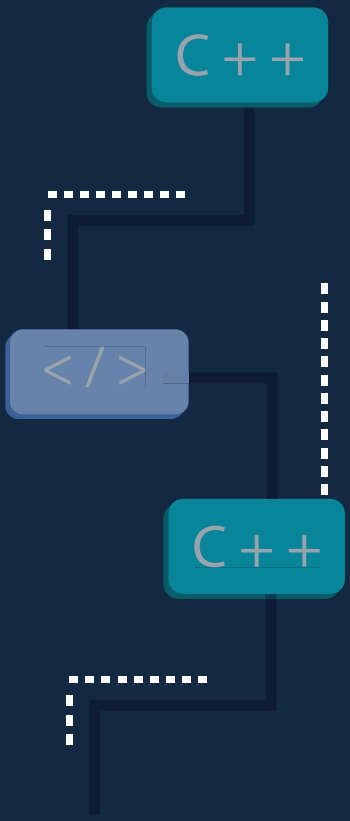

top
КОМПЬЮТЕРНАЯ АКАДЕМИЯ

Язык программирования JAVA

Урок №3

Операторы

Содержание

1. Операции

Операции в языке Java – это специальные символы, которые показывают какое действие необходимо выполнить для заданных операндов.

Операнд – может быть литералом, переменной или выражением, над которыми производится операция.

Операции могут применяться к одному (унарная операция), двум (бинарная операция) или трем (тернарная операция) операндам.

Операции могут быть объединены в выражения.

В отличии от C++ в Java отсутствует перегрузка операций, так как создатели языка решили, что перегрузка операций существенно запутывает код и усложняет его понимание.

Есть несколько операций, перегруженных по умолчанию:

- + – используется для сложения чисел и конкатенации строк;
- & – используется для побитовой операции с числами и для логического И;
- | – используется для побитовой операции с числами и для логического ИЛИ;
- ^ – используется для побитовой операции с числами и для логического исключающего ИЛИ;
- == – сравнивает любые типы;
- != – сравнивает любые типы.

Арифметические операции

Обозначение	Описание	Пример

+	Унарный плюс, не меняет значение операнда.	`int i = -1;`
		`i = +i;`
		`System.out.println(i);`

	Бинарный плюс, суммирует операнды.	`int i = -1;`
		`i = i + 3;`
		`System.out.println(i);`

-	Унарный минус – меняет знак операнда на противоположный.	`int i = 1;`
		`i = -i;`
		`System.out.println(i);`

	Бинарный минус – вычитает правый операнд из левого.	`int i = 3;`
		`i = i - 2;`
		`System.out.println(i);`

*	Умножение операндов	`int i = 2;`
		`i = i * 2;`
		`System.out.println(i);`

/	Деление левого операнда на правый.	`int i = 4;`
		`i = i / 2;`
		`System.out.println(i);`

%	Вычисление остатка от деления левого операнда на правый.	`int i = 3;`
		`i = i % 2;`
		`System.out.println(i);`

++	Бинарный оператор увеличения значения переменной на 1. Инкремент.	i++; заменяет выражение i = i + 1;
--	Бинарный оператор уменьшения значения переменной на 1. Декремент.	i--; заменяет выражение i = i - 1;

Побитовые операции

Побитовые операции производят вычисления использую в качестве операндов побитовое представление числа (в двоичной системе счисления).

ВАЖНО!!! *В качестве операндов в побитовых операциях могут быть использованы только целые примитивные типы или их классы-обертки.*

Операция	Описание	Пример	Результат
&	И (AND)	В каждом разряде результата получаем значащий бит, в случае если присутствует значащий бит в левом и правом операнде. `int i = 123 & 456;` Битовое представление 00000000000000000000000001111011 & 00000000000000000000000111001000 = 00000000000000000000000001001000	72

|	ИЛИ (OR)	В каждом разряде результата получаем значащий бит, в случае если присутствует значащий бит в левом или правом операнде.
		`final int one = 123;`
		`final int two = 456;`
		`int i = one | two;`
		Битовое представление
		00000000000000000000000001111011 |
		00000000000000000000000111001000 =
		00000000000000000000000111111011
			507

Операция	Описание	Пример	Результат
^	исключающее ИЛИ (XOR)	В каждом разряде результата получаем значащий бит, в случае если присутствует значащий бит в левом или правом операнде. Если бит есть в разряде обеих операндов, в результате получаем 0. `int i = 123 ^ 456;` Битовое представление 00000000000000000000000001111011 ^ 00000000000000000000000111001000 = 00000000000000000000000110110011	435
~	поразрядное отрицание (NOT)	Инвертирует все биты числа на противоположные (включая знаковый бит) `int i = ~123;` Битовое представление: ~ 00000000000000000000000001111011= 11111111111111111111111110000100	–124
<<	сдвиг влево	Смещает биты левого операнда влево на количество бит указанном в правом операнде, заполняя справа нулем.	
>>	сдвиг вправо	Смещает биты левого операнда вправо на количество бит указанном в правом операнде, заполняя разряды справа нулем.	
>>>	без знаковый сдвиг вправо	Смещает биты левого операнда вправо на количество бит указанном в правом операнде, заполняя слева нулем игнорируя битовый сдвиг.	

Integer.toBinaryString(value) – преобразует целое число value в строку в побитовом представление.

Пример метода, выводящего число в побитовом представлении с дополнением незначимыми нулями:

```
private static void printToBinary(int value)
{
 System.out.println(String.format("%32s", Integer.
   toBinaryString(value)).replace(' ', '0'));
}
```

Операции сравнения

Результат операций сравнения всегда имеет тип **boolean**.

Обозначение	Условие	Пример
>	операнд слева больше, чем операнд справа	int a = 7; boolean r = a > 5;
<	операнд слева меньше, чем операнд справа	int a = 7; boolean r = a < 5;
>=	операнд слева больше или равен операнду справа	int a = 7; boolean r = a >= 5;
<=	операнд слева меньше или равен операнду справа	int a = 7; boolean r = a <= 5;
==	операнд слева в точности равен операнду справа	int a = 7; boolean r = a == 5;
!=	операнд слева не равен операнду справа	int a = 7; boolean r = a != 5;

Логические операции

Логические операции применяются только к операндам типа **boolean**. Результат логических операций имеет тип **boolean**.

Обозначение операции	Описание
&	Логическое И
|	Логическое ИЛИ
^	Логическое исключающее ИЛИ
||	Оператор быстрой оценки ИЛИ*
&&	Оператор быстрой оценки И*
!	Отрицание

* *Операторы быстрой оценки не вычисляют результат выражения в правом операнде, если результат операции можно определить по значению первого операнда.*

Пример для И:

```
boolean t = 5 < 3 && 5 > 3
```

В данном примере в результате вычисления выражения левого операнда получим false, поэтому для вычисления результата операции нет необходимости вычислять правый операнд.

Пример для ИЛИ:

```
boolean t = (2 == 2) || 3 != 2
```

В данном примере в результате вычисления выражения левого операнда получим true, поэтому для вычисления результата операции ИЛИ нет необходимости вычислять правый операнд.

Таблица истинности логических операций

Операнд 1	Операнд 2	Операция			
		|	&	^	!Операнд1
false	false	false	false	false	true
true	false	true	false	true	false
false	true	true	false	true	true
true	true	true	true	false	false

Операция присваивания

Для сохранения в переменной значения литерала, переменной или значения выражения используется операция присваивания. Оператор присваивания имеет самый низкий приоритет, поэтому в начале выполнятся все операции в выражении и только потом вычисленное значение передастся в переменную.

Синтаксис:

```
переменная = выражение;
```

Пример:

```
int x = 3; // переменная будет инициализирована
   // значением 3
x = 6; // значение переменной будет перезаписано,
  // старое значение будет потеряно
```

Java поваляет присваивать значения одновременно нескольким переменным.

Пример:

```
int x;
int y;
int z;
```

```
x = y = z = 3;
System.out.println(x + y + z);
```

Результат: 9

Тернарный оператор

Тернарный оператор содержит три операнда. Результатом операции может быть второй или третий операнд. Первый операнд должен иметь тип boolean.

Синтаксис:

```
выражение1 ? выражение2 : выражение3;
```

Операция вернет значение «выражение2», если результат вычисления «выражения1» примет значение истина, иначе вернет значение «выражения3».

Пример:

```
int x = 1;
int y = 2;
int a = (x != y) ? x : y; // для лучшего чтения кода
       // рекомендуется брать
       // условие в скобки.
System.out.println(a);
```

Результат: 1

Приоритеты операций

Операции	Описание
() []	Круглые и квадратный скобки
++ -- + - ~ !	Декремент, инкремент, унарный плюс, унарный минус, поразрядное отрицание, логическое отрицание

Операции	Описание
* / %	Умножение, деление, остаток от деления
-	Сложение вычитание
>> >>> <<	Побитовые сдвиги: вправо, беззнаковый вправо, побитовый сдвиг влево
> >= < <=	Сравнение на: больше, больше или равно, меньше, меньше или равно
== !=	Сравнение на равенство, сравнение на неравенство
& ^ | && ||	Поразрядное и логическое: И, исключающее ИЛИ, ИЛИ, быстрой оценки И, ИЛИ
? :	Тернарная условная операция
операция=	Комбинированные операторы

Операции расположены в таблице от высокого приоритета к более низкому, сверху вниз и слева направо.

2. Управляющие конструкции

Для реализации алгоритма ветвления в коде в Java присутствует два условных оператора, это оператор **if … else** и оператор **switch**. Для реализации циклических алгоритмов есть три оператора, это **for**, **while**, **do..while**. Синтаксис и принцип работы похож на синтаксис и работу в языках C++ и C#.

Условные операторы

Синтаксис:

```
if (выражение) операция;
```

В круглых скобках может быть указано любое выражение, результатом которого должен быть тип **boolean**. В случае если в скобках выражение примет значение true, то будет выполнена «операция» следующая за круглыми

скобками, иначе управление программы перейдет к следующей строке кода.

Пример:

```
int a = 3;
if (a == 3) a = a + 2;
System.out.println(a);
```

Результат: 5

Если в результате выполнения условия в операторе **if** необходимо выполнить несколько операций, необходимо использовать «блочный разделитель» (фигурные скобки) для обозначения блока операций, которые будут выполнены.

Пример:

```
int a = 3;
if (a == 3) { // начало блока
 a = a + 2;
 a--;
 // конец блока }
System.out.println(a);
```

Результат: 4

Совместно с оператором **if** возможно использоваться ключевого слова **else**.

Синтаксис:

```
if (выражение) операция1;
else операция2;
```

В случае если в скобках «выражение» примет значение false, то будет выполнена «операция 2», «операция 1» выполнена не будет.

Пример:

```
int x = 3;
 if (x > 3) System.out.println(++x);
 else  System.out.println(--x);
```

Результат: 2

Для **else**, в случае если есть необходимость выполнить несколько команд, так же может быть использован блочный разделитель.

Пример:

```
int a = 3;
if (a != 3) {
 System.out.println(a - 3);
}
else
{
 System.out.println(a + 3);
}
```

Оператор **if** может использоваться совместно с **else**, для создания логических цепочек.

Пример:

```
int age = 18;
if (age < 1) System.out.println("Грудничок");
else if (age >= 1 && age < 16) System.out.
   println("Ребенок");
else if (age >= 16 && age < 19)
   System.out.println("Подросток");
else if (age >= 19) System.out.println("Взрослый");
```

Для упрощения реализации алгоритма множественного выбора в Java добавлен оператор **switch**. Оператор

анализирует выражение в скобках и передает управление одному из сценариев (**case**). Далее программа выполнит весь код, находящийся в сценарии до конца оператора **switch**. Значения вариантов в **case** не должны повторяться.

Синтаксис:

```
switch (выражение)
{
 case вариант1:
  операция1;
 case вариант2:
  операция2;
 case вариант3:
  операция3;
 default:
  операция;
}
```

В скобках допустимо наличие значений примитивных типов `byte, short, char, int*`.

(Начиная с версии JSE 5.0, в качестве выражения можно использовать перечисления. Начиная с версии JSE 7.0 , в качестве выражения можно использовать строки (String).).*

Пример:

```
int x = 2;
switch(x)
{
 case 1:
  System.out.println(1);
 case 2: // будет выбран этот сценарий и выполнены
   //все команды до концы блока switch
  System.out.println(2);
 case 3:
  System.out.println(3);
}
```

Результат: 2, 3

Для прерывания сценария в блоке switch можно использовать ключевое слово break.

Пример:

```
int x = 2;
switch(x)
{
 case 1:
  System.out.println(1);
  break; // прерывание выполнение сценария,
  // управление передается в конец за блок switch
 case 2:
  System.out.println(2);
  break;
 case 3:
  System.out.println(3);
  break;
}
```

Результат: 1

Ключевое слово default может использоваться в операторе **switch** в случае, если ни один сценариев не был выполнен. В блоке switch может присутствовать только одно ключевое слово default. Порядок расположения блоков **case** и **default** не имеет значения, но принято располагать блоки case по возрастанию значения, а default – в самом конце.

Пример:

```
int x = 5;
switch(x)
{
 case 3:
  System.out.println(3);
```

```
  break;
 case 1:
  System.out.println(1);
  break;
 default:
  System.out.println("default");
 case 2:
  System.out.println(2);
  break;
}
```

Результат: default