Программирование на языке Java. Операторы

Картузов А.В.

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

В Java имеется 44 встроенных оператора. Их можно разбить на 4 класса—арифметические, битовые, операторы сравнения и логические.

1. Арифметические операторы

Арифметические операторы используются для вычислений так же как в алгебре (см. таблицу со сводкой арифметических операторов ниже). 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.

Оператор	Результат	Оператор	Результат
+	Сложение	+ =	сложение с присваиванием
-	вычитание (также унарный минус)	-=	вычитание с присваиванием
*	Умножение	*=	умножение с присваиванием

Copyright © 1999-2004 Javable.com. All rights reserved.

/	Деление	/=	деление с присваиванием
%	деление по модулю	%=	деление по модулю с присваиванием
++	Инкремент	—	декремент

1.1. Четыре арифметических действия

Ниже, в качестве примера, приведена простая программа, демонстрирующая использование операторов. Обратите внимание на то, что операторы работают как с целыми литералами, так и с переменными.

```
class BasicMath  {
        public static void int a = 1 + 1;
                int  b = a *  3;
                main(String args[]) {
                        int c = b / 4;
                        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);
                        }
                }
```

Исполнив эту программу, вы должны получить приведенный ниже результат:

```
        C: \> java BasicMath
        a = 2
        b = 6
        c = 1
        d = 4
        e = -4
```

1.2. Оператор деления по модулю

Copyright © 1999-2004 Javable.com. All rights reserved.

Оператор деления по модулю, или оператор mod, обозначается символом %. Этот оператор возвращает остаток от деления первого операнда на второй. В отличие от C++, функция mod в Java работает не только с целыми, но и с вещественными типами. Приведенная ниже программа иллюстрирует работу этого оператора.

```
class Modulus {
        public static void main (String args []) {
                int x = 42;
                double y = 42.3;
                System.out.println("x mod 10 = " + x % 10);
                System.out.println("y mod 10 = " + y % 10);
                }
        }
```

Выполнив эту программу, вы получите следующий результат:

```
        C:\> Modulus
        x mod 10 = 2
        y mod 10 = 2.3
```

1.3. Арифметические операторы присваивания

Для каждого из арифметических операторов есть форма, в которой одновременно с заданной операцией выполняется присваивание. Ниже приведен пример, который иллюстрирует использование подобной разновидности операторов.

```
class OpEquals {
        public static void main(String args[]) {
                int a = 1;
                int b = 2;
                int c = 3;
                a += 5;
                b *= 4;
                c += a * b;
                c %= 6;
                System.out.println("a = " + a);
                System.out.println("b = " + b);
                System.out.println("c = " + c);
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
                }
        }
```

А вот и результат, полученный при запуске этой программы:

```
        C:> Java OpEquals
        a = 6
        b = 8
        c = 3
```

1.4. Инкремент и декремент

В C существует 2 оператора, называемых операторами инкремента и декремента (++ и —) и являющихся сокращенным вариантом записи для сложения или вычитания из операнда единицы. Эти операторы уникальны в том плане, что могут использоваться как в префиксной, так и в постфиксной форме. Следующий пример иллюстрирует использование операторов инкремента и декремента.

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

Результат выполнения данной программы будет таким:

```
        C:\ java IncDec
        a = 2
        b = 3
        c = 4
        d = 1
```


Copyright © 1999-2004 Javable.com. All rights reserved.

2. Целочисленные битовые операторы

Для целых числовых типов данных—long, int, short, char и byte, определен дополнительный набор операторов, с помощью которых можно проверять и модифицировать состояние отдельных битов соответствующих значений. В таблице приведена сводка таких операторов. Операторы битовой арифметики работают с каждым битом как с самостоятельной величиной.

Оператор	Результат	Оператор	Результат
~	побитовое унарное отрицание (NOT)		
&	побитовое И (AND)	&=	побитовое И (AND) с присваиванием
|	побитовое ИЛИ (OR)	|=	побитовое ИЛИ (OR) с присваиванием
^	побитовое исключающее ИЛИ (XOR)	^=	побитовое исключающее ИЛИ (XOR) с присваиванием
>>	сдвиг вправо	>> =	сдвиг вправо с присваиванием
>>>	сдвиг вправо с заполнением нулями	>>>=	сдвиг вправо с заполнением нулями с присваиванием
<<	сдвиг влево	<<=	сдвиг влево с присваиванием

2.1. Пример программы, манипулирующей с битами

В таблице, приведенной ниже, показано, как каждый из операторов битовой арифметики воздействует на возможные комбинации битов своих операндов. Приведенный после таблицы пример иллюстрирует использование этих операторов в программе на языке Java.

A	B	OR	AND	XOR	NOT A

Copyright © 1999-2004 Javable.com. All rights reserved.

0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

```
class Bitlogic {
        public static void main(String args []) {
                String binary[] = { "OOOO", "0001", "0010",
                            "0011", "0100", "0101",
                            "0110", "0111", "1000",
                            "1001", "1010", "1011",
                            "1100", "1101", "1110",
                            "1111" };
                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);
                int g = ~a & 0x0f;
                System.out.println(" a = " + binary[a]);
                System.out.println(" b = " + binary[b]);
                System.out.println(" ab = " + binary[c]);
                System.out.println(" a&b = " + binary[d]);
                System.out.println(" a^b = " + binary[e]);
                System.out.println("~a&b|a^~Ь = " + binary[f]);
                System.out.println(" ~a = " + binary[g]);
                }
        }
```

Ниже приведен результат, полученный при выполнении этой программы:

```
C: \> Java BitLogic
        a = 0011
        b = 0110
        a | b = 0111
        a & b = 0010
        a ^ b = 0101
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
        ~a & b | a & ~b = 0101
        ~a = 1100
```

3. Сдвиги влево и вправо

Оператор << выполняет сдвиг влево всех битов своего левого операнда на число позиций, заданное правым операндом. При этом часть битов в левых разрядах выходит за границы и теряется, а соответствующие правые позиции заполняются нулями. В предыдущей главе уже говорилось об автоматическом повышении типа всего выражения до int в том случае если в выражении присутствуют операнды типа int или целых типов меньшего размера. Если же хотя бы один из операндов в выражении имеет тип long, то и тип всего выражения повышается до long.

Оператор >> означает в языке Java сдвиг вправо. Он перемещает все биты своего левого операнда вправо на число позиций, заданное правым операндом. Когда биты левого операнда выдвигаются за самую правую позицию слова, они теряются. При сдвиге вправо освобождающиеся старшие (левые) разряды сдвигаемого числа заполняются предыдущим содержимым знакового разряда. Такое поведение называют расширением знакового разряда.

В следующей программе байтовое значение преобразуется в строку, содержащую его шестнадцатиричное представление. Обратите внимание—сдвинутое значение приходится маскировать, то есть логически умножать на значение 0x0f, для того, чтобы очистить заполняемые в результате расширения знака биты и понизить значение до пределов, допустимых при индексировании массива шестнадцатиричных цифр.

```
class  HexByte {
        static public 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 = 0x" + hex[(b >> 4) & 0x0f] +
                            hex[b & 0x0f]);
                }
        }
```

Ниже приведен результат работы этой программы:

Copyright © 1999-2004 Javable.com. All rights reserved.

```
        C:\> java HexByte
        b = 0xf1
```

4. Беззнаковый сдвиг вправо

Часто требуется, чтобы при сдвиге вправо расширение знакового разряда не происходило, а освобождающиеся левые разряды просто заполнялись бы нулями.

```
class ByteUShift   {
        static public 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;
                byte c = (byte) (b >> 4);
                byte d = (byte) (b >> 4);
                byte e = (byte) ((b & 0xff) >> 4);
                System.out.println(" b = 0x" + hex(b >> 4) & 0x0f] +
                            hex[b & 0x0f]);
                System.out.println(" b >> 4 =  0x" + hex[(c >> 4) & 0x0f] +
                            hex[c & 0x0f]);
                System.out.println("b >>> 4 = 0x" + hex[(d >> 4) & 0x0f] +
                            hex[d & 0x0f]);
                System.out.println("(b & 0xff) >> 4 = 0x" +
                            hex[(e >> 4) & 0x0f] + hex[e & 0x0f]);
                }
        }
```

Для этого примера переменную b можно было бы инициализировать произвольным отрицательным числом, мы использовали число с шестнадцатиричным представлением 0xf1. Переменной c присваивается результат знакового сдвига b вправо на 4 разряда. Как и ожидалось, расширение знакового разряда приводит к тому, что 0xf1 превращается в 0xff. Затем в переменную d заносится результат беззнакового сдвига b вправо на 4 разряда. Можно было бы ожидать, что в результате d содержит 0x0f, однако на деле мы снова получаем 0xff. Это—результат расширения знакового разряда, выполненного при автоматическом повышении типа переменной b до int перед операцией сдвига вправо. Наконец, в выражении для переменной e нам удается

Copyright © 1999-2004 Javable.com. All rights reserved.

добиться желаемого результата—значения 0x0f. Для этого нам пришлось перед сдвигом вправо логически умножить значение переменной b на маску 0xff, очистив таким образом старшие разряды, заполненные при автоматическом повышении типа. Обратите внимание, что при этом уже нет необходимости использовать беззнаковый сдвиг вправо, поскольку мы знаем состояние знакового бита после операции AND.

```
C: \> java ByteUShift
b = 0xf1
b >> 4 = 0xff
b >>> 4 = 0xff
b & 0xff) >> 4 = 0x0f
```

5. Битовые операторы присваивания

Так же, как и в случае арифметических операторов, у всех бинарных битовых операторов есть родственная форма, позволяющая автоматически присваивать результат операции левому операнду. В следующем примере создаются несколько целых переменных, с которыми с помощью операторов, указанных выше, выполняются различные операции.

```
class OpBitEquals {
        public static void main(String args[]) {
                int a = 1;
                int b = 2;
                int c = 3;
                a |= 4;
                b >>= 1;
                c <<= 1;
                a ^= c;
                System.out.println("a = " + a);
                System.out.println("b = " + b);
                System.out.println("c = " + c);
                }
        }
```

Результаты исполнения программы таковы:

```
C:\> Java OpBitEquals
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
a = 3
b = 1
c = 6
```

6. Операторы отношения

Для того, чтобы можно было сравнивать два значения, в Java имеется набор операторов, описывающих отношение и равенство. Список таких операторов приведен в таблице.

Оператор	Результат
==	равно
!=	не равно
>	больше
<	меньше
>=	больше или равно
<=	меньше или равно

Значения любых типов, включая целые и вещественные числа, символы, логические значения и ссылки, можно сравнивать, используя оператор проверки на равенство == и неравенство !=. Обратите внимание—в языке Java, так же, как в C и C++ проверка на равенство обозначается последовательностью (==). Один знак (=)—это оператор присваивания.

7. Булевы логические операторы

Булевы логические операторы, сводка которых приведена в таблице ниже, оперируют только с операндами типа boolean. Все бинарные логические операторы воспринимают в качестве операндов два значения типа boolean и возвращают результат того же типа.

Оператор	Результат	Оператор	Результат
&	логическое И (AND)	&=	И (AND) с присваиванием
|	логическое ИЛИ (OR)	=	ИЛИ (OR) с

Copyright © 1999-2004 Javable.com. All rights reserved.

			присваиванием
^	логическое исключающее ИЛИ (XOR)	^=	исключающее ИЛИ (XOR) с присваиванием
||	оператор OR быстрой оценки выражений (short circuit OR)	==	равно
&&	оператор AND быстрой оценки выражений (short circuit AND)	!=	не равно
!	логическое унарное отрицание (NOT)	?:	тернарный оператор if-then-else

Результаты воздействия логических операторов на различные комбинации значений операндов показаны в таблице.

A	B	OR	AND	XOR	NOT A
false	false	false	false	false	true
true	false	true	false	true	false
false	true	true	false	true	true
true	true	true	true	false	false

Программа, приведенная ниже, практически полностью повторяет уже знакомый вам пример BitLogic. Только но на этот раз мы работаем с булевыми логическими значениями.

```
class BoolLogic {
        public static void main(String args[]) {
                boolean a = true;
                boolean b = false;
                boolean c = a | b;
                boolean d = a & b;
                boolean e = a ^ b;
                boolean f = (!a & b) | (a & !b);
                boolean g = !a;
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
                System.out.println(" a = " + a);
                System.out.println(" b = " + b);
                System.out.println(" a|b = " + c);
                System.out.println(" a&b = " + d);
                System.out.println(" a^b = " + e);
                System.out.println("!a&b|a&!b = " + f);
                System.out.println(" !a = " + g);
                }
        }


C: \> Java BoolLogic
        a = true
        b = false
        a|b = true
        a&b = false
        a^b = true
        !a&b|a&!b = true
        !a = false
```

Операторы быстрой оценки логических выражений (short circuit logical operators)

Существуют два интересных дополнения к набору логических операторов. Это—альтернативные версии операторов AND и OR, служащие для быстрой оценки логических выражений. Вы знаете, что если первый операнд оператора OR имеет значение true, то независимо от значения второго операнда результатом операции будет величина true. Аналогично в случае оператора AND, если первый операнд—false, то значение второго операнда на результат не влияет—он всегда будет равен false. Если вы в используете операторы && и || вместо обычных форм & и |, то Java не производит оценку правого операнда логического выражения, если ответ ясен из значения левого операнда. Общепринятой практикой является использование операторов && и || практически во всех случаях оценки булевых логических выражений. Версии этих операторов & и | применяются только в битовой арифметике.

8. Тернарный оператор if-then-else

Общая форма оператора if-then-use такова:

Copyright © 1999-2004 Javable.com. All rights reserved.

```
        выражение1? выражение2: выражение3
```

В качестве первого операнда—<выражение1>—может быть использовано любое выражение, результатом которого является значение типа boolean. Если результат равен true, то выполняется оператор, заданный вторым операндом, то есть, <выражение2>. Если же первый операнд равен false, то выполняется третий операнд—<выражение3>. Второй и третий операнды, то есть <выражение2> и <выражение3>, должны возвращать значения одного типа и не должны иметь тип void.

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

```
class Ternary {
        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("a = " + a);
                System.out.println("b = " + b);
                System.out.println("c = " + c);
                System.out.println("d = " + d);
                System.out.println("a / b = " + e);
                System.out.println("c / d = " + f);
                }
        }
```

При выполнении этой программы исключительной ситуации деления на нуль не возникает и выводятся следующие результаты:

```
C: \>java Ternary

        a = 42
        b = 2
        c = 99
        d = 0
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
        a / b = 21
        c / d  = 0
```

9. Приоритеты операторов

В Java действует определенный порядок, или приоритет, операций. В элементарной алгебре нас учили тому, что у умножения и деления более высокий приоритет, чем у сложения и вычитания. В программировании также приходится следить и за приоритетами операций. В таблице указаны в порядке убывания приоритеты всех операций языка Java.

Высший			
( )	[ ]	.	
~	!		
*	/	%	
+	-		
>>	>>>	<<	
>	>=	<	<=
==	!=		
&			
^			
|	<&		
&&			
||			
?:			
=	op=		
Низший			

В первой строке таблицы приведены три необычных оператора, о которых мы пока не говорили. Круглые скобки () используются для явной установки приоритета. Как вы

Copyright © 1999-2004 Javable.com. All rights reserved.

узнали из предыдущей главы, квадратные скобки [] используются для индексирования переменной-массива. Оператор . (точка) используется для выделения элементов из ссылки на объект—об этом мы поговорим в главе 7. Все же остальные операторы уже обсуждались в этой главе.

9.1. Явные приоритеты

Поскольку высший приоритет имеют круглые скобки, вы всегда можете добавить в выражение несколько пар скобок, если у вас есть сомнения по поводу порядка вычислений или вам просто хочется сделать свои код более читабельным.

```
a >> b + 3
```

Какому из двух выражений, a >> (b + 3) или (a >> b) + 3, соответствует первая строка? Поскольку у оператора сложения более высокий приоритет, чем у оператора сдвига, правильный ответ—a>> (b + a). Так что если вам требуется выполнить операцию (a>>b)+ 3 без скобок не обойтись.

10. Что дальше?

Итак, мы рассмотрели все виды операторов языка Java. Теперь вы можете сконструировать любое выражение с различными типами данных. В следующей главе познакомимся с конструкциями ветвления, организацией циклов и научимся управлять выполнением программы.

Copyright © 1999-2004 Javable.com. All rights reserved.