Курс: «Разработка программного обеспечения на Java»

Тема: Логические операторы, операторы ветвлений, побитовые операторы

План

1. Преобразование типов данных
2. Логические операторы
3. Таблица приоритетов операторов
4. Конструкции логического выбора. Операторы ветвлений
5. Побитовые операторы

1. Преобразование типов данных

Возникают ситуации, когда величину определенного типа необходимо присвоить переменной другого типа. Для некоторых типов это можно проделать и без приведения типа, в таких случаях говорят об автоматическом преобразовании типов.

В Java автоматическое преобразование возможно только в том случае, когда точности представления чисел переменной-приемника достаточно для хранения исходного значения. Автоматическое преобразование произойдет, например, при занесении значения переменной типа short в переменную типа int. Это называется расширением (widening) исходного типа, поскольку тип меньшей разрядности расширяется до большего совместимого типа.

Размера типа int всегда достаточно для хранения чисел из диапазона, допустимого для типа short, поэтому в подобных ситуациях оператора явного приведения типа не требуется. Обратное в большинстве случаев неверно, поэтому для занесения значения типа int в переменную типа short необходимо использовать оператор приведения типа. Данную процедуру иногда называют сужением (narrowing), поскольку вы явно сообщаете транслятору, что величину необходимо преобразовать, чтобы она уместилась в переменную нужного вам типа.

Для приведения величины к определенному типу перед ней нужно указать этот тип, заключенный в круглые скобки. Если бы при такой операции целое значение выходило за границы допустимого для типа short диапазона, оно было бы уменьшено путем деления по модулю на допустимый для short диапазон (результат деления по модулю на число — это остаток от деления на это число).

Пример:

```
int i1 = 100;
short s1 = (short)i1; // Явное привидение типов

short s2 = 50;
int i2 = s2; //  Автоматическое привидение типов
```

Если в выражении используются переменные типов byte, short и int, то во избежание переполнения тип всего выражения автоматически повышается до int. Если же в выражении тип хотя бы одной переменной — long, то и тип всего выражения тоже повышается до long. Все целые литералы, в конце которых не стоит символ L, имеют тип int.

Мы также можем приводить вещественный тип к целому, но при этом, вся десятичная часть будет отброшена.

Пример:
```
double d = 3.14;
int i = (int)d; // i = 3
```

2. Логические операторы

Логические операторы работают только с операндами типа boolean. Все логические операторы с двумя операндами объединяют два логических значения, образуя результирующее логическое значения.

Оператор	Описание
&	Логическое AND (И)
&&	Сокращённое AND
|	Логическое OR (ИЛИ)
||	Сокращённое OR
^	Логическое XOR (исключающее OR (ИЛИ))
!	Логическое унарное NOT (НЕ)
&=	AND с присваиванием
|=	OR с присваиванием
^=	XOR с присваиванием

Результаты выполнения логических операций:

A	B	A | B	A & B	A ^ B	!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

Логические операторы AND (И), OR (ИЛИ) и XOR (исключающее ИЛИ) выполняют над логическими величинами те же операции, что и их аналоги из семейства битовой логики. Унарный оператор NOT (НЕ) инвертирует логическое значение.

Существуют два дополнения к набору логических операторов. Это альтернативные версии операторов AND и OR, служащие для быстрой оценки логических выражений. 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 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.

3. Таблица приоритетов операторов

Приоритет (от высшего к низшему)	
1	() [] •
2	~ !
3	* / %
4	+ -
5	>> >>> <<
6	> >= <= <
7	!= ==
8	&
9	^
10	|
11	&&
12	||
13	?:
14	op= =

Наивысший приоритет имеют операторы: круглые скобки (), которые используются для явной установки приоритета. Квадратные скобки [] используются для индексирования переменной-массива. Оператор . (точка) используется для выделения элементов из ссылки на объект.

4. Конструкции логического выбора. Операторы ветвлений

Условный оператор if

Оператор if обеспечивает выполнение или пропуск инструкции в зависимости от указанного логического условия. Если условие истинно, то инструкция выполняется.

Синтаксис:
if (условие) инструкция;

На месте инструкции может быть как обычная инструкция (одна команда), так и составная инструкция (блок содержащий несколько команд, в том числе, другие условные операторы).

Примеры:

```
int a = 25;
if (a != 0) System.out.println( 100/a );

int b = 25;
if (b != 0) {
  System.out.println( 100/b );
}
```

Не смотря на то, что код в первом примере выглядит компактнее, только во втором примере можно было бы выполнить несколько инструкций в случае истинности условия.

У оператора if существует формат с дополнительной частью else:

```
if (условие)
        инструкция1;
else
        инструкция2;
```

В случае истинности условия выполняется простая или составная инструкция1, а в случае ложности простая или составная инструкция2.

Пример:

```
int a = 5;
if (a >= 0) System.out.println("a - положительное число»);
else
        System.out.println("a - отрицательное число»);
```

Оператор множественного выбора switch

Инструкция множественного выбора switch позволяет выполнять различные части программы в зависимости от того, какое значение будет иметь некоторая целочисленная переменной (её называют «переменной-переключателем», а «switch» с английского переводится как раз как «переключатель»).

Синтаксис:

```
switch (переключатель)  {
  case значение1:
    инструкция1;
    break;
  case значение2:
    инструкция2;
    break;
  …
  default:
    инструкция_по_умолчанию;
}
```

Рассмотрим элементы оператора:

- переключатель — это целочисленная переменная или выражение дающее целочисленный результат;
- значение1, значение2, … — это целочисленные литералы, с которыми будет сравниваться значение переключателя. Если переключатель равен значениюN, то программа будет выполняться со строки, следующей за case значениеN: и до ближайшего встреченного break, либо до конца блока switch (если break не встретится);
- default: — это метка инструкции после которой будут выполняться в том случае, если выше ни одно из значенийN не совпало с переключателем. Метка default — необязательная: можно её не включать в блок switch меток или не выполнять после неё никаких команд;
- инструкцияN — простая или составная инструкция. Притом в случае составной несколько команд не обязательно объединять в блок, можно их просто написать друг за другом разделяя с помощью «;» (и начиная новые строки для удобства).

5. Побитовые операторы

Все целочисленные типы представляются двоичными числами различной длины. Например, значение типа byte, равное 42, в двоичном представлении имеет вид 00101010, в котором каждая позиция представляет степень числа два. В данном случае, побитовый оператор интерпретируют операнд как последовательность бит, которая называется битовой маской. Результат оператора - новая битовая последовательность такого же размера.

Битовые операции обычно нужны лишь для экстремальной оптимизации. В обычных случаях битовые трюки затрудняют понимание кода, применяйте их только если они вам осознано необходимы.

Побитовые операторы	
~	Побитовый унарный оператор NOT
&	Побитовый AND
&=	Побитовый AND с присваиванием
|	Побитовый OR
|=	Побитовый OR с присваиванием
^	Побитовый исключающее OR
^=	Побитовый исключающее OR с присваиванием
>>	Сдвиг вправо
>>=	Сдвиг вправо с присваиванием
>>>	Сдвиг вправо с заполнением нулями
<<	Сдвиг влево
<<=	Сдвиг влево с присваиванием
>>>=	Сдвиг вправо с заполнением нулями с присваиванием

Результаты выполнения побитовых логических операторов:

A	B	A | B	A & B	A ^ B	~A
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0