Chapitre 6

Les opérateurs

Les opérateurs sont des symboles du langage qui permettent de manipuler des variables en effectuant des opérations de calcul, la récupération de résultat, l'évaluation d'expressions logiques, la comparaison, l'incrémentation/décrémentation, etc

1. Les opérateurs arithmétiques
2. Les opérateurs d’assignation
3. Les opérateurs d’incrémentation
4. Les opérateurs de comparaison
5. Les opérateurs logiques

1. Les opérateurs arithmétiques
2. Les opérateurs d’assignation
3. Les opérateurs d’incrémentation
4. Les opérateurs de comparaison
5. Les opérateurs logiques

Les opérateurs arithmétiques (1/3)

- Permettent de réaliser des opérations de calcul sur les variables dont le type de données est d'ordre mathématique :
  - types primitifs : byte, short, char, int, long, float, double
  - Wrappers : Byte, Short, Integer, Long, Float, Double

- Les opérateurs en question :
  - + : permet de faire l'addition sur deux ou plusieurs variables (appelées **opérandes**)
  - - : permet de faire la soustraction sur deux ou plusieurs variables
  - * : permet de faire la multiplication sur deux ou plusieurs variables
  - / : permet de faire la division sur deux ou plusieurs variables
    - Lorsque les deux opérandes de l'opérateur / sont des types *byte, short, char, int ou long*, alors il agit en division entière et donc renvoi un quotient entier
    - Si une des opérandes est de type *float* ou *double*, alors l'opérateur / agit en division normal et renvoi un nombre décimal
  - % : permet d'obtenir le *reste* issu d'une *division entière*

- Le cas du type String :
  - L'opérateur + peut s'appliquer sur les variables de types **String**
  - Lorsque les variables sont de types **String**, alors + opère comme un symbole de concaténation des mots

- Quelques règles :

A. Le résultat d'une opération arithmétique entre deux opérandes de types différents aura comme type résultat celui de l'opérande qui a la taille la plus grande (*cf. chapitre 5, slide 8*). Exemple : une addition entre un *int* et un *long* aura comme type résultat : *long*

B. Le résultat d'une opération arithmétique entre un nombre entier (byte, short, int, long) et un nombre réel (float, double) aura comme résultat un nombre réel et suivant l'énoncé de la règle A pour le type retour

C. Il est possible de réaliser une opération arithmétique entre un type primitif et un Wrapper. Dans ce cas Java va « unboxer » le type Wrapper afin d'effectuer le calcul, puis appliquer la règle A pour évaluer le type retour

Les opérateurs arithmétiques (2/3)

- Exemples d'opérations entre opérandes de mêmes types primitifs :

Opérateurs	Exemple	Résultat	Type résultat
+	int a = 5; int b = 7; a + b	12	int
-	float a = 5.6f; float b = 7.1f; b - a	1.5	float
*	long a = 8; long b = 120; a*b	960	long
/	long a = 123; long b = 11; a/b	12	long

- Le cas du type String :

Opérateurs	Exemple	Résultat	Type résultat
+	String s1 = "Hello"; String s2 = "World"; s1 + s2	HelloWorld	String
+	String s1 = "Hello"; int a = 123; s1 + a	Hello123	String

- Exemples d'opérations entre opérandes de types différents :

Opérateurs	Exemple	Résultat	Type résultat
+	int a = 5; long b = 7; a + b	12	long
-	float a = 5.6f; short b = 7; a - b	-1.4	float
*	float a = 8.1; double b = 12.001d; a*b	97.2081	double
/	long a = 123; float b = 11.0; a/b	11.1818	float

Les opérateurs arithmétiques (3/3)

- Les opérations de calcul complexes :
  - On peut avoir une opération de calcul qui combine plusieurs opérateurs arithmétiques à la fois : +, -, *, /
  - Exemple : comment évaluerez-vous l'opération suivante si vous étiez à la place de l'interpréteur Java pour trouver le bon résultat ?
    - ➔ *10-504*21+741-25*12+989/31*
    - ➔ *Réponse :*

      La réponse se trouve dans les règles de priorisation des opérateurs mathématiques

- **Notion de priorisation :**
  - L'application des opérateurs dans une opération doit respecter les règles de priorité résumées dans le tableau suivants :

Opérateurs	Niveau de priorité
*	2
/	2
+	1
-	1

Plus l'indice de priorité est élevé, plus l'opérateur est prioritaire

Si deux opérateurs ont le même indice, alors ils sont égaux en priorité, et donc l'application de l'un ou l'autre n'a pas d'impact

Ainsi, en tenant compte de ces règles de priorité, l'opération *10-504*21+741-25*12+989/31*, peut être traduite avec des parenthèses ainsi qu'il suit :

*10-(504*21)+741-(25*12)+(989/31)*

⇔ *10 - 10584 + 741 – 300 + 31.903225806451*

⇔ *-10574 + 741 -300 + 31.903225806451*

⇔ *-9833 – 300 + 31.903225806451*

⇔ *-10133 + 31.903225806451*

⇔ *-10101.09677419*

1 Les opérateurs arithmétiques

2 Les opérateurs d’assignation

3 Les opérateurs d’incrémentation

4 Les opérateurs de comparaison

5 Les opérateurs logiques

Les opérateurs d'assignation

- Permettent de réaliser l'action d'affectation d'une opération de calcul dans une variable résultat

- Les opérateurs en question :
  - = : permet d'affecter (assigner) le résultat d'un calcul (situé à droite du symbole) à une variable (située à gauche du symbole)
  - += : additionne une valeur à une variable *donnée* et y affecte le résultat dans cette même variable
  - -= : soustrait une valeur à une variable *donnée et y affecte le résultat dans cette même variable*
  - *= : multiplie une valeur à une variable *donnée et y affecte le résultat dans cette même variable*
  - /= : divise une valeur à une variable *donnée et y affecte le résultat dans cette même variable*

- Exemples :

Opérateurs	Exemple	Equivalence	Résultat
=	int a = 5; int b = 7; r = a + b ;	N/A	r = 12
+=	int a = 5; int b = 7 a += b ;	a += b ⇔ a = a + b	a = 12
-=	float a = 5.6f; float b = 7.1f; b -= a	a -= b ⇔ a = a - b	a = -1.5
*=	long a = 8; long b = 120; a *= b	a *= b ⇔ a = a*b	a = 960
/=	long a = 123; long b = 11; a /= b	a /= b ⇔ a = a/b	a = 12

1 Les opérateurs arithmétiques

2 Les opérateurs d’assignation

3 Les opérateurs d’incrémentation

4 Les opérateurs de comparaison

5 Les opérateurs logiques

Les opérateurs d'incrémentation

- Permettent d'incrémenter (d'une unité) ou de décrémenter (d'une unité) une variable entière de type : byte, short, int, long
- Les opérateurs en question :
  - ++ : incrémente de 1 la valeur d'une variable entière
    - Opérateur de postincrémentation : Exemple : i++ ⇔ $i = i + 1$
    - Opérateur de préincrémentation : Exemple : ++i ⇔ $i = i + 1$
  - -- : décrémente de 1 la valeur d'une variable entière
    - Opérateur de postdécrémentation : Exemple : i-- ⇔ $i = i - 1$
    - Opérateur de prédécrémentation : Exemple : --i ⇔ $i = i - 1$
- Exemples :

Opérateurs	Exemple		Résultat
++	int i = 5;	i++;	i = 6
--	int i = 5;	i-- ;	i = 4

- Priorisation :
  - L'opérateur de pré ou de post incrémentation/décrémentation est plus prioritaire que tout opérateur arithmétique
  - L'opérateur de préincrémentation (respectivement prédécrémentation) est plus prioritaire que celui de postincrémentation (respectivement postdécrémentation)
  - La différence entre la préincrémentation et la postincrémentation sur une variable i, est qu'avec la préincrémementation la variable est **d'abord incrémentée avant d'être utilisée**, alors qu'avec la postincrémentation, la variable est **d'abord utilisée et incrémentée après** l'opération (même principe pour la pré/post décrémentation)

Exemple : si tab est un tableau, soit la variable i = 5, alors :

- tab[++i] = tab[5+1] = tab[6]. La valeur de i est d'abord évaluée avant de renvoyer la case recherchée du tableau
- tab[i++] = tab[5]. La valeur initiale de i est d'abord utilisée pour renvoyer la case recherchée du tableau, puis sa valeur n'est incrémentée qu'à la fin

1 Les opérateurs arithmétiques

2 Les opérateurs d’assignation

3 Les opérateurs d’incrémentation

4 Les opérateurs de comparaison

5 Les opérateurs logiques

Les opérateurs de comparaison (1/2)

- Permettent de comparer deux variables entre eux
  - Porte uniquement sur les types primitifs : byte, short, char, int, long, float, double
  - Et les Wrappers de types primitifs
  - **L'unique type de retour de la comparaison est le type boolean (c'est-à-dire, true ou false)**
  - La comparaison peut s'effectuer entre variables de types différents

- Les opérateurs en question :
  - == : permet de comparer si deux variables sont **égales** en valeurs. Ne pas confondre avec le symbole = d'affectation/assignation
  - > : permet de comparer si la variable de gauche est **strictement supérieure** en valeur à la variable de droite
  - >= : permet de comparer si la variable de gauche est **supérieur ou égale** en valeur à la variable de droite
  - < : permet de comparer si la variable de gauche est **strictement inférieure** en valeur à la variable de droite
  - <= : permet de comparer si la variable de gauche est **inférieure ou égale** en valeur à la variable de droite
  - != : permet de comparer si deux variables sont **différents** en valeurs

- Attention !!!
  - Si l'on utilise ces opérateurs pour comparer deux objets, c'est-à-dire, les variables de *type de référence*, alors ce sont les **références** (les adresses mémoires) de ces objets qui seront comparées et pas leurs valeurs intrinsèques. C'est un piège à éviter !

  - Nous verrons dans le chapitre 12 comment on compare les objets (instance de classe) ou les chaînes de caractères (String) entre eux

Les opérateurs de comparaison (2/2)

- Exemples :

Opérateurs	Exemple		Résultat
==	double a = 10.3d; float b = 27.8f;	a == b	Retourne **false**, car a et b n'ont pas les mêmes valeurs
>	long a = 67; short b = 45;	a > b	Retourne **true**, car la valeur de a est strictement supérieure à celle de b
>=	Integer a = 101; Long b = 101;	a >= b	Retourne **true**, car les wrappers a et b sont égaux en valeurs, et par conséquent on peut dire que a est supérieur ou égal à b
<	int a = 79; int b = 79;	a > b	Retourne **false**, car la valeur de a n'est pas strictement inférieur à b
<=	Float a = 1234.42f; long b = 789;	a <= b	Retourne **false**, car la valeur de a n'est pas inférieure ou égale à celle de b
!=	int a = 507; int b = 699;	a != b	Retourne **true**, car les valeurs de a et de b sont différentes

- Attention !!! Cas d'utilisation des opérateurs == et != sur les objets :

Personne p1 = new Personne();
Personne p2 = new Personne()

Faire : **p1 == p2** ou **p1 != p2**, ne compare pas si les objets p1 et p2 sont équivalents ou différents en contenance. Mais compare plutôt si les deux objets pointent à la même adresse mémoire (c-à-d, s'ils ont la même référence) [cf. chapitre 4, slides 10 et 11). Ainsi, **p1 == p2 => false**, et **p1 != p2 => true**

Par contre :
Si on a par exemple : Personne p1 = new Personne();
Personne p2 = p1;
Alors p1 et p2 pointent sur la même adresse mémoire. Ainsi, **p1 == p2 => true** et **p1 != p2 => false**

1 Les opérateurs arithmétiques

2 Les opérateurs d’assignation

3 Les opérateurs d’incrémentation

4 Les opérateurs de comparaison

5 Les opérateurs logiques

Les opérateurs logiques (1/3)

- Permettent de vérifier que plusieurs conditions sont vraies ou fausses

- Les opérateurs en question :
  - || : représente un OU logique appliqué dans une **expression conditionnelle**
    - Exemple d'expression : (condition1) || (condition2)
    - Retournera true, si au moins une de ces conditions est vraie
    - Retournera false, si aucune des deux conditions n'est vraie (i.e, si toutes les deux sont fausses)
  - && : représente un ET logique appliqué dans une **expression conditionnelle**
    - Exemple d'expression : (condition1) && (condition2)
    - Dès que la condition1 est évaluée comme fausse, l'interpréteur Java s'arrête et retourne false
    - Retournera true uniquement si les deux conditions sont évaluées comme étant chacune vraie
  - ! : représente la Négation appliquée sur une **expression conditionnelle**
    - Exemple d'expression : !condition
    - Retourne true si *condition* est évaluée comme étant fausse
    - Retourne false si *condition* est évaluée comme étant vraie

- Une condition c'est quoi ?
  - C'est un test mettant en exergue des opérateurs de comparaison entre des données
  - Exemple : **(10 > 15)** est une condition. Dans le cas d'espèce, cette condition est fausse, car 10 n'est pas supérieur à 15

Les opérateurs logiques (2/3)

❑ Exemples :

Opérateurs	Exemple		Résultat	Commentaire
||	int a = 10; int b = 15; int c = 5; int d = 5;	(a > b) || (c == d)	true	La condition (a > b) vaut *false*, mais la condition (c == d) vaut *true*. Ainsi, on a : *false* || *true* ==> *true*
&&	int a = 10; int b = 15; int c = 5; int d = 5;	(a > b) && (c == d)	false	La condition (a > b) vaut *false*, mais la condition (c == d) vaut *true*. Ainsi, on a : *false* && *true* ==> *false*
!	int a = 10; int b = 15;	!(a > b)	true	La condition (a > b) vaut *false*, mais en appliquant la négation de false, on obtient *true*

❑ Algèbre de Bool

➢ Pour maîtriser l'évaluation d'une expression conditionnelle qui utilise les opérateurs ||, && et !, il faut mémoriser les tables de Bool suivantes :

x	*y*	*x* || *y*
true	false	true
false	true	true
true	true	true
false	false	false

x	*y*	*x* && *y*
true	false	false
false	true	false
true	true	true
false	false	false

x	!*x*
true	false
false	true

Les opérateurs logiques (3/3)

- Les expressions conditionnelles complexes :
  - On peut avoir une expression qui combine plusieurs opérateurs logiques à la fois : ||, &&, !
  - Exemple, comment traduirez-vous l'expression conditionnelle qui se découle de la phrase suivante ?
    - ➔ *x est supérieur à 10* ***et*** *x est inférieur ou égal à 50* ***ou*** *que y est inférieur à 0* ***mais*** *est* ***différent*** *de -102*
    - ➔ *Réponse :*

      En général, pour traduire une phrase en expression conditionnelle, il faut identifier les mots comme ***ou***, ***et***, ***différent***, ***n'est pas***, ***est égal***, ***mais***, ***inférieur***, ***supérieur***, *etc, qui permettent d'identifier les opérateurs à utiliser dans l'expression :*

      (x > 10) && (x <= 50) || (y < 0) && (y != -102)

- Comment évaluer les expressions conditionnelles complexes ? **Notion de priorisation :**
  - Si nous prenons l'expression ci-dessus, il va s'en dire qu'il s'agit d'une expression complexe difficile à évaluer le résultat
  - Par où commencera-t-on à évaluer cette expression si l'on pose x = 70 et y = -99 ?
  - Réponse : l'évaluation d'une expression va de la gauche vers la droite avec application de la priorité suivante entre opérateurs

Opérateurs	Niveau de priorité
!	3
&&	2
||	1

Plus l'indice de priorité est élevé, plus l'opérateur est prioritaire

Ainsi dans l'expression *(x > 10) && (x <= 50) || (y < 0) && (y != -102)*, vu que && est plus prioritaire que ||, on peut ajouter des parenthéses pour rendre la lecture plus facile :
(*(x > 10) && (x <= 50)* ***)*** *||* ***(*** *(y < 0) && (y != -102)* ***)***

➔ *Pour x = 70,* ***(*** *(x > 10) && (x <= 50)* ***)*** *=> (true && false) =>* ***false ;*** *pour y = -99,* ***(*** *(y < 0) && (y != -102)* ***)*** *=> (true && true) =>* ***true ;*** d'où finalement on a : *false || true => true*

➔ ***Conclusion :*** *pour x = 70 et y = -99, (x > 10) && (x <= 50) || (y < 0) && (y != -102) => true*


SUCCESS