# Лекция 2. Типизация в Java

# Занятие 1

**Типизация** *– это способ определить свойства и поведение объекта и защититься от использования объектов одного класса вместо другого, или по крайней мере управлять таким использованием.*

В Java, как и во всех современных языках программирования, поддерживается несколько типов данных. Однако Java является строго типизированным языком. Это означает, что тип любой переменной объявлен однозначно и, с момента объявления этой переменной, никакой другой тип задать ей невозможно. Сделано это из соображений безопасности: все переменные проверяются на ***совместимость*** типов. В Java, в отличие от языков с нестрогой типизацией, отсутствует автоматическое приведение типов. Несовпадение типов приводит не к предупреждению о возможной ошибке, а к ошибке и неработоспособности кода. Для каждого типа определены наборы допустимых значений и разрешенных операций.

В Java существует 2 типа данных: простые (примитивные) и ссылочные.

|  |
| --- |
|  |
| Рис. 1. Типы данных в Java |

**Переменные простого типа хранят заданное значение, в то время как ссылочные типы хранят ссылку (адрес) на созданный объект в памяти.**

Каждой переменной в памяти выделяется определённая область для хранения ее значения. Переменной простого типа присваивается ***само значение***, поэтому говорят, что ***переменная простого типа хранит значение***. Переменной ссылочного типа возвращается ***ссылка (адрес) на значение*** в памяти, поэтому говорят, что переменная ***ссылочного типа хранит ссылку на значение***.

# Простые (примитивные) типы данных

Java – объектно-ориентированный язык, однако простые типы представляют собой одиночные значения, а не сложные объекты. Это было сделано из соображений повышения производительности. Простые типы данных хранят свои значения в специально отведенной для них области памяти.

В Java существует 8 простых типов, разделенных на 3 группы: числовые (целочисленные и вещественные), логические и символьные.

В таблице 1 приведен список операций сравнения значений переменных простых численных типов.

|  |
| --- |
| Таблица 1. Операции сравнения простых типов данных |
| |  |  | | --- | --- | | Операция | Обозначение | | Равно | == | | Не равно | != | | Больше | > | | Меньше | < | | Больше или равно | >= | | Меньше или равно | <= | |

# Простые числовые типы

Над всеми числовыми типами выполняются арифметические операции, приведенные в таблице 2.

|  |
| --- |
| Талица 2. Арифметические операции над числовыми типами данных |
| |  |  | | --- | --- | | Операция | Обозначение | | Сложение | a + b | | Вычитание | a - b | | Умножение | a \* b | | Деление | a / b | | Взятие остатка | a % b | | Инкремент | a++, ++a | | Декремент | a--, --a | | Увеличить a на b | a += b | | Увеличить a в b раз | a \*= b | | Уменьшить a на b | a -= b | | Уменьшить  a в b раз | a /= b | |

# Целочисленные типы

Целочисленными являются все положительные и отрицательные числа, не имеющие дробной части. К целочисленным типам применимы все стандартные арифметические операции с учётом приоритета и скобок, а также операции сравнения (см. табл. 1), инкремент, дикремент, побитовые операции и операции сдвига.

Существует четыре целочисленных типа: **byte, short, int, long**.

|  |
| --- |
| Таблица 3. Целочисленные типы данных |
| |  |  |  |  | | --- | --- | --- | --- | | Тип | Размер | Левая граница числового диапазона | Правая граница числового диапазона | | byte | 8 бит (1 байт) | -128 | 127 | | short | 16 бит (2 байта) | -32768 | 32767 | | int | 32 бита (4 байта) | -2147483648 | 2147483647 | | long | 64 бит (8 байт) | -9223372036854775808 | 9223372036854775807 | |

Над целочисленными типами выполняются следующие побитовые логические (побитовые) операции (см. табл. 4).

|  |
| --- |
| Таблица 4. Побитовые логические операции над целочисленными типами данных |
| |  |  | | --- | --- | | Операция | Обозначение | | Побитовое «и» (AND) | a & b | | Побитовое «или» (OR) | a | b | | Побитовое исключающее «или» (XOR) | a ^ b | | Побитовое отрицание (NOT) | ~a | | Побитовый знаковый сдвиг влево | a << b | | Побитовый знаковый сдвиг вправо | a >> b | | Побитовый беззнаковый сдвиг вправо | a >>> b | |

Побитовые логические операции могут также быть применены к переменным типа byte и short, но в этом случае переменная такого типа будет неявно приведена к типу int, и только после этого к ней будет применена логическая операция.

# Вещественные типы

Числа с плавающей точкой (иногда их называют действительными или вещественными числами) применяются при вычислении выражений, в которых требуется точность до десятичного знака. Например, вычисление квадратного корня, значений синуса, косинуса и т.п. Существует два типа с плавающей точкой: ***float*** и ***double***, которые представляют числа одинарной и двойной точности. Над вещественными типами справедливы все арифметические операции, операции сравнения, но не выполняются побитовые.

|  |
| --- |
| Таблица 5. Вещественные типы данных |
| |  |  |  |  | | --- | --- | --- | --- | | Тип | Размер | Левая граница числового диапазона | Правая граница числового диапазона | | float | 32 бита (4 байта) | -3.4E+38 | 3.4E+38 | | double | 64 бита (8 байт) | -1.7E+308 | 1.7E+308 | |

Кроме того, в переменных такого типа можно хранить не только числа. У них есть еще три стандартных зарезервированных в языке значения (табл. 6).

|  |
| --- |
| Таблица 6. Специальные значения вещественных типов данных |
| |  |  |  | | --- | --- | --- | | Описание | Обозначение | Зарезервированное значение | | +∞ | infinity | Float.POSITIVE\_INFINITY Double.POSITIVE\_INFINITY | | -∞ | -infinity | Float.NEGATIVE\_INFINITY Double.NEGATIVE\_INFINITY | | Not a Number (неопределенность) | NaN | Float.NaN Double.NaN | |

Особенностью типов float и double является значение NaN. NaN никогда не равно самому себе и ничему другому.

# Символьный тип

**char** – символьный тип данных. Каждому символу соответствует определённое число из таблицы Unicode, т.е., фактически, это все буквы, числа и специальные символы существующие на сегодняшний день. Так как char-символы являются Unicode-символами, для хранения char-переменных выделяется 16 бит (2 байта). Диапазон допустимых значений - от 0 до 65536 (отрицательных значений не существует).

С переменными этого типа можно производить только операции присваивания, но зато различными способами. Самый простой из них выглядит так:

char c = 'b';

Символ можно представить также в виде его кода, записанного в восьмиричной системе счисления:

char c = '\077';

где 077 – это обязательно трехзначное число, не большее чем 377 (=255 в десятичной системе счисления),

или же в 16-ричной системе счисления следующим образом:

char c = '\u12a4';

кроме того, существуют специальные символы, такие как знак абзаца, разрыв страницы и др. Знак абзаца запишется, например, так:

char c = '\n';

Не следует путать символ 'a' со строкой "a", состоящей из одного символа. На экране монитора они выглядят одинаково, но в программах ведут себя по разному.

Хотя тип char используется для хранения Unicode-символов, его можно использовать как целочисленный тип, используя сложение или вычитание.

char ch1;

ch1 = 'x';

textViewInfo.append("ch1 содержит " + ch1);

ch1++; // увеличим на единицу

textViewInfo.append("ch1 содержит " + ch1);

В результате получим:

ch1 содержит x

ch1 содержит y

# Логический тип

Тип **boolean** предназначен для хранения логических значений и может принимать только одно из двух возможных значений: ***true*** или ***false***. Буленовский тип данных в Java не поддерживает тождественность true = 1 и false =0, как, например, С++.

Данный тип всегда возвращается при использовании операторов сравнения (см. табл. 1). Также он используется в управляющих операторах **if** (оператор условия) и **for** (оператор цикла).

boolean check = true;

В операторах if используется укороченная запись при значении true:

if (check == true) ... // необязательный вариант

if (check) ... // укороченная запись

# Приведение типов

Иногда возникают ситуации, когда переменной одного типа необходимо присвоить значение другого типа. Для некоторых типов это можно проделать и без *явного (автоматического)* приведения.

**В Java автоматическое преобразование возможно только в том случае, когда точности представления чисел переменной-приемника достаточно для хранения исходного значения**.

Такое преобразование происходит, например, при занесении литеральной константы или значения переменной типа byte или short в переменную типа int. Это называется расширением (*widening*) или повышением (*promotion*),поскольку тип меньшей разрядности расширяется (повышается) до большего совместимого типа. Размера типа int всегда достаточно для хранения чисел из диапазона, допустимого для типа byte, поэтому в подобных ситуациях оператора явного приведения типа не требуется.

Обратное в большинстве случаев неверно, поэтому для занесения значения типа int в переменную типа byte необходимо использовать оператор приведения типа. Эту 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.

В приведенном ниже фрагменте кода демонстрируется приведение типа источника (переменной типа int) к типу приемника (переменной типа byte). Если бы при такой операции целое значение выходило за границы допустимого для типа byte диапазона, оно было бы уменьшено путем деления по модулю на допустимый для byte диапазон (результат деления по модулю на число — это остаток от деления на это число),

int а = 100;   
byte b = (byte) а;

# Автоматическое преобразование типов в выражениях

При вычислениях значения выражения точность, требуемая для хранения промежуточных результатов, зачастую должна быть выше, чем требуется для представления окончательного результата.

byte а = 40;   
byte b = 50;   
byte с = 100;   
int d = a\* b / с;

Результат промежуточного выражения (а\*b) вполне может выйти за диапазон допустимых для типа byte значений. Именно поэтому Java автоматически повышает тип каждой части выражения до типа int, так что для промежуточного результата (а\* b) хватает места.

Автоматическое преобразование типа иногда может оказаться причиной неожиданных сообщений транслятора об ошибках. Например, показанный ниже код, хотя и выглядит вполне корректным, приводит к сообщению об ошибке на фазе трансляции. В нем мы пытаемся записать значение 50\*2, которое должно прекрасно уместиться в тип byte, в байтовую переменную. Но из-за автоматического преобразования типа результата в int мы получаем сообщение об ошибке от транслятора — ведь при занесении int в byte может произойти потеря точности.

byte b = 50;   
b = b\* 2:   
^Incompatible type for =. Explicit cast needed to convert int to byte.   
(Несовместимый тип для =. Необходимо явное преобразование int в byte)

Исправленный текст:

byte b = 50;   
b = (byte) (b\* 2);

что приводит к занесению в b правильного значения 100.

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. Не забывайте, что все целые литералы, в конце которых не стоит символ L (или 1), имеют тип int.

Если выражение содержит операнды типа float, то и тип всего выражения автоматически повышается до float. Если же хотя бы один из операндов имеет тип double, то тип всего выражения повышается до double. По умолчанию Java рассматривает все литералы с плавающей точкой как имеющие тип double. Приведенная ниже программа показывает, как повышается тип каждой величины в выражении для достижения соответствия со вторым операндом каждого бинарного оператора.

class Promote {   
 public static void main (String args []) {   
 byte b= 42;   
 char с = 'a’;   
 shorts = 1024;   
 int i = 50000;   
 float f = 5.67f;   
 doubled =.1234;   
 double result = (f\*b) + (i/ c) - (d\* s);   
 System, out. println ((f\* b)+ "+ "+ (i / c)+ " -" + (d\* s));   
 System, out. println ("result = "+ result);

}   
}

Подвыражение f\*b — это число типа float, умноженное на число типа byte, поэтому его тип автоматически повышается до float. Тип следующего подвыражения i / с (int, деленный на char) повышается до int. Аналогично этому тип подвыражения d\*s (double, умноженный на short) повышается до double. На следующем шаге вычислений мы имеем дело с тремя промежуточными результатами типов float, int и double. Сначала при сложении первых двух тип int повышается до float и получается результат типа float. При вычитании из него значения типа double тип результата повышается до double. Окончательный результат всего выражения — значение типа double.

В приведенном ниже примере создаются переменные каждого из простых типов и выводятся значения этих переменных.

class SimpleTypes {   
 public static void main(String args []) {   
 byte b = 0x55;   
 short s = 0x55ff;   
 int i = 1000000;   
 long l = 0xffffffffL;   
 char с = ’a’;   
 float f= .25f;   
 double d = .00001234;   
 boolean bool = true;   
 System.out.println("byte b = " + b);   
 System.out.println("short s = " +s);   
 System.out.println("int i =” + i);   
 System.out.println("long 1 = " + l);   
 System.out.println("char с =” + с);   
 System.out.println("float f = " + f);   
 System.out.println("double d = " + d);   
 System.out.println("boolean bool =” + bool); }   
}

Запустив эту программу, вы должны получить результат, показанный ниже:

byte b = 85   
shorts = 22015   
int i = 1000000   
long 1 = 4294967295   
char с = a   
float f = 0.25   
double d=1.234e-005   
boolean bool = true

Обратите внимание на то, что [целые числа](http://edu.sernam.ru/book_el_math.php?id=5) печатаются в десятичном представлении, хотя мы задавали значения некоторых из них в шестнадцатеричном формате.

# Практическая часть

# Переменные и константы в Java

В Java каждая переменная имеет тип. При объявлении переменной сначала указывается ее тип, а затем — ее имя:

double salary;

nt vacationDays;

long earthPopulation;

char yesChar;

boolean done;

**Правила образования имен переменных:**

1. Имя переменной должно начинаться с буквы и представлять собой комбинацию букв и цифр.
2. Все символы в имени переменной важны, причем регистр клавиатуры также имеет значение.
3. Длина имени переменной практически не ограничена.
4. В качестве имен переменных **нельзя** использовать зарезервированные слова.
5. В одной строке программы можно размещать несколько объявлений, например: int i,j; // Обе переменные — целочисленные.

**Присваивание и инициализация.**

После объявления переменной ее необходимо проинициализировать (задать ей значение) с помощью оператора присваивания («=»), поскольку использовать переменную, которой не присвоено никакого значения, невозможно – компилятор выдаст ошибку инициализации.

int vacationDays; // объявление.

vacationDays = 12; // оператор присваивания

**Общая формула объявления и инициализации переменной:**

1. Объявление переменной с последующей инициализацией

|  |
| --- |
| тип\_переменной имя\_переменной; *// объявление переменной*  имя\_переменной = значение\_переменной; *// инициализация* |

(2)Объявление и инициализация переменной

|  |
| --- |
| тип\_переменной имя\_переменной = значение\_переменной; *// объявление и инициализация переменной* |

Для создания константы в Java используется ключевое слово ***final***. Ключевое слово final означает, что присвоить значение такой переменной можно лишь один раз и навсегда. Использовать в именах констант только прописные буквы необязательно.

# Управляющие операторы. Условные операторы if-else, switch-case

В Java существует единственный условный оператор ветвления – оператор if. Его может дополнять оператор else. Ветвление, в котором участвуют оба оператора называется ***полным***, если только if – неполным (при этом else без if использоваться не может).

Условный оператор в языке Java имеет вид:

**if (условие) {**

**/\*действия\*/**

**}else{** *// не обязательный элемент*

**/\*действия\*/**

**}**

В качетсве условия может быть либо значение логической переменной (boolean), либо результат выполнения логического выражения.

boolean b;

… // в течение программы b задается определенное значение.

if(b)

{ System.out.println("b is true "); }

else

{ System.out.println("b is false "); }

Конструкция **if/else** может оказаться неудобной, если есть необходимость сделать выбор из многих вариантов, при **известных заранее значениях условий** выбора. В таких случаях используется оператор **switch**. Форма оператора **switch**.

**switch (переменная\_условия){**

**case значение\_1:**

**/\*дейсвтия\*/**

**[break;]**

**case значение\_2:**

**/\*дейсвтия\*/**

**[break;]**

**[default: /\*дейсвтия\*/]**

**}**

Выполнение начинается с метки **case**. Выбор **case** определяется, исходя из условия, что (**переменная\_условия == значение\_1)**, и продолжается до следующего оператора **break** или конца оператора **switch**. Если ни одна метка не совпадает со значением переменной, выполняется раздел **default**, если он предусмотрен. Заметим, что метка **case** должна быть **целочисленной**, строкой (**String**) или.

**Самостоятельно:**

1. Написать программу, которая будет выводить на экран следующие результат 2 операций (1 сложение, 2 вычитание): прибавить/отнять 1 к пограничным значениям byte. Объяснить результаты.
2. Каков результат выполнения следующего кода:

**boolean** b = fasle;

**if**(b == 0){System.out.print(“b is 0”);}

1. Каков результат выполнения следующего кода:

**byte** b = **17**;

**int** i = b;

b=i;

1. Объявите константу **const** типа **int** и выведите ее на консоль. Какой результат компиляции программы?

**Внимание: в заданиях 5 и 6 использовать только целые числа в качестве входных данных.**

1. Написать программу, которая будет выполнять следующий алгоритм:

1) ввести значения x, y, как аргументы метода main();

2) если x<0 и y<0, найти их модули и перейти к п. 5, иначе перейти к следующему пункту;

3) если x<0 или y<0, увеличить каждую величину на 0,5 и перейти к п. 5, иначе перейти к следующему пункту;

5) вывести значения x и y;

6) конец.

1. Написать программу, на вход которой поступает число, при этом:

1) Если число =1 или 2 или 3, знаковым сдвигом влево сдвинуть его на 1 его и вывести на экран результат операции.

2) Если число = 4, прибавить к нему такое значение, сдвинутое беззанаково на 1 вправо и вывести результат на экран.

3) Любое другое число: вывести на экран сообщение «Unknown command!»

**Внимание:** для успешного выполнения заданий 5 и 6 необходимо использовать функцию ***Integer.parseInt***(String value) – это библиотечная функция, которая преобразует введеную строку в число целого типа (int). Дело в том, что аргументы метода main() считываются из консоли, как строки (тип String), для того, чтобы работать с ними, как с числами необходимо преобразовать их с помощью указанной функции.