Les opérateurs en Java

http://alexandre-mesle.com

22 novembre 2022

Définition

Une **opération** est l'application d'un **opérateur** à des **opérandes**

Exemple

$9 + 12$

- 9 et 12 sont les opérandes
- $+$ est l'opérateur

Définition

Une **opération** est l'application d'un **opérateur** à des **opérandes**

Exemple

$9 + 12$

- 9 et 12 sont les opérandes
- $+$ est l'opérateur

Définition

Une **opération** est l'application d'un **opérateur** à des **opérandes**

Exemple

$9 + 12$

- 9 et 12 sont les opérandes
- $+$ est l'opérateur

Définition

Une **opération** est l'application d'un **opérateur** à des **opérandes**

Exemple

$9 + 12$

- 9 et 12 sont les opérandes
- $+$ est l'opérateur

Définition

L'**arité** d'un opérateur est le nombre d'opérandes auxquelles il s'applique.

Exemple

- Le + est d'arité 2, il est **binaire**
- − est d'arité 1 (**unaire**) ou 2

En java, les opérateurs sont unaires, binaires, et un seul est ternaire.

Définition

L'**arité** d'un opérateur est le nombre d'opérandes auxquelles il s'applique.

Exemple

- Le + est d'arité 2, il est **binaire**
- − est d'arité 1 (**unaire**) ou 2

En java, les opérateurs sont unaires, binaires, et un seul est ternaire.

Définition

L'**arité** d'un opérateur est le nombre d'opérandes auxquelles il s'applique.

Exemple

- Le + est d'arité 2, il est **binaire**
- − est d'arité 1 (**unaire**) ou 2

En java, les opérateurs sont unaires, binaires, et un seul est ternaire.

Définition

L'**arité** d'un opérateur est le nombre d'opérandes auxquelles il s'applique.

Exemple

- Le + est d'arité 2, il est **binaire**
- − est d'arité 1 (**unaire**) ou 2

En java, les opérateurs sont unaires, binaires, et un seul est ternaire.

Définition

Un opérateur est **associatif** si le résultat de l'opération ne dépend pas des positions des parenthèses.

Exemple

- Le $+$ est associatif, parce que $(a + b) + c = a + (b + c)$
- $-$ ne l'est pas : $(a - b) - c \neq a - (b - c)$

Définition

Un opérateur est **associatif** si le résultat de l'opération ne dépend pas des positions des parenthèses.

Exemple

- Le $+$ est associatif, parce que $(a + b) + c = a + (b + c)$
- $-$ ne l'est pas : $(a - b) - c \neq a - (b - c)$

Définition

Un opérateur est **associatif** si le résultat de l'opération ne dépend pas des positions des parenthèses.

Exemple

- Le $+$ est associatif, parce que $(a + b) + c = a + (b + c)$
- $-$ ne l'est pas : $(a - b) - c \neq a - (b - c)$

Définition

Une opérateur est **préfixe**, **infixe** ou **postfixe** selon qu'il se place avant, entre, ou après ses opérandes.

- Le $+$ est infixe
- Le $-$ unaire est préfixe

Définition

Une opérateur est **préfixe**, **infixe** ou **postfixe** selon qu'il se place avant, entre, ou après ses opérandes.

- Le $+$ est infixe
- Le $-$ unaire est préfixe

Définition

Une opérateur est **préfixe**, **infixe** ou **postfixe** selon qu'il se place avant, entre, ou après ses opérandes.

- Le $+$ est infixe
- Le $-$ unaire est préfixe

Définition

Un opérateur unaire est toujours prioritaire sur un opérateur binaire ou ternaire.

Définition

La **négation arithmétique** d'un nombre est ce qu'il faut lui additionner pour obtenir 0.

Définition

La **négation binaire** d'un nombre x, notée `~x`, s'obtient en prenant la négation de chacun de ses bits.

Exemple

$\sim 186 = 69$

Définition

La **négation binaire** d'un nombre x, notée ~x, s'obtient en prenant la négation de chacun de ses bits.

Exemple

$\sim 186 = 69$

Définition

Les opérateurs unaires sont **associatifs à droite**, ce qui signifie qu'ils sont exécutés de la droite vers la gauche.

Exemple

```
~-~i = ~(-(~i))
```

Définition

Les opérateurs unaires sont **associatifs à droite**, ce qui signifie qu'ils sont exécutés de la droite vers la gauche.

Exemple

```
~-~i = ~(-(~i))
```

- Les opérateurs binaires sont généralement associatifs à gauche.
- Les opérateurs unaires sont prioritaires sur les opérateurs binaires.
- Ils ne sont pas tous de même priorité

- Les opérateurs binaires sont généralement associatifs à gauche.
- Les opérateurs unaires sont prioritaires sur les opérateurs binaires.
- Ils ne sont pas tous de même priorité

- Les opérateurs binaires sont généralement associatifs à gauche.
- Les opérateurs unaires sont prioritaires sur les opérateurs binaires.
- Ils ne sont pas tous de même priorité

Définition

- L'opération `x << y` décale les bits de x de y positions vers la gauche
- `x >> y` décale les bits de x de y positions vers la droite

Exemple

- $6 << 2 = 24$
- $100 >> 3 = 12$

Définition

- L'opération `x << y` décale les bits de x de y positions vers la gauche
- `x >> y` décale les bits de x de y positions vers la droite

Exemple

- $6 << 2 = 24$
- $100 >> 3 = 12$

Définition

- L'opération `x << y` décale les bits de x de y positions vers la gauche
- `x >> y` décale les bits de x de y positions vers la droite

Exemple

- $6 << 2 = 24$
- $100 >> 3 = 12$

Définition

- L'opération `x << y` décale les bits de x de y positions vers la gauche
- `x >> y` décale les bits de x de y positions vers la droite

Exemple

- $6 << 2 = 24$
- $100 >> 3 = 12$

Définition

Les symboles suivants opèrent sur la représentation binaire des nombres :

- & : *ET* logique.
- | : *OU* logique.
- ^ : *OU* exclusif logique.

Exemple

- 0011 1100 ET 0000 1111 = 0000 1100
- 0011 1100 OU 0000 1111 = 0011 1111
- 0011 1100 OU EXCLUSIF 0000 1111 = 0011 0011

Définition

Les symboles suivants opèrent sur la représentation binaire des nombres :

- & : *ET* logique.
- | : *OU* logique.
- ^ : *OU* exclusif logique.

Exemple

- 0011 1100 ET 0000 1111 = 0000 1100
- 0011 1100 OU 0000 1111 = 0011 1111
- 0011 1100 OU EXCLUSIF 0000 1111 = 0011 0011

Définition

Les symboles suivants opèrent sur la représentation binaire des nombres :

- & : *ET* logique.
- | : *OU* logique.
- ^ : *OU* exclusif logique.

Exemple

- 0011 1100 ET 0000 1111 = 0000 1100
- 0011 1100 OU 0000 1111 = 0011 1111
- 0011 1100 OU EXCLUSIF 0000 1111 = 0011 0011

Définition

Les symboles suivants opèrent sur la représentation binaire des nombres :

- & : *ET* logique.
- | : *OU* logique.
- ^ : *OU* exclusif logique.

Exemple

- 0011 1100 ET 0000 1111 = 0000 1100
- 0011 1100 OU 0000 1111 = 0011 1111
- 0011 1100 OU EXCLUSIF 0000 1111 = 0011 0011

Définition

Les symboles suivants opèrent sur la représentation binaire des nombres :

- & : *ET* logique.
- | : *OU* logique.
- ^ : *OU* exclusif logique.

Exemple

- 0011 1100 ET 0000 1111 = 0000 1100
- 0011 1100 OU 0000 1111 = 0011 1111
- 0011 1100 OU EXCLUSIF 0000 1111 = 0011 0011

Définition

Les symboles suivants opèrent sur la représentation binaire des nombres :

- & : *ET* logique.
- | : *OU* logique.
- ^ : *OU* exclusif logique.

Exemple

- 0011 1100 ET 0000 1111 = 0000 1100
- 0011 1100 OU 0000 1111 = 0011 1111
- 0011 1100 OU EXCLUSIF 0000 1111 = 0011 0011

Définition

Le = est un opérateur binaire.

Exemple

```
a = b + (c = 3);
```

Définition

Le = est un opérateur binaire.

Exemple

```
a = b + (c = 3);
```

noms	opérateurs
produit	*, /, %
somme	+, −
décalage binaire	>>, <<
ET binaire	&
OU Exlusif binaire	^
OU binaire	|
affectation	=

Il est possible de contracter les instructions unaires :

Exemple

- `i = i + 1` $\Longrightarrow$ `i++` : post-incrémentation
- `i = i + 1` $\Longrightarrow$ `++i` : pré-incrémentation
- `i = i - 1` $\Longrightarrow$ `i--` : post-décrémentation
- `i = i - 1` $\Longrightarrow$ `--i` : pré-décrémentation

Il est possible de contracter les instructions unaires :

Exemple

- `i = i + 1` $\Longrightarrow$ `i++` : post-incrémentation
- `i = i + 1` $\Longrightarrow$ `++i` : pré-incrémentation
- `i = i - 1` $\Longrightarrow$ `i--` : post-décrémentation
- `i = i - 1` $\Longrightarrow$ `--i` : pré-décrémentation

Il est possible de contracter les instructions unaires :

Exemple

- $\texttt{i} = \texttt{i} + 1 \Longrightarrow \texttt{i++}$: post-incrémentation
- $\texttt{i} = \texttt{i} + 1 \Longrightarrow \texttt{++i}$: pré-incrémentation
- $\texttt{i} = \texttt{i} - 1 \Longrightarrow \texttt{i--}$: post-décrémentation
- $\texttt{i} = \texttt{i} - 1 \Longrightarrow \texttt{--i}$: pré-décrémentation

Il est possible de contracter les instructions unaires :

Exemple

- $\texttt{i} = \texttt{i} + 1 \Longrightarrow \texttt{i++}$: post-incrémentation
- $\texttt{i} = \texttt{i} + 1 \Longrightarrow \texttt{++i}$: pré-incrémentation
- $\texttt{i} = \texttt{i} - 1 \Longrightarrow \texttt{i--}$: post-décrémentation
- $\texttt{i} = \texttt{i} - 1 \Longrightarrow \texttt{--i}$: pré-décrémentation

Définition

`variable = variable operateurBinaire expression` $\Longrightarrow$
`variable operateurBinaire= expression.`

avant	après
a = a + b	a += b
a = a − b	a −= b
a = a * b	a *= b
a = a / b	a /= b
a = a % b	a %= b
a = a >> i	a >>= i
a = a << i	a <<= i
a = a & b	a &= b
a = a ^ b	a ^= b
a = a | b	a |= b

Vous vous douterez que l'égalité ne peut pas être contractée...

Nous ordonnons de façon grossière les types de la façon suivante :

Définition

double > float > long > int > short > byte

Est-il intelligent d'écrire ceci ?

Exemple

```
int i = 4;
System.out.println("L'inverse de " + i + " est " + 1/i);
```

Est-il mieux de procéder ainsi ?

Exemple

```
int i = 4;
System.out.println("L'inverse de " + i + " est " + 1./i);
```

Maintenant essayons d'arranger ça :

Exemple

```
int i = 4, j= 5;
System.out.println("Le quotient de " + i + " et " + j + " est
    " + i/j + ".");
```

Définition

Le **cast** est une conversion de type explicite dans une opération.

Il se note en plaçant entre parenthèse le type vers lequel on veut convertir avant l'expression à convertir.

Définition

Le **cast** est une conversion de type explicite dans une opération.

Il se note en plaçant entre parenthèse le type vers lequel on veut convertir avant l'expression à convertir.

Par exemple,

Exemple

```
int i = 4, j= 5;
System.out.println("Le quotient de " + i + " et " + j +
        " est " + (float)i/j + ".");
```

Est-il intelligent d'écrire ceci ?

Exemple

```
int i = 4, j= 5;
System.out.println("Le quotient de " + i + " et " + j +
        " est " + (float)(i/j) + ".");
```

noms	opérateurs
opérateurs unaires	cast, −, ~, ++, −−
produit	*, /, %
somme	+, −
décalage binaire	>>, <<
ET binaire	&
OU Exlusif binaire	^
OU binaire	|
affectation	=