**Типы данных. Преобразование типов. Основные операции**

**Введение**

Основное предназначение компьютеров — это вычисления. Математические вычисления, изображение, звук, 3D-эффекты — это результат вычислений, который появляется благодаря размещению чисел в ячейках памяти компьютера и работе с ними. Задача программиста — написать программу, это значит заполнить память компьютера нужными данными, произвести с ними необходимые действия, получить результат и вывести его пользователю в требуемом виде. Соответственно, для того чтобы написать минимально полезную программу, нужно знать, как организовать ячейки памяти для хранения данных, заводить так называемые переменные и выполнять вычисления с ними, вводить и выводить значения переменных (в простейшем случае с консоли и на консоль).

Для того чтобы завести переменную, можно в любом месте функции вставить конструкцию:

<тип> <имя переменной>[ = <значение>];

Пример 1:

int a;

double b = 0.5;

В языке Java программисту изначально предоставлен большой набор функционала для разработки в виде стандартных классов, которые можно и нужно использовать, чтобы не тратить свое время и силы на написание простейших, но часто используемых функций, таких, например, как работа с консолью или математических вычислений.

Вывод текста или значений переменных на консоль выполняется при помощи стандартного класса System и его методов.

Пример 2:

int x = 5;

double y = 7.32;

System.out.println(x + “ ” + y);

System.out.format(Locale.US, "x = %d, y = %.2f", a + b);

Для ввода переменных удобно использовать класс Scanner, а точнее его объект.

Пример 3:

Scanner in = new Scanner(System.in);

int x = in.nextInt();

in.useLocale(Locale.US);

double r = in.nextDouble();

Одной из основных особенностей Java является то, что данный язык является строго типизированным. А это значит, что каждая переменная имеет определенный тип и данный тип строго определен специальными зарезервированными ключевыми словами Java. Тип данных определяет диапазон значений, которые может хранить переменная или константа.

Переменные не что иное, как зарезервированные места памяти для хранения значений. Поэтому при создании переменной Вы резервируете некоторое пространство в памяти, а ОС, основываясь на типе данных, который присвоен переменной, выделяет требуемое количество памяти и решает, что может быть сохранено в зарезервированную память.

Существует два типа данных в Java:

* простые или примитивные типы данных;
* ссылочные типы данных (ссылка/объект).

## **Примитивные типы данных**

Есть восемь типов данных, поддерживаемых Java. Все размеры примитивных типов строго фиксированы и не зависят от машинной архитектуры. Основные типы данных предопределены языком и названы по [ключевому слову](http://proglang.su/java/syntax" \l "klyuchevye-slova-v-java).

* byte (целые числа, 1 байт)
* short (целые числа, 2 байта)
* int (целые числа, 4 байта)
* long (целые числа, 8 байтов)
* float (вещественные числа, 4 байта)
* double (вещественные числа, 8 байтов)
* char (символ Unicode, 2 байта)
* boolean (значение истина/ложь, 1 байт)

И отдельно отметим ключевое слово void, которое означает отсутствие какого-либо типа. Это слово используется, когда нужно показать, что тип не важен или не задан.

Пример:

int func(){

return 0;

};

void func(){

};

Типы данных можно разбить на 4 группы:

|  |  |  |  |
| --- | --- | --- | --- |
| **Целые числа** | **Числа с плавающей точкой** | **Символы** | **Логические значения** |
| byte, short, int, long | float, double | char | boolean |

**Целые числа**

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип** | **Длина в байтах** | **Длина в битах** | **Диапазон** |
| byte | 1 | 8 | [-128, 127] или [-27, 27-1] |
| short | 2 | 16 | [-32768, 32767] или [-215, 215-1] |
| int | 4 | 32 | [-2147483648, 2147483647] или [-231, 231-1] |
| long | 8 | 64 | [-9223372036854775808, 9223372036854775807] или [-263, 263-1] |

Все эти типы представляют целочисленные значения со знаком: положительные или отрицательные.

### **Тип byte**

Наименьшим целочисленным типом является byte. Переменные этого типа очень удобны для работы с потоками ввода-вывода и при манипулировании двоичными данными.

Пример:

byte value = 2;

byte value = -2;

### **Тип short**

Тип short применяется реже всех остальных типов в ситуациях, когда действительно важна экономия памяти.

Пример:

short value = 3;

short value = -1000;

### **Тип int**

Самый часто используемый целочисленный тип. Причина в том, что при вычислении выражений, значения типа byte или short будут неявно преобразованы в int, поскольку все целочисленные значения Java по умолчанию воспринимает как типа int.

Пример:

byte a = 1;

byte b = 2;

byte result = a + b;

При компиляции этого кода, а мы получим ошибку:

«error: incompatible types: possible lossy conversion from int to byte».

Верная запись:

byte result = (byte)(a + b);

Также при задании целочисленных типов Java поддерживает разделение разрядов знаком «\_» .

Пример:

int value = 234\_567\_\_789;

### **Тип long**

Когда длины типа int недостаточно для хранения значения, нужно использовать long. Его диапазон значений достаточно велик. Часто этот тип используется при работе со временем или с большими расстояниями, например.

Так как все целочисленные значения по умолчанию расцениваются как значения типа int, то для обозначения того, что число это типа long после числа ставят литерал L или l.

long value = 2; //Ошибка

long value = 2l; //Верно

long value = 2L; //Верно и удобно

#### **Числа с плавающей точкой**

Числа с плавающей точкой (или действительные числа) представлены типами float и double. Используются для хранения значений с точностью до определенного знака после десятичной точки.

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип** | **Длина в байтах** | **Длина в битах** | **Диапазон** |
| float | 4 | 32 | [-1.4\*10-45, 3.4\*1038 ] |
| double | 8 | 64 | [-4.9\*10-324, 1.8\*10308 ] |

### **Тип float**

Тип float является 32-битным (как int), поэтому определяет числовое значение с плавающей точкой одинарной точности. Этот тип используется, когда нужно числовое значение с дробной частью, но без особой точности. Например, для сохранения памяти в больших массивах чисел с плавающей точкой.

При задании числа с плавающей точкой Java по умолчанию воспринимает его как double, поэтому чтобы указать, что данное значение должно рассматриваться как float, нам надо использовать суффикс f или F.

Пример:

float value = 30.6; //Неверно

float value = 30.6f; //Верно

float value = 30.6F;//Верно и удобно

### **Тип double**

Тип double используется для хранения значений с плавающей точкой двойной точности. Обработка значений двойной точности выполняется быстрее, чем обработка значений одинарной точности.

Пример:

double z = 1.0;

Кстати, float и double могут хранить не только число, но и исключительные значения. Такие, как:

* Infinity
* NAN

Пример 1:

double value = 12.0 / 0;

System.out.println(value); //Infinity

Пример 2:

double value = Math.sqrt(-4);

System.out.println(value); //NaN

**Логические значения.** **Тип boolean**

Примитивный тип boolean предназначен для хранения логических значений. Данный тип может принимать одно из двух возможных значений: true (истина) или false (ложь).

Значения boolean возвращаются во всех логических операциях (например, операции сравнения). Применяется при построении циклов.

Пример:

boolean value = true;

#### **Символьные значения. Тип char**

Символьный тип данных представляет собой один 16-битный Unicode символ. Unicode — это объединение десятков кодировок символов, он включает в себя латинский, греческий, арабский алфавиты, кириллицу и многие другие наборы символов.

Диапазон допустимых значений - от 0 (‘\ u0000’)  до 65536 (‘\ uffff’). Отрицательных значений для типа char не существует. Это единственный беззнаковый целочисленный тип данных в Java.

Символы char  можно задавать при помощи символа, заключенного в одинарные кавычки, а можно с помощью соответствующего кода-числа из таблицы Unicode-символов. Например символ ‘Ы’ соответствует числу  1067.

Пример 1:

char a=1067;

char b ='Ы';

System.out.println(a); //Ы

System.out.println((int)b); //1067

Пример 2:

int a = 3;

char b = '1';

char result = (char) (a+ b);

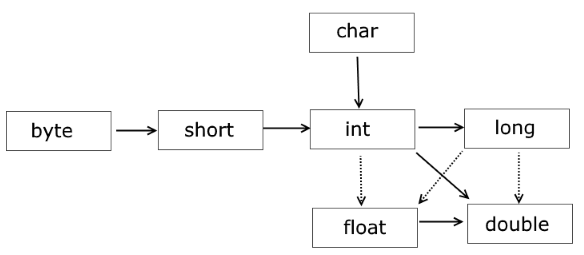
System.out.println(result); //4

**Преобразование типов**

Иногда возникают ситуации, когда у вас есть величина какого-то определенного типа, а вам нужно ее присвоить переменной другого типа. Для некоторых типов это можно проделать и без приведения типа, в таких случаях говорят об автоматическом (неявном) преобразовании типов. В других случаях программист должен сам провести преобразование типа к нужному — это называется явным преобразованием типов.

#### **Автоматические (неявные) преобразования**

При изучении типов данных уже говорилось о том, что целочисленные типы данных byte и short при вычислении Java всегда приводит к типу int, а тип float к типу double. Фактически это и есть неявное преобразование типов.



Стрелками на рисунке показано, какие преобразования типов могут выполняться автоматически без потери точности. Пунктирными стрелками показаны автоматические преобразования с потерей точности.

Преобразования от меньшего типа к большему называются расширяющими и именно эти преобразования проходят без потери данных и без потери точности.

Пример 1:

byte a = 40;

byte b = 50;

byte с = 100;

int d = a\* b / с;

Можно сказать, что существуют следующие правила повышения 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.

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.

Пример 2:

int a = 40;

float b = 50.0F;

double с = 100.0;

double d = a \* b / c;

Преобразования от большего типа к меньшему называются сужающими и в этих случаях вся ответственность по переносу данных ложится на программиста.

Пример 3:

int a = 2147483647;

float b = a; // от типа int к типу float

System.out.println(b);  // 2.14748365E9

#### **Явные преобразования**

Все остальные преобразования типов, не рассмотренные ранее, осуществляются явным образом, а именно указанием нужного типа, заключенного в круглые скобки. Обычно это сужающие преобразования от типа с большей разрядностью к типу с меньшей разрядностью. При явном преобразовании программист должен внимательно оценить все риски потери данных и выбрать тип, который сведет эти риски к нулю.

Пример 1:

long a = 4;

int b = (int) a;

Пример 2:

byte b = 50;

b = b\* 2;

Ошибка: Error:(5, 14) java: incompatible types: possible lossy conversion from int to byte

b = (byte) (b\* 2);

При сужающем преобразовании, если происходит, так называемое, переполнение, старшие биты просто отбрасываются.

Пример 2:

int a = 258;

byte b = (byte) a;

System.out.println(b); // 2

Число а: 00000000 00000000 00000001 00000010.

Значения типа byte занимают в памяти только 8 бит. Поэтому двоичное представление числа int усекается до 8 правых разрядов.

Число b: 00000010

При преобразовании значений с плавающей точкой к целочисленным значениям, происходит усечение дробной части.

Пример 4:

double a = 56.9898;

int b = (int)a;

System.out.println(b); // 56

Для того, чтобы именно округлить число с плавающей точкой, нужно воспользоваться функцией «round” класса Math из стандартной библиотеки Java.

double a = 56.9898;

int b = (int)Math.round(a);

Для выполнения различных математических операций в Java в пакете java.lang определен класс Math.

В классе определены две константы типа double: **E** и **PI**.

И большое многообразие математических функций:

|  |  |  |
| --- | --- | --- |
| **Тип** | **Метод** | **Описание** |
| double | abs(double a) | Возвращает абсолютное значение (модуль) числа типа double. |
| double | exp(double a) | Возвращает экспоненту аргумента. |
| double | max(double a, double b) | Возвращает больший из аргументов. |
| double | random() | Возвращает случайное число от 0.0 (включительно) до 1 (не включительно). |
| double | sin(double a) | Возвращает синус аргумента. |
| double | toRadians() | Преобразует градусы в радианы. |

### **Арифметические операции**

Большинство операций в Java являются стандартными математическими операциями. Есть унарные операции (выполняются над одним операндом), бинарные - над двумя операндами, а также тернарные - выполняются над тремя операндами. Операндом является переменная или значение (например, число), участвующее в операции.

**Унарные арифметические операции**

 Унарные арифметические операции — это операции, которые производятся над одним числом: ++ (инкремент) и -- (декремент). Инкремент подразумевает под собой увеличение значения переменной на единицу, декремент — наоборот, уменьшение значения переменной на единицу.

Каждая из операций имеет две разновидности: префиксная и постфиксная. Префиксный тип операции означает, что сначала значение переменной увеличивается на 1, а затем это значение переменной передается дальше. Постфиксный тип операции означает, что сначала значение переменной передается дальше, а затем увеличивается на 1.

Пример 1:

int a = 8;

int b = ++a;

System.out.println(a);// 9

System.out.println(b);// 9

a = 8;

b = a++;

System.out.println(a);// 9

System.out.println(b);// 8

a = 8;

b = --a;

System.out.println(a);// 7

System.out.println(b);// 7

a = 8;

b = a--;

System.out.println(a);// 7

System.out.println(b);// 8

**Бинарные арифметические операции**

К бинарным арифметическим операциям относятся:

* вычитание («-»)
* сложение («+»)
* умножение («\*»)
* деление («/»)
* остаток от деления («%»)

Пример 1:

int a = 10;

int b = 7;

int c = a + b;// 17

int d = 4 - a;// -6

int k = b \* 5;// 35

int i = a / b;// 1

double k = 10 / 4;// 2

double k = 10.0 / 4;// 2.5

int j = a % b;// 3

### **Приоритет арифметических операций**

Одни операции имеют больший приоритет чем другие и поэтому выполняются вначале. Операции в порядке уменьшения приоритета:

* ++ (инкремент), -- (декремент)
* \* (умножение), / (деление), % (остаток от деления)
* + (сложение), - (вычитание)

Приоритет операций следует учитывать при выполнении набора арифметических выражений. Чтобы задать свой порядок вычислений, нужно использовать ().

Пример 1:

int a = 8;

int b = 7;

int c = a + 5 \* ++b;

System.out.println(c);// 48

int d = (a + 5) \* ++b;

System.out.println(d);// 104

### **Операции с присваиванием**

Операцию присваивания можно скомбинировать с любой арифметической операцией и записать сокращенно, в виде рядом стоящих знаков присваивания и используемой арифметической операции.

Пример 1:

int c = 3;

c += 5; //c = c + 5 = 8

c -= 2; //c = c – 2 = 6

c /= 3; //c = c / 3 = 2;

возможные примеры для практики

class SimpleTypes {

public static void main(String args []) {

byte b = 0х55;

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 l = " + 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);

} }

public static void main(String []args){ float piValue = (float)Math.PI; double piValueExt = Math.PI; System.out.println("Float value: " + piValue ); System.out.println("Double value: " + piValueExt ); }