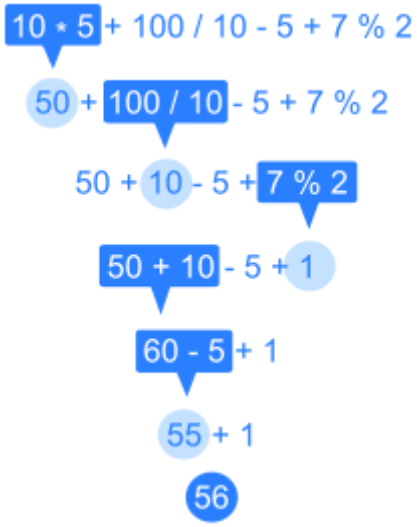
Les expressions sont évaluées en suivant l'ordre de priorité des opérateurs.



Les opérateurs

Le tableau ci-dessous présente la liste des opérateurs utilisables en Java, avec leur signification et leur associativité, dans l'ordre de leur évaluation (du premier au dernier évalué) et le type de données auquel chaque opérateur s'applique. Les opérateurs regroupés entre deux lignes épaisses ont la même priorité.

Opérateur	Description	Type	Associativité
( )	Appel de méthode	*classes et objets*	de gauche à droite
[ ]	Elément d'un tableau	*tableaux*	
.	Membre d'une classe ou d'un objet	*classes et objets*	
++	Incrémentation post ou pré-fixée	`byte char short int long float double`	de droite à gauche
--	Décrémentation post ou pré-fixée	`byte char short int long float double`	
+	Positif	`byte char short int long float double`	
-	Négation	`byte short int long float double`	
!	Non logique	`boolean`	
~	Non binaire	`byte char short int long`	
(*type*)	Conversion de type	*tous*	
*	Multiplication	`byte char short int long float double`	de gauche à droite
/	Division	`byte char short int long float double`	
%	Modulo (reste de la division entière)	`byte char short int long`	
+	Addition	`byte char short int long float double` `String` *(concaténation)*	de gauche à droite
-	Soustraction	`byte char short int long float double`	
<<	Décalage de bit vers la gauche	`byte char short int long`	de gauche à droite
>>	Décalage de bit vers la droite (signe conservé)	`byte char short int long`	
>>>	Décalage de bit vers la droite (signe décalé)	`byte char short int long`	
<	Inférieur	`byte char short int long float double`	de gauche à droite
<=	Inférieur ou égal	`byte char short int long float double`	
>	Supérieur	`byte char short int long float double`	
>=	Supérieur ou égal	`byte char short int long float double`	
==	Egal	`byte char short int long float double` *objet*	de gauche à droite
!=	Différent	`byte char short int long float double` *objet*	
&	ET binaire	`byte char short int long boolean`	de gauche à

			droite
^	OU exclusif binaire	`byte char short int long boolean`	de gauche à droite
|	OU binaire	`byte char short int long boolean`	de gauche à droite
&&	ET logique	`boolean`	de gauche à droite
||	OU logique	`boolean`	de gauche à droite
?:	Opérateur ternaire de condition	`boolean ?` *tous* `:` *tous*	de droite à gauche
=	Affectation	*tous*	de droite à gauche
+=	Addition et affectation	`byte char short int long float double String` *(concaténation)*	
-=	Soustraction et affectation	`byte char short int long float double`	
*=	Multiplication et affectation	`byte char short int long float double`	
/=	Division et affectation	`byte char short int long float double`	
%=	Modulo et affectation	`byte char short int long float double`	
<<=	Décaler à gauche et affectation	`byte char short int long`	
>>=	Décaler à droite (excepté signe) et affectation	`byte char short int long`	
>>>=	Décaler à droite (signe aussi) et affectation	`byte char short int long`	
&=	ET binaire et affectation	`byte char short int long boolean`	
^=	OU exclusif binaire et affectation	`byte char short int long boolean`	
|=	OU binaire et affectation	`byte char short int long boolean`	
,	Enchaînement d'expressions	*tous*	de gauche à droite

Chaque case de la colonne « associativité » regroupe les opérateurs de même priorité.

Expressions

`short` et `byte`

Java effectue une conversion en valeur de type `int` de manière implicite sur les valeurs de type `short` et `byte` dès qu'une opération est effectuée, ce qui peut donner des résultats non conforme à ce que l'on pourrait attendre (détails (http://linformalibre.f2lt.fr/index.php?title=Particularit%C3%A9_des_Op%C3%A9rations_Binaires_sur_les_bytes_et_les_shorts_en_Java)).

Expressions booléennes

Les expressions booléennes employant les opérateurs ET logique (&&) et OU logique (||) sont évaluées de manière paresseuse. C'est à dire que l'évaluation s'arrête aussitôt que le résultat est déterminé.

Exemple avec l'opérateur ET logique (vrai si les deux opérandes sont vrais), si le premier opérande est faux le résultat est faux et le deuxième opérande n'est pas évalué. Si le deuxième opérande est le résultat de l'appel à une méthode, cette méthode ne sera pas appelée :

```
String s = getText();
if ((s!=null) && (s.length()>0))
// si s vaut null, la longueur n'est pas testée car cela
provoquerait une exception.
{ /*...*/ }
```

Parfois ce comportement n'est pas désirable. Dans ces cas-là, il faut utiliser les opérateurs binaires équivalents ET (&) et OU (|).

Le remplacement dans l'exemple précédent serait cependant une erreur :

```
String s = getText();
if ((s!=null) & (s.length()>0))
// si s vaut null, la longueur est tout de même testée et
provoque donc une exception !
{ /*...*/ }
```

Savoir si une racine carrée est entière

La fonction calculant les racines carrées pouvant trouver des nombres à virgules, voici une courte technique pour savoir s'il s'agit d'un entier :

```
public boolean testRacineEntiere(int n)
{
    int r = (int)Math.sqrt(n);
    return (r*r == n);
}
```