ОПЕРАТОРЫ JAVA. ПРИОРИТЕТ. ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ ОПЕРАТОРОВ

Операторы в языке программирования «Java» — это специальные символы, которые сообщают транслятору о том, что вы хотите выполнить операцию с некоторыми операндами.

Операндом называют аргумент операции, данное, которое обрабатывается командой; грамматическая конструкция, обозначающая выражение, задающее значение аргумента операции; иногда операндом называют место, позицию в тексте, где должен стоять аргумент операции. Например, в выражение *a+b* операндами являются *a* и *b,* а в выражении *sin x* операндом является *x*.

Некоторые операторы требуют *одного операнда*, их называют **унарными**. Одни операторы ставятся *перед операндами* и называются **префиксными**, другие — *после*, их называют **постфиксными операторами.** Большинство же операторов ставят *между двумя операндами*, такие операторы называются **инфиксными бинарными операторами.** Существует **тернарный оператор**, *работающий с тремя операндами.*

Встроенные операторы в «Java» можно разбить на 4 класса:

I. Арифметические;

II. Битовые;

III. Операторы сравнения;

IV. Логические.

Арифметические операторы

Оператор	Результат
+	Сложение
-	Вычитание (также унарный минус)
*	Умножение
/	Деление
%	Остаток от деления (деление по модулю)
+=	Сложение (с присваиванием)
-=	Вычитание (с присваиванием)
*=	Умножение (с присваиванием)
/=	Деление (с присваиванием)
%=	Остаток от деления (с присваиванием)
++	Инкремент
--	Декремент

Примечание. Арифметические операторы используются для вычислений так же как в алгебре. Допустимые операнды должны иметь числовые типы. Например, использовать эти операторы для работы с логическими типами нельзя, а для работы с типом char можно, поскольку в «Java» тип char — это подмножество типа int.

Битовые операторы

Оператор	Результат
|	*ИЛИ*
|=	*ИЛИ* (с присваиванием)

&	*И*
&=	*И* (с присваиванием)
^	Исключающее *ИЛИ*
^=	Исключающее *ИЛИ* (с присваиванием)
~	Унарное отрицание
>>	Сдвиг вправо
>>=	Сдвиг вправо (с присваиванием)
>>>	Сдвиг вправо с появлением нулей
>>>=	Сдвиг вправо с появлением нулей и присваиванием
<<	Сдвиг влево
<<=	Сдвиг влево (с присваиванием)

Примечание. Битовые операторы используются для целых числовых типов данных — long, int, short, char и byte. Операторы битовой арифметики работают с каждым битом как с самостоятельной величиной.

Операторы отношения

Оператор	Результат
<	Меньше
>	Больше
<=	Меньше либо равно
>=	Больше либо равно
==	Равно
!=	Не равно

Примечание. Операторы отношения используются для сравнения символов, целых и вещественных чисел, логических значений, а также для сравнения ссылок при работе с объектами.

Логические операторы

Оператор	Результат
||	*ИЛИ*
&&	*И*
!	Унарное отрицание

К логическим операторам относится также оператор определения принадлежности типу instanceof (используется для определения того, является ли текущий объект экземпляром указанного класса), оператор **[ ]** и тернарный оператор **?:** (if-then-else).

Общая форма оператора if-then-else такова:

выражение1? выражение2: выражение3

Примечание. Логические операции выполняются только над значениями типа boolean (true или false).

Приоритет операторов

Высший			
()	[]		
++	--	~	!
*	/	%	
+	-		
>>	>>>	<<	

>	<	>=	<=
==	!=		
&			
^			
|			
&&			
||			
?:			
Присваивания: +=, -=, *=, /=, %=, |=, &=, ^=, >>=, >>>= , <<=			
Низший			

ПРИМЕРЫ использования операторов

Пример №1. Арифметические операторы.

Программный код	Результат выполнения программы

`public static void main(String[] args) {`
`  int a = 1 + 1;`
`  int  b = a * 3;`
`  int c = b / 2;`
`  int d = b - a;`
`  int e = -d;`
`  System.out.println("a =  " + a);`
`  System.out.println("b =  " + b);`
`  System.out.println("c =  " + c);`
`  System.out.println("d =  " + d);`
`  System.out.println("e =  " + e);`
`}`
	`a =  2`
	`b =  6`
	`c =  3`
	`d =  4`
	`e = -4`

Пример №2. Арифметические операторы с присваиванием.

Программный код	Результат выполнения программы

`public static void main(String[] args) {`
`  int a = 1;`
`  int b = 2;`
`  int c = 3;`
`  int d = 4;`
`  int e = 5;`
`  a += 5;`
`  b *= 4;`
`  c += a * b;`
`  c %= 6;`
`  d -= 10;`
`  e /= 5;`
`  System.out.println("a = " + a);`
`  System.out.println("b = " + b);`
`  System.out.println("c = " + c);`
`  System.out.println("d = " + d);`
`  System.out.println("e = " + e);`
`}`
	`a = 6`
	`b = 8`
	`c = 3`
	`d = -6`
	`e = 1`

Пример №3. Оператор Остаток от деления.

Программный код	Результат выполнения программы

`public static void main(String[] args) {`
`  int x = 52;`
`  double y = 52.3;`
`  System.out.println("x = " + x % 10);`
`  System.out.println("y = " + y % 10);`
`}`
	`x = 2`
	`y = 2.299999999999997`

Пример №4. Операторы Инкремент и Декремент.

Программный код	Результат выполнения программы

`public static void main(String[] args) {`
`  int a = 1;`
`  int b = 2;`
`  int c = ++b;`
`  int d = a++;`
`  c++;`
`  System.out.println("c = " + c);`
`  System.out.println("d = " + d);`
`}`
	`c = 4`
	`d = 1`

Пример №5. Битовые операторы.

Программный код	Результат выполнения программы

`public static void main(String[] args) {`
`  String binary[] = { "OOOO", "0001", "0010",`
`  "0011", "0100", "0101","0110", "0111", "1000",`
`  "1001", "1010"};`
`  int a = 3;  //  0+2+1  или двоичное 0011`
`  int b = 6;  //  4+2+0  или двоичное 0110`
`  int c = a | b;`
`  int d = a & b;`
`  int e = a ^ b;`
`  int f = (~a & b) | (a & ~b);`
`  System.out.println(" a = " + binary[a]);`
`  System.out.println(" b = " + binary[b]);`
`  System.out.println(" c = " + binary[c]);`
`  System.out.println(" d = " + binary[d]);`
`  System.out.println(" e = " + binary[e]);`
`  System.out.println(" f = " + binary[f]);`
`}`
	`a = 0011`
	`b = 0110`
	`c = 0111`
	`d = 0010`
	`e = 0101`
	`f = 0101`

Пример №6. Битовые операторы с присваиванием.

Программный код	Результат выполнения программы

`public static void main(String[] args) {`
`  int a = 1;`
`  int b = 2;`
`  a |= 4;`
`  b >>= 1;`
`  System.out.println("a = " + a);`
`  System.out.println("b = " + b);`
`}`
	`a = 5`
	`b = 1`

Пример №7. Оператор Сдвиг вправо.

Программный код	Результат выполнения программы

`public static void main(String[] args) {`
`  char hex[] = { '0', '1', '2', '3', '4',`
`'5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'};`
`  byte b = (byte) 0xf1;`
`  System.out.println("b = " + hex[(b >> 4)& 0x0f]);`
`}`
	`b = f`

Примечание. Байтовое значение преобразуется в строку, содержащую его шестнадцатеричное представление. Обратите внимание, что сдвинутое значение приходится маскировать, т. е. логически умножать на значение 0x0f для того, чтобы очистить заполняемые в результате расширения знака биты и понизить значение до пределов, допустимых при индексировании массива шестнадцатеричных цифр.

Пример №8. Операторы Отношения.

Программный код	Результат выполнения программы

`public static void main(String[] args) {`
`  int a = 5;`
`  int b = 10;`
`  boolean c = a < b;`
`  boolean d = a > b;`
`  boolean e = (a == b);`
`  boolean f = (a != b);`
`  System.out.println(" c = " + c);`
`  System.out.println(" d = " + d);`
`  System.out.println(" e = " + e);`
`  System.out.println(" f = " + f);`
`}`
	`c = true`
	`d = false`
	`e = false`
	`f = true`

Пример №9. Оператор Унарное отрицание.

Программный код	Результат выполнения программы

`public static void main(String[] args) {`
`  boolean a = true;`
`  boolean b = !a;`
`  System.out.println(" b = " + b);`
`}`
	`b = false`

Пример №10. Тернарный оператор if-then-else (выражение1? выражение2: выражение3).

Программный код	Результат выполнения программы

`public static void main(String[] args) {`
`  int a = 42;`
`  int b = 2;`
`  int c = 99;`
`  int d = 0;`
`  int e = (b == 0) ? 0 : (a / b);`
`  int f = (d == 0) ? 0 : (c / d);`
`  System.out.println("e = " + e);`
`  System.out.println("f = " + f);`
`}`
	`e = 21`
	`f = 0`

Примечание. В качестве первого операнда может быть использовано любое выражение, результатом которого является значение типа boolean. Если результат равен ***true,*** то выполняется оператор, заданный ***вторым операндом.*** Если же первый операнд равен ***false,*** то выполняется ***третий операнд***. Второй и третий операнды должны возвращать значения одного типа.

В приведенной программе этот оператор используется для проверки делителя перед выполнением операции деления. В случае нулевого делителя возвращается значение 0.