3.Операции

Набор операций языка Java совпадает с набором операций языка C++. Арифметические операции приведены в табл.

Таблица 4.Арифметические операции

Операция	Результат
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток от деления
++	Инкремент (увеличение операнда на 1)
--	Декремент (уменьшение операнда на 1)
+=	Сложение с присваиванием
-=	Вычитание с присваиванием
*=	Умножение с присваиванием
/=	Деление с присваиванием
%=	Вычисление остатка и присваивание

Операцию деления по модулю можно применять как к целым числам, так и к числам с плавающей точкой. В C/C++ эта операция применима только к целым числам.

При выполнении следующего фрагмента

```
int x = 42;
double y = 42.3;
System.out.println("x mod 10 = " + x % 10);
System.out.println("y mod 10 = " + y % 10);
```

будет выведено:

```
x mod 10 = 2
y mod 10 = 2.3
```

3.1. Операции сравнения

Таблица 5. Операции сравнения

Операция	Результат
==	Равно
!=	Не равно
>	Больше чем
<	Меньше чём
>=	Больше чем или равно
<=	Меньше чем или равно

Результат этих операций - значение типа `boolean`. Операции отношений часто используются в выражениях, которые управляют условными операторами и различными операторами цикла.

Данные любых Java-типов, включая целые числа, числа с плавающей точкой, символы и булевские переменные, можно сравнивать, используя операции проверки равенства (==) и проверки неравенства (!=). Обратите внимание, что в Java (как в C и C++) равенство обозначается двумя знаками равенства, а не одним. (Напомним, что одиночный знак равенства (=) используется для операции присваивания.) С помощью операций упорядочивания (<, >, <=, >=) можно сравнивать только числовые типы (т. е., чтобы увидеть, какой операнд больше или меньше, чем другой, можно сравнивать только целочисленные, с плавающей точкой и символьные операнды).

Как было сказано ранее, результат, полученный операциями отношений, представляет собой значение типа `boolean`. Например, следующий кодовый фрагмент содержит правильные утверждения:

```
int a = 4;
int b = 1;
```

```
boolean c = a < b;
```

В этом случае результат выражения `a < b` (который есть `false`) сохраняется в `c`.

В программах C/C++ часто встречаются следующие типы операторов:

```
int done;
//…
if(!done) …  // Верно в C/C++
if(done) …   // но не в Java.
```

На языке Java эти операторы должны быть записаны так:

```
if(done == 0) ... // Это Java-стиль,
if(done != 0) ...
```

Причина в том, что Java не определяет `true` и `false` таким же образом, как C/C++. В C/C++ `true` — любое значение, отличное от нуля, а `false` — нуль. В Java `true` и `false` — нечисловые значения, которые не имеют отношения к нулю или не нулю. Поэтому, чтобы выполнить проверку на равенство нулю, вы должны явно использовать одну или несколько операций отношений.

3.2. Логические операции

Представленные в табл. 6 операции булевой логики работают только с операндами типа `boolean` и формируют результат типа `boolean`.

Таблица 6. Логические операции

Операция	Результат
&	Логическое И (Logical AND)
|	Логическое ИЛИ (Logical OR)
^	Логическое исключающее ИЛИ (Logical XOR (exclusive OR))
||	Укороченное ИЛИ (Shirt-ciruit OR)
&&	Укороченное И (Shirt-ciruit AND)
!	Логическое отрицание (Logical unary NOT)
&=	Присваивание с И (AND assignment)
|=	Присваивание с ИЛИ (OR assignment)
^=	Присваивание с исключающим ИЛИ (XOR assignment)
==	Равно (Equal to)
!=	Не равно (Not equal to)
?:	Троичная условная операция (Ternary if-then-else)

Табл.7 показывает эффект каждой логической операции.

Таблица 7. Результаты 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

3.3. Короткие логические операции

Java обеспечивает вторые версии булевских операций И и ИЛИ, известные как *укороченные* (short-circuit) логические операции. Как можно видеть в предшествующей таблице, односимвольная операция ИЛИ приводит к `true`-результату, когда операнд `A - true`, независимо от того, каков операнд `B`. Точно так же операция И приводит к `faise`-результату, когда `A – false`, независимо от того, каков `B`. Если используются двухзначные (||) и (&&), а не однозначные (|) и (&) формы логических операций ИЛИ и И, Java вообще не будет выполнять оценку правого

операнда перед вычислением всего выражения, так как результат выражения определяется одним левым операндом. Это полезно, когда правильное функционирование правого операнда зависит от того, является ли левый операнд `true` или `false`. Например, следующий кодовый фрагмент показывает, как вы можете воспользоваться преимуществом сокращения логической оценки, чтобы заранее обеспечить правильное выполнение операции деления (обходя любые оценки ее второго операнда во время выполнения):

```
if (denom != 0 && num / denom > 10)
```

Так как используется короткая форма операции И (&&), нет риска порождения исключительной ситуации при работе программы, когда `denom` окажется нулевым. Если бы эта строка программы была написана с использованием версии И, состоящей из одного знака &, то во время выполнения программы нужно было бы оценивать обе стороны соответствующего выражения, что при нулевом `denom` вызвало бы исключительную ситуацию.

Использование короткой формы И и ИЛИ стало стандартной практикой в случаях использования булевской логики, оставляя версии с одиночным символом исключительно для поразрядных операций. Однако имеются исключения из этого правила. Например, рассмотрим следующий оператор:

```
if(c==1 & e++ < 100)  d = 100;
```

Использование однозначной операции (&) гарантирует здесь, что операция инкремента будет применяться к `e` вне зависимости от того, равна переменная `c` единице или нет.