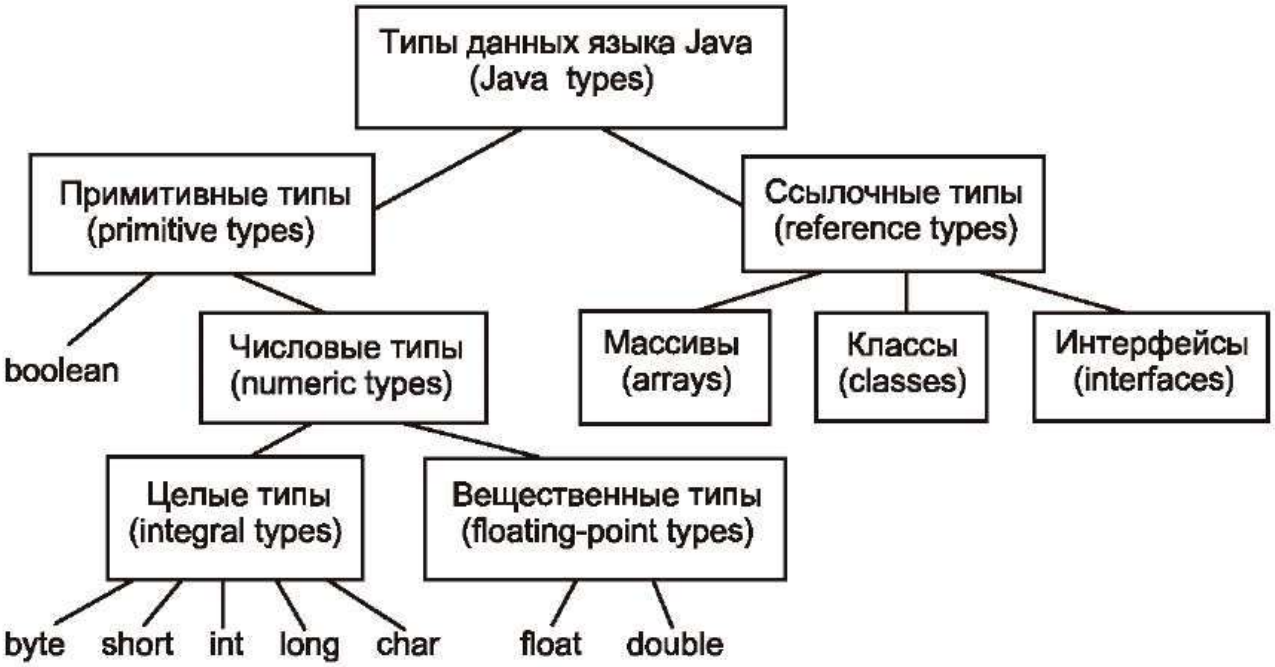
Типы данных Java

Язык программирования Java является языком со строгой типизацией. Это означает, что любая переменная и любое выражение имеют какой-либо тип данных, известный уже на момент компиляции приложения. Тип данных определяет:

- внутреннее представление данных в памяти компьютера;
- множество значений, которые могут принимать величины этого типа;
- операции, которые могут выполняться над величинами этого типа.

Введение типов данных является одной из базовых концепций языка Java, заключающейся в том, что при выполнении операции присваивания переменной значения выражения, переменная и выражение должны быть одного типа. Такая проверка выполняется компилятором, что значительно упрощает поиск ошибок и приводит к повышению надежности программы. Все типы исходных данных, встроенные в язык Java, делятся на две группы: примитивные типы и ссылочные типы.



Примитивные типы Java – это типы, для которых в ячейке памяти содержатся непосредственно данные – скалярные величины, которые не являются объектами и имеют значения по умолчанию. При выполнении операции присваивания для переменных, имеющих примитивный тип, выполняется копирование данных из одной ячейки памяти в другую. Примитивными являются логический тип boolean, целые числовые типы byte, short, int, long, char и плавающие числовые типы float и double.

Значения логического типа boolean возникают в результате различных сравнений и используются, главным образом, в условных операторах и операторах циклов. Логических значений всего два: true (истина) и false (ложь) – это служебные слова Java. Никакие другие значения переменной логического типа присвоить невозможно. Величина типа boolean занимает в памяти 1 байт. В отличие от языков C и C++, в Java константа true не равна 1, а константа false не равна 0.

Для целых типов спецификация языка Java определяет разрядность (количество байтов, выделяемых для хранения значений типа в оперативной памяти) и диапазон значений следующим образом:

Тип	Разрядность (байт)	Минимальное значение	Максимальное значение
byte	1	-128	127
short	2	-32768	32767
int	4	-2147483648	2147483647
long	8	-922372036854775808	922372036854775807
char	2	0	65535

Для работы с символами в Java используется тип данных char, в котором символ представлен в 16-битным значением в Unicode-таблице. Unicode – это стандарт кодирования символов, который позволяет предоставить знаки всех языков мира. Каждый символ представлен двумя байтами, которые позволяют хранить в себе целое число от 0 до 65535, что позволяет использовать 65536 различных символов. Однако требуется специальное обозначение, чтобы иметь возможность задавать в программе любой символ Unicode, поскольку никакая клавиатура не позволит вводить 65536 различных знаков. Код любого символа в кодировке Unicode набирается в апострофах после обратной наклонной черты и латинской буквы u четырьмя шестнадцатеричными цифрами. Например, если в программу нужно вставить знак с кодом 6917, необходимо его представить в шестнадцатеричном формате (1B05) и записать это значение как \u1B05, причем буква u должна быть строчной, а шестнадцатеричные цифры A, B, C, D, E, F можно использовать как заглавные, так и строчные. Таким образом, можно закодировать все символы Unicode последовательностями от \u0000 до \uFFFF.

Вещественные числа могут быть представлены в двух форматах:

- с фиксированной точкой – совпадает с обычной математической записью десятичного числа с дробной частью, дробная часть отделяется от целой части с помощью точки;
- с плавающей точкой – применяется при записи очень больших или очень малых чисел. В этом формате число, стоящее перед символом «E», умножается на число 10 в степени, указанной после символа «E».

Значения вещественных типов в компьютере представляются приближенно и определяются двумя типами: float – с одинарной точностью, double – с двойной точностью:

Тип	Разрядность (байт)	Диапазон значений	Точность
float	4	3.4e-38 < |x| < 3.4e38	7-8 цифр
double	8	1.7e-308 < |x| < 1.7e308	17 цифр

К обычным вещественным числам добавляются еще три значения:

- положительная бесконечность, выражаемая константой POSITIVE_INFINITY и возникающая при переполнении положительного значения;
- отрицательная бесконечность, выражаемая константой NEGATIVE_INFINITY и возникающая при переполнении отрицательного значения;
- «не число», выражаемое константой NaN (Not a Number) и возникающее при делении вещественного числа на нуль или умножении нуля на бесконечность.

Кроме того, спецификация языка Java различает положительный и отрицательный нули (+0.0 и -0.0), возникающие при делении на бесконечность соответствующего знака.

Ссылочными типами называются типы данных, для которых в ячейке памяти содержится не сами данные, а только адреса этих данных, то есть ссылки на данные. К ссылочным типам относятся классы, интерфейсы и массивы. Существует также специальный нулевой тип – это зарезервированное слово null, обозначающее нулевой адрес.

Свойства ссылочного типа данных:

- при выполнении операции присваивания в ссылочную переменную заносится адрес данных, а не сами данные;
- непосредственный доступ к адресу, хранящемуся в ссылочных переменных, в языке Java отсутствует;
- если ссылочной переменной не присвоено значение, в ней хранится значение null;
- ссылки можно присваивать друг другу, только если они совместимы по типам;
- ссылочным переменным можно присваивать значение null.

Преобразование примитивных типов данных

Тождественное преобразование является самым простым – в Java преобразование выражения любого типа к точно такому же типу всегда допустимо и успешно выполняется.

Расширение типа для примитивных типов данных означает, что осуществляется переход от менее емкого типа к более емкому – например, от типа byte (длина 1 байт) к типу int (длина 4 байта). Такие преобразования безопасны в том смысле, что новый тип всегда гарантированно вмещает в себя все данные, которые хранились в старом типе, и таким образом не происходит потери данных. Именно поэтому компилятор осуществляет это преобразование сам, никаких специальных действий для этого

предпринимать не требуется. Расширяющими являются следующие 19 преобразований:

- от byte к short, int, long, float, double;
- от short к int, long, float, double;
- от char к int, long, float, double;
- от int к long, float, double;
- от long к float, double;
- от float к double.

Нельзя провести преобразование к типу char от типов меньшей (byte) или равной (short) длины, и, наоборот, к short от char без потери данных. Это связано с тем, что char, в отличие от остальных целочисленных типов, является беззнаковым типом данных.

Сужение типа для примитивных типов данных означает, что переход осуществляется от более емкого типа к менее емкому. При таком преобразовании всегда есть риск потерять данные. Например, если число типа int было больше 127, то при приведении его к типу byte значения битов старше восьмого будут потеряны. В Java такое преобразование должно совершаться явным образом, т.е. программист в коде должен явно указать, что он намеревается осуществить такое преобразование и готов идти на потерю данных. Следующие 23 преобразования являются сужающими:

- от byte к char;
- от short к byte, char;
- от char к byte, short;
- от int к byte, short, char;
- от long к byte, short, char, int;
- от float к byte, short, char, int, long;
- от double к byte, short, char, int, long, float.

При сужении целочисленного типа все старшие биты, не попадающие в новый тип, просто отбрасываются, не производится округления или каких-либо других действий для получения более корректного результата. Операция явного преобразования типов предполагает указание перед переменной или выражением в скобках того типа, к которому надо преобразовать значение. Например, если написать (byte) перед значением типа int, мы в качестве результата получим значение типа byte.

Выражения и операторы

Выражения являются основными составляющими любого Java-приложения. Выражения строятся с использованием значений, переменных, операторов и вызовов методов.

Операторы в языке Java – это специальные символы, которые сообщают о том, что необходимо выполнить какую-либо операцию с некоторыми данными – операндами. Операндом является переменная или значение, участвующее в операции.

Различают *унарные операции* – они выполняются над одним операндом, *бинарные* - над двумя операндами, а также *тернарные* - выполняются над тремя операндами. Операторы, которые расположены перед операндами, называются префиксными, операторы, которые расположены после операндов – постфиксными, операторы, расположенные между двумя операндами - инфиксными операторами.

Операторы языка Java можно разделить на следующие категории: арифметические операторы, логические операторы, операторы сравнения, побитовые операторы и операторы присваивания, а также тернарный оператор.

Арифметические операторы служат для выполнения арифметических действий над числами.

Оператор	**Описание**
+	сохранение знака числа
-	смена знака числа
+	сложение
-	вычитание
*	умножение
/	деление
%	остаток от деления
++	инкремент (увеличение на единицу)
--	декремент (уменьшение на единицу)

Операторы смены знака, инкремента и декремента являются унарными операторами. Унарный оператор извлекает из переменной значение, изменяет его и снова помещает в ту же переменную. Остальные арифметические операторы являются бинарными – они всегда имеют два операнда и помещают результат в третью переменную.

Операторы инкремента и декремента можно ставить как перед операндом, так и после него. Если оператор стоит перед операндом, то значение операнда сначала изменяется на единицу, а уже потом используется в дальнейших вычислениях. Если же оператор стоит после операнда, то его значение сначала вычисляется, а уже потом изменяется на единицу.

При выполнении арифметической операции над операндами разных типов результат операции будет иметь наибольший тип:

- если один из операндов имеет тип double, то результат выражения имеет тип double;
- если один из операндов имеет тип float, то результат выражения имеет тип float;
- если один из операндов имеет тип long, то результат выражения имеет тип long, иначе результат выражения имеет тип int.

Если результат операции с целочисленными данными выходит за диапазон типа данных, то старшие биты отбрасываются, и результирующее значение будет неверным. При попытке деления целочисленного значения на 0 возникает исключение java.lang.ArithmeticException. При выполнении операций над числами с плавающей точкой при выходе за верхнюю или нижнюю границу диапазона получается POSITIVE_INFINITY и NEGATIVE_INFINITY соответственно, а при получении слишком маленького числа, которое не может быть нормально сохранено в этом типе данных, результат равен -0.0 или +0.0.

Логические операторы выполняют действия над логическими значениями.

Оператор	**Описание**
!	логическое НЕ
|	логическое ИЛИ
&	логическое И
^	исключающее ИЛИ
||	сокращённое логическое ИЛИ
&&	сокращённое логическое И

Операция ! является унарной и осуществляет инвертирование значения: если значение операнда равно true, то результатом выполнения операции будет false, и наоборот. Все остальные логические операции являются бинарными. Результатом операции | будет true, если значение любого из операндов равно true; результатом операции & будет true, если значение каждого из операндов равно true; результатом операции ^ будет true, если значение только одного из операндов равно true.

Сокращённые логические операции || и && выполняются аналогично операциям | и &, однако правый операнд сокращенных операций вычисляется только в том случае, если от него зависит результат операции – если левый операнд операции | имеет значение false, или левый операнд операции & имеет значение true.

Операторы сравнения являются бинарными и предназначены для сравнения операндов по значению. Результатом выполнения операторов сравнения является логическое значение.

Оператор	**Описание**
==	равно
!=	не равно
>	больше
<	меньше
>=	больше или равно
<=	меньше или равно

Побитовые операторы выполняют действия над целочисленными значениями и применяются к каждому отдельному биту каждого операнда.

Оператор	Описание
~	поразрядное логическое НЕ
|	поразрядное логическое ИЛИ
&	поразрядное логическое И
^	поразрядное исключающее ИЛИ
<<	поразрядный сдвиг влево
>>	поразрядный сдвиг вправо
>>>	поразрядный сдвиг вправо без учёта знака

Оператор ~ называется также побитовым дополнением, является унарным оператором и инвертирует все биты операнда: если значение операнда равно 0, то результатом выполнения операции будет 1, и наоборот. Все остальные логические операции являются бинарными. Результатом выполнения оператора | является 1, если соответствующий бит в любом из операндов равен 1; результатом выполнения оператора & является 1, если соответствующий бит в каждом из операндов равен 1; результатом выполнения оператора ^ является 1, если соответствующий бит только в одном из операндов равен 1.

Оператор << смещает все биты значения влево на указанное количество позиций. Крайние левые биты значения при сдвиге теряются, а расположенные в крайних правых позициях биты, освобожденные в результате сдвига, заполняются нулями.

Оператор >> смещает все биты значения вправо на указанное количество позиций. Крайние правые биты значения при сдвиге теряются, а расположенные в крайних левых позициях биты, освобожденные в результате сдвига, заполняются предыдущим содержимым старшего бита, что позволяет сохранить знак значения.

Оператор >>> смещает все биты значения вправо на указанное количество позиций. Крайние правые биты значения при сдвиге теряются, а расположенные в крайних левых позициях биты, освобожденные в результате сдвига, заполняются нулями, что может изменить знак значения.

Оператор присваивания записывается при помощи символа = и является бинарным оператором. Он вычисляет значение своего правого операнда и присваивает его левому операнду, а также выдает в качестве результата присвоенное значение, которое может быть использовано другими операциями. При этом тип данных переменной в левой части оператора присваивания должен быть совместим с типом данных значения в его правой части. Последовательность из нескольких операций присваивания выполняется справа налево.

Составные операторы присваивания получаются в результате комбинирования арифметических и побитовых операторов с оператором присваивания: +=, -=,*=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=. Их использование позволяет упростить запись

выражения в тех случаях, когда текущее значение переменной изменяется, а затем присваивается этой же переменной.

Тернарный оператор использует три операнда и записывается в следующей форме:

Условие ? Выражение1 : Выражение2

Если Условие имеет значение true, то вычисляется Выражение1, и его результат становится результатом выполнения всего оператора. Если же Условие равно false, то вычисляется Выражение2, и его значение становится результатом работы оператора. Оба операнда Выражение1 и Выражение2 должны возвращать значения одинакового или совместимого типа.

Приоритет операций

Все операции, присутствующие в записи выражения, выполняются поочерёдно в соответствии с их приоритетом. Чем выше оператор расположен в приведённой ниже таблице, тем больше у него приоритет.

Знак операции	Наименование	Ассоциативность
++, --	постинкремент, постдекремент	справа налево
++, --, +, -, ~, !	преинкремент, предекремент, унарный плюс, унарный минус, поразрядное дополнение, логическое «не»	справа налево
*, /, %	умножение, деление, остаток от деления	слева направо
+, -	сложение, вычитание	слева направо
<<, >>, >>>	сдвиг влево, сдвиг вправо, беззнаковый сдвиг вправо	слева направо
<, >, <=, >=, instanceof	меньше, больше, меньше или равно, больше или равно, сравнение типа	слева направо
==, !=	равно, не равно	слева направо
&	битовое «и»	слева направо
^	исключающее «или»	слева направо
|	битовое «или»	слева направо
&&	логическое «и»	слева направо
||	логическое «или»	слева направо
?:	тернарный оператор	слева направо
=, +=, -=,*=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=	операторы присваивания	справа налево

Операции, которые расположены на одном уровне в таблице, выполняются согласно ассоциативности выполнения (слева направо или справа налево). При ассоциативности слева направо сначала выполняются операции, записанные в выражении левее, при ассоциативности справа налево – наоборот.

Приоритет выполнения операций можно изменить с помощью скобок (операции в скобках выполняются раньше). Если скобки отсутствуют, сначала выполняются более приоритетные операции. Применение круглых скобок, даже избыточных, не ведет к снижению производительности программы, поэтому скобки можно также использовать для повышения лёгкости чтения программного кода.