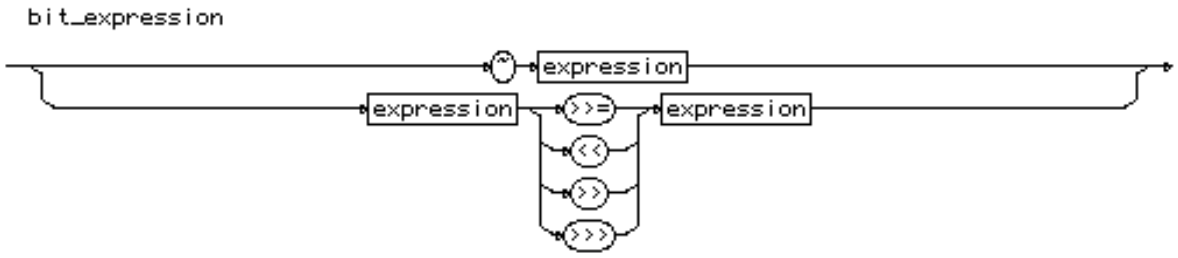
Concaténation +
assignation +=

Les opérandes sont automatiquement converties.

```
System.out.println("\ni="+i+" j="+j+" x="+x+" y="+y);
```

les opérateurs de concaténation	action	exemple
+	concaténation	"conca"+"ténation";
+=	ajoute à la fin	s+=" à la fin"; // s=s+" à la fin"

opérateurs de manipulation binaire



manipulations &, |, ^

décalages binaires (deux opérandes):

- valeur binaire sur laquelle on effectue le décalage
- le nombre de décalages à effectuer. On a donc:

```
i>>k; // decaler i vers la droite de k bits avec son signe
i<<k; // decaler i vers la gauche de k bits
i>>>k; // decaler i vers la droite de k bits sans signe
```

version assignée.

les opérateurs de manipulation binaire	action	exemple
<<	décalage à gauche	i<<n;
>>	décalage à droite signé	i>>4;
>>>	décalage à droite non signé	i>>>2;
<<=	décalage à gauche assigné	k<<=n;
>>=	décalage à droite signé assigné	k>>=n;
>>>=	décalage à droite non signé assigné	k>>>=n;

Equivalence ...

opération de manipulation binaire

les opérations arithmétiques sur les entiers.

Pour i entier, on a:

~iest équivalent à(i-1)-1

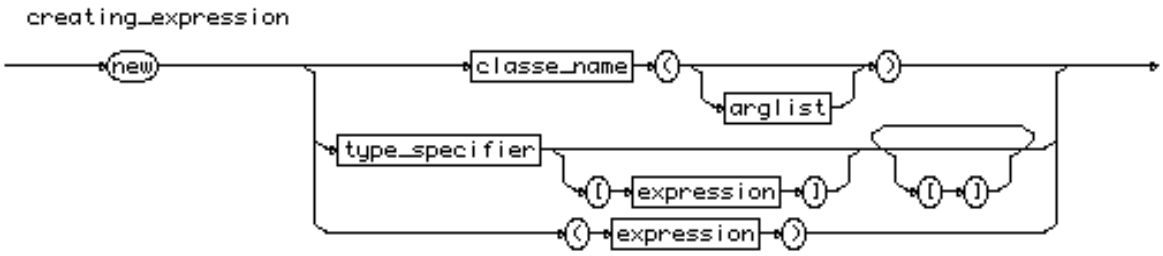
i>>kest équivalent ài/2k si i>0

i<<kest équivalent ài*2k

i>>>kest équivalent ài/2k si i>0

i>>>kest équivalent à(i<<k)+(2<<(L-k-1) si i<0
où L=32 si i est int et L=64 si i est long;

opérateur d'allocation



L'opérateur d'allocation *new*

- créer un objet instance d'une classe
- créer et de fixer les dimensions des vecteurs

déclarer un vecteur, pour fixer sa taille:

```
// i un vecteur de 100 entiers
int i[] = new int [100];

// c une matrice de 10 par 10
int c[][] = new char[10][10];
```

Les indices des vecteurs commencent à 0,

```
i[0] = 1; //la première position de i initialisée à 1
i[9] = 10; //la dernière position de i initialisée à 10

c[0][0]='a'; // première case de c
c[9][9]='z'; // dernière case de c
```

opérateur d’allocation ...

Seule la première dimension doit être contrainte pour un type ayant plusieurs dimensions.

```
int c[][] = new char[10][]; //c une matrice de 10 par ?
```

Les autres dimensions déclarée à l’exécution:

```
c[0] = new char [24]; // première ligne de c = 24 caractères
```

Les dimensions variables:

```
c[1] = new char [12]; // deuxième ligne de c = 12 caractères
```

D’une manière générale, une allocation peut être d’une variable de dimension n peut être vue comme une serie de n-1 boucles imbriquées effectuant chaque allocation.

```
int c[][] = new char[10][10];
// est équivalent à:
int c[][] = new char[10][];
for (int i=0; i< c.length; i++)
     //allocation de chaque ligne de c
     c[i] = new char [10];
```

conversion de type

- un caractère=> sa valeur entière. ?
- un entier=> sa valeur comme un ?
- convertir les valeurs des types

int i;

```
char c
c=(int) i // assigne à c un caractère de valeur i
i=(char) c; //assigne à i un entier de la valeur de c
```

faites explicitement en JAVA

possibilité de perdre de l'information:

```
int i;
i == (int) ((float) i); // n'est pas toujours vrai
```

tableau des conversions sans perte

tableau des conversions sans perte							
de à	**byte**	**short**	**int**	**long**	**float**	**double**	**char**
byte	oui	oui	oui	oui	oui	oui	oui
short		oui	oui	oui	oui	oui	
int			oui	oui	perte de précision	oui	
long				oui	perte de précision	perte de précision	
float					oui	oui	
double						oui	
char			oui	oui	oui	oui	oui

Les autres conversions peuvent être effectuées sans perte d'information dans certaines conditions, généralement celles qui restreignent le type le plus large aux valeurs du type le plus étroit.