# Простые типы данных

В Java существует четыре группы базовых типов: для работы с целыми числами, для работы с числами в формате с плавающей точкой (действительные числа), символы и логический тип — таким образом, всего получается восемь базовых типов. Базовые типы Java перечислены в табл. 1.1.

Таблица 1.1. Базовые (простые) типы в Java

| Тип данных (название) | Количество битов | Пояснение | Класс-оболочка |
| --- | --- | --- | --- |
| byte | 8 | Целые числа в диапазоне от  -128 до 127 | Byte |
| short | 16 | Целые числа в диапазоне от  -32768 до 32767 | Short |
| int | 32 | Целые числа в диапазоне от  -2147483648 до 2147483647 | Integer |
| long | 64 | Целые числа в диапазоне от  -9223372036854775808 до 9223372036854775807 | Long |
| float | 32 | Действительные числа. По абсолютной величине изменяются в диапазоне от 3,4 × 10-38 до 3,4 × 1038 | Float |
| double | 64 | Действительные числа двойной точности. По абсолютной величине изменяются в диапазоне от 1,7 × 10–308 до 1,7 × 10308 | Double |
| char | 16 | Символьный тип для представления символьных значений (букв). Диапазон значений от 0 до 65536 (каждое значение соответствует определенному символу) | Character |
| boolean | 8[[1]](#footnote-0) | Логический тип данных. Переменная этого типа может принимать два значения: true (истина) и false (ложь) | Boolean |

В этой же таблице приведены названия классов-оболочек для базовых типов. Классы-оболочки используются в тех случаях, когда переменную соответствующего типа необходимо рассматривать как объект.

## Приведение типов данных

Язык программирования Java – язык строгой типизации переменных. При вычислении выражения, в которое входят переменные разных типов, автоматически выполняется преобразование входящих в выражение переменных к общему формату. Процесс автоматического преобразования типов подчиняется нескольким базовым правилам:

* Типы переменных, входящих в выражение, должны быть совместимыми.
* Целевой тип (тип, к которому выполняется приведение) должен быть «шире» исходного типа. Другими словами, преобразование должно выполняться без потери данных
* Перед выполнением арифметической операции типы byte, short и char расширяются до типа int. Если в выражении есть операнды типа long, то расширение осуществляется до типа long. Если в выражении есть операнды типа float, то расширение осуществляется до типа float. Если в выражении есть операнды типа double, то расширение осуществляется до типа double.

В Java предусмотрено явное приведение типов и явное определение типа литерала с помощью **суффиксов типа**. Для приведения выражения к нужному типу перед этим выражением указывается имя типа, заключенное в круглые скобки.

## Ввод и вывод данных

Рассмотрим организацию консольного ввода с помощью класса Scanner.

Для работы с этим классом необходимо включить в заголовок файла программы инструкцию import java.util.\*, то есть подключить (импортировать) пакет java.util.

Общая схема реализации процесса введения данных с консоли посредством класса Scanner такова: на основе стандартного потока ввода System.in создается объект класса Scanner, через который и осуществляется консольный ввод.

Scanner inp =**new** Scanner(System.***in***);

Следующие методы класса Scanner обеспечивают ввод данных:

nextLine() – считывание текстовой строки;

nextInt() – считывание целого числа;

nextDouble() – считывание действительного числа.

Пример:

**import** java.util.Scanner;  
  
**public class** MyClass {  
 **public static void** main(String[] args) {  
 System.***out***.println(**"Input int"**);  
 Scanner inp =**new** Scanner(System.***in***);  
 **int** x=inp.nextInt();  
 System.***out***.println(x);  
 }  
}

## Класс Math

Таблица 1.2. Некоторые методы класса Math

| **Оператор класса Math** | **Примечание** |
| --- | --- |
| **Тригонометрические и обратные тригонометрические функции** | |
| sin(x) | sin(x) – синус |
| cos(x) | cos(x) – косинус |
| tan(x) | tg(x) – тангенс |
| asin(x) | arcsin(x) – арксинус |
| *acos*(x) | arccos(x) – арккосинус |
| atan(x) | arctg(x) – арктангенс |
| atan2(y, x) | Возвращает угол, соответствующий точке с координатами x,y, лежащий в пределах (-\pi,\pi]. |
| toRadians(angdeg) | angdeg / 180.0 \* PI; - перевод углов из градусов в радианы. |
| toDegrees(angrad) | angrad \* 180.0 / PI; - перевод углов из радиан в градусы. |
| **Степени,*****экспоненты*, логарифмы** | |
| exp(x) | e^x - *экспонента* |
| expm1(x) | e^x-1. При x, близком к 0, дает гораздо более точные значения, чем exp(x)-1 |
| log(x) | ln(x) – *натуральный логарифм*. |
| log10(x) | \log_{10}(x) – десятичный логарифм. |
| log1p(x) | \ln(1+x). При x, близком к 0, дает гораздо более точные значения, чем \log(1+x) |
| sqrt(x) | \sqrt{} - квадратный корень |
| cbrt(x) | \sqrt[3]{} - кубический корень |
| hypot(x,y) | \sqrt{x^2+y^2} - вычисление длины гипотенузы по двум катетам |
| pow(x, y) | x^y – возведение x в степень y |
| sinh(x) | \sh(x)=\frac{e^x-e^{-x}}{2} – гиперболический синус |
| cosh(x) | \ch(x)=\frac{e^x+e^{-x}}{2} – гиперболический косинус |
| tanh(x) | \th(x)=\frac{e^x-e^{-x}}{e^x+e^{-x}} – гиперболический тангенс |
| **Модуль, знак, минимальное, максимальное число** | |
| abs(m)  abs(x) | Абсолютное значение числа. Аргумент типа int, long, float или double. Результат того же типа, что аргумент. |
| signum(a)  signum(x) | Знак числа. Аргумент типа float или double. Результат того же типа, что аргумент. |
| min(m,n)  min(x,y) | Минимальное из двух чисел. Аргументы одного типа. Возможны типы: int, long, float, double. Результат того же типа, что аргумент. |
| max(m,n)  max(x,y) | Максимальное из двух чисел. Аргументы одного типа. Возможны типы: int, long, float, double. Результат того же типа, что аргумент. |
| **Округления** | |
| *ceil*(x) | Ближайшее к **x** целое, большее или равное **x** |
| *floor*(x) | Ближайшее к **x** целое, меньшее или равное **x** |
| round(a)  round(x) | Ближайшее к **x** целое. Аргумент типа float или double. Результат типа long, если аргументdouble, и типа int – если float. То же, что (int)*floor*(x + 0.5). |
| rint(x) | Ближайшее к **x** целое. |
| *ulp*(a)  *ulp*(x) | Расстояние до ближайшего большего чем аргумент значения того же типа ("дискретность" изменения чисел в формате с плавающей точкой вблизи данного значения). Аргумент типа float или double. Результат того же типа, что аргумент. |
| **Случайное число, остаток** | |
| random() | *Псевдослучайное число* в диапазоне от 0.0 до 1.0. |
| IEEEremainder(x,y) | Остаток от целочисленного деления **x/y**, то есть **x-y\*n**, где **n** – результат целочисленного деления |

В приведенной таблице величины имеют следующие типы:

x, y, angdeg, angrad – тип double,

a– тип float,

m, n– целые типов long или int.

Математические функции возвращают значения типа double, если в примечании не указано иное.

## Основные операторы для работы с целочисленными величинами

В приведенной ниже таблице (табл. 1.3) i и j обозначают целочисленные выражения, а v – целочисленную переменную.

Таблица 1.3. Базовые целочисленные операции в Java

| **Оператор** | **Название** | **Пример** | **Примечание** |
| --- | --- | --- | --- |
| **+** | Оператор сложения | i+j | В случае, когда операнды i и j имеют разные типы или типы byte, shortили char, действуют правила *автоматического преобразования* типов. |
| **–** | Оператор вычитания | i-j |
| **\*** | Оператор умножения | i\*j |
| **/** | Оператор деления | i/j | Результат округляется до целого путем отбрасывания дробной части как для положительных, так и для отрицательных чисел. |
| **%** | Оператор остатка от целочисленного деления | i%j | Возвращается остаток от целочисленного деления |
| **=** | Оператор присваивания | v=i | Сначала вычисляется выражение i, после чего полученный результат копируется в ячейку v |
| **++** | Оператор *инкремента* (увеличения на 1) | v++ | v++ эквивалентно v=v+1 |
| **--** | Оператор *декремента* (уменьшения на 1) | v-- | v-- эквивалентно v=v-1 |
| **+=** |  | v+=i | v+=i эквивалентно v=v+i |
| **-=** |  | v-=i | v-=i эквивалентно v=v-i |
| **\*=** |  | v\*=i | v\*=i эквивалентно v=v\*i |
| **/=** |  | v/=i | v/=i эквивалентно v=v/i |
| **%=** |  | v%=i | v%=i эквивалентно v=v%i |

Отдельно выделим важные методы классов Integer и Long:

Функции преобразования строкового значения в числовой формат:

Integer.parseInt( <строка> )

Long.parseLong( <строка> )

Функции Integer.signum (<число>) и Long.signum (<число>) возвращают знак числа, то есть

1, если число положительное,

0, если оно равно 0

-1, если число отрицательное.

**Условный оператор**

Неполный условный оператор:

if(<условие>)

<оператор1>;

Полный условный оператор:

if(<условие>)

<оператор1>;

else

<оператор2>;

**Тернарный оператор if-then-else**

Общая форма оператора if-then-else такова:

<условие> ? <значение1> : <значение2>;

В качестве первого операнда – <условие> – может быть использовано любое выражение, результатом которого является значение типа boolean. Если результат равен true, то тернарный условный оператор возвращает <значение1>, иначе <значение2>. При этом <значение1> и <значение2>, должны возвращать значения одного типа и не должны иметь тип void.

**Оператор выбора switch**

Синтаксис оператора:

switch(<выражение>){

case <значение1>: <последовательность операторов 1;> break;

……………………………

case <значениеN>:<последовательность операторов N;> break;

default:< последовательность операторов;> break;

}

Отметим, что нельзя указывать *диапазон* значений, нельзя перечислять через запятую значения, которым соответствуют одинаковые *операторы*.

**Оператор цикла for**

for(<блок инициализации>; <условие выполнения тела цикла>; <блок изменения счетчиков>)<оператор>;

**Оператор цикла while()**

Синтаксис:

while(<условие>){

// <тело цикла>

}

**Оператор цикла do-while()**

Синтаксис:

do{

// <тело цикла>

}

while(<условие>);

# Объявление и инициализация массивов

Синтаксис объявления массивов в Java совпадает с синтаксисом объявления массивов в языке С# и имеет вид:

//объявление одномерного массива

типЭлементов[] *имя*=**new** типЭлементов[*размер*];

типЭлементов *имя*[]=**new** типЭлементов[*размер*];

При этом допускается указывать квадратные скобки либо после имени типа массива, либо после имени массива. Например, вместо команды int[] nums можно использовать команду int nums[].

В Java многомерные массивы представляют собой массивы массивов. При объявлении переменной многомерного массива для указания каждого дополнительного индекса используют отдельный набор квадратных скобок. //объявление двумерного массива

типЭлементов[][] *имя*=**new** типЭлементов[*размер\_1*][*размер\_2*];

Обращение к элементу одномерного массива осуществляется через имя массива с указанием в квадратных скобках индекса элемента. Индексация элементов массива начинается с нуля.

При объявлении переменной массива, ее можно инициализировать списком значений, указывав (через оператор присваивания) список значений в фигурных скобках. Например: int[] data={3,8,1,7};

## Перебор значений массива

1 способ:

тип[] имя массива = new <тип>[];

for (int i = 0; i < имя.length; i++) {

тело цикла

}

2 способ:

for (тип element : имя массива) {

тело цикла

}

Отметим, что во втором случае мы не имеем доступа к индексу массива.

## Класс java.util.Arrays

Класс Arrays предназначен для работы с массивами. Рассмотрим некоторые методы класса Arrays

| **Название метода** | **Описание** |
| --- | --- |
| copyOf | предназначен для копирования массива; |
| copyOfRange | копирует часть массива; |
| toString | позволяет получить все элементы в виде одной строки; |
| sort | сортирует массив методом quick sort; |
| binarySearch | ищет элемент методом бинарного поиска; |
| fill | заполняет массив некоторым значением |
| equals | проверяет массивы на идентичность. Массивы считаются равными, и возвращается true, если они имеют одинаковую длину и равны элементы массивов с одинаковыми индексами |
| deepEquals | проверяет на идентичность массивы массивов; |
| asList | возвращает массив как коллекцию |

**Строки**

Для работы с текстом служат два встроенных Java-класса: String и StringBuffer. Объекты класса String изменять нельзя, а объекты класса StringBuffer – можно. В некоторых случаях можно использовать массив символов char [].

Классы String и StringBuffer определены в базовом пакете java.lang, который доступен по умолчанию, поэтому для создания объекта класса String или StringBuffer импорт пакетов выполнять не нужно. Оба класса определены как неизменяемые (final), то есть они не могут быть суперклассами для наследования.

**Объекты класса String**

**Конструкторы класса String:**

* Конструктор создания пустой строки - конструктор без параметров. Пример команды создания объекта класса String со значением в виде пустой строки:

String s=new String();

* Конструктор создания текстовой строки на основе символьного массива. В этом случае аргументом конструктору передается имя массива символов. Результатом является текст, составленный из всех символов массива в порядке их размещения в массиве. Пример создания текстовой строки на основе символьного массива:

char symbols[]={'a','b','c'};

String s=new String(symbols);

В этом конструкторе, помимо имени массива, можно указать индекс элемента массива, начиная с которого будет извлекаться строка, а также длину строки в символах. Например, так:

char symbols={'a','b','c','d','e','f'};

String s=new String(symbols,2,3); // s="cde"

* Конструктор копирования объекта. Параметром конструктора является переменная текстового типа, ссылающаяся на уже существующий текстовый объект или текстовый литерал. В результате создается новый объект с таким же текстовым значением, как и исходный. Например:

String obj=new String("Текстовая строка");

String s=new String(obj);

Некоторые методы класса String

| **Название метода** | **Описание** |
| --- | --- |
| length()  compareTo(String anotherString)  compareToIgnoreCase(String str)  regionMatches(boolean ignoreCase, int ind1, String other, int ind2, int len)  regionMatches(int ind1, String other, int ind2, int len)  concat(String str)  contains(CharSequence s)  endsWith(String suffix)  startsWith(String prefix)  startsWith(String prefix, int toffset)  equals(Object anObject)  equalsIgnoreCase()  getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)  indexOf(int ch)  indexOf(int ch, int fromIndex)  indexOf(String str)  indexOf(String str, int fromIndex)  lastIndexOf(int ch)  lastIndexOf(int ch, int fromIndex)  lastIndexOf(String str)  lastIndexOf(String str, int fromIndex)  replace(char oldChar, char newChar)  replace(CharSequence target, CharSequence replacement)  substring(int beginIndex, int endIndex)  toLowerCase()  toUpperCase()  trim()  valueOf(a) | возвращает длину строки  лексиграфическое сравнение строк; лексиграфическое сравнение строк без учета регистра символов;  тест на идентичность участков строк, можно указать учет регистра символов;  Здесь ind1 – индекс начала подстроки данной строки, ind2 – индекс начала подстроки другой строки str. Результат false получается в следующих случаях:   * хотя бы один из индексов ind1 или ind2 отрицателен; * хотя бы одно из ind1 + len или ind2 + len больше длины соответствующей строки; * хотя бы одна пара символов не совпадает.   тест на идентичность участков строк;  возвращает соединение двух строк;  проверяет, входит ли указанная последовательность символов в строку;  проверяет завершается ли строка указанным суффиксом;  проверяет, начинается ли строка с указанного префикса;  проверяет, начинается ли строка в указанной позиции с указанного префикса;  проверяет идентична ли строка указанному объекту;  возвращает символьное представление участка строки;  поиск первого вхождения символа в строке;  поиск первого вхождения символа в строке с указанной позиции;  поиск первого вхождения указанной подстроки;  поиск первого вхождения указанной подстроки с указанной позиции;  поиск последнего вхождения символа;  поиск последнего вхождения символа с указанной позиции;  поиск последнего вхождения строки;  поиск последнего вхождения строки с указанной позиции;  замена в строке одного символа на другой;  замена одной подстроки другой;  возвращает заданную подстроку исходной строки;  преобразовать строку в нижний регистр;  преобразовать строку в верхний регистр;  удаляет начальные и конечные пробелы;  статические методы преобразования различных типов в строку. |

Класс StringBuffer

У класса StringBuffer несколько конструкторов:

* конструктор без аргументов StringBuffer()

При использовании конструктора без аргумента создается объект класса StringBuffer со значением в виде пустой текстовой строки, а также автоматически резервируется память еще для 16-ти символов (буфер памяти).

* конструктор с числовым аргументом

Чтобы в явном виде указать размер буфера памяти при создании объекта класса StringBuffer, используют конструктор с числовым аргументом.

* конструктор с текстовым аргументом (типа String или StringBuffer).

Для создания копии уже существующего текстового объекта применяют конструктор с текстовым аргументом.

Некоторые методы для работы с объектами класса StringBuffer

| **Название метода** | **Описание** |
| --- | --- |
| length()  capacity()  ensureCapacity()  setLength()  charAt()  setCharAt()  getChars()  append()  insert()  reverse()  delete()  deleteCharAt()  replace() | возвращает текущую длину текстовой строки  возвращает выделенный для данной текстовой переменной объем памяти (в символах, то есть количество символов, которые можно записать в текстовую строку)  выделяет память для уже созданного объекта. Размер выделяемой памяти указывается аргументом метода  устанавливает длину текстовой строки  возвращает символ в строке с указанным индексом  У метода два аргумента: индекс символа в строке и символьное значение. Символ строки с заданным первым аргументом индексом заменяется символом, указанным вторым аргументом метода. Изменяется исходная строка  Копирование строки в символьный массив. Аргументы метода:  начальный индекс подстроки и индекс первого не входящего в подстроку символа, массив, в который выполняется копирование, а также индекс элемента в этом массиве, начиная с  которого в массив производится посимвольное копирование  подстроки  добавляет заданный текст в конец исходной строки  выполняет вставку текста, указанного вторым аргументом метода в строку вызова. Первым аргументом метода указывается индекс начала вставки подстроки  меняет порядок следования символов в строке вызова.  удаляет подстроку из строки вызова. Первым аргументом метода указывается индекс начала удаляемой подстроки, вторым – индекс первого после удаляемой подстроки символа  удаляет символ с заданным индексом из строки вызова  удаляет подстроку из строки вызова и на ее место вставляет другой текст. Первым аргументом метода указывается индекс начала удаляемой подстроки, вторым – индекс первого после удаляемой подстроки символа. Третий аргумент метода – текст, вставляемый вместо удаленной подстроки. |

Форматирование строк

В центре системы поддержки форматированного вывода находится класс Formatter. Он предлагает преобразования формата, позволяющие отображать числа, строки, время и даты практически в любом виде по вашему желанию.

Для форматирования вывода нужно создать объект Formatter.

Методы класса Formatter

| **Название метода** | **Описание** |
| --- | --- |
| format(String fтtString, Object ... args) | Форматирует apгументы, переданные в args, в соответствие со спецификаторами формата, содержащимися в ftString. Возвращает вызываемый объект. |

**Спецификаторы формата:**

**%a –** Шестнадцатеричное значение с плавающей точкой

**%b –** Логическое (булево) значение аргумента

**%c –** Символьное представление аргумента

**%d –** Десятичное целое значение аргумента

**%h –** Хэш-код аргумента

**%e –** Экспоненциальное представление аргумента

**%f –** Десятичное значение с плавающей точкой

**%g –** Выбирает более короткое представление из двух: **%е** или**%f**

**%o –** Восьмеричное целое значение аргумента

**%n –** Вставка символа новой строки (перевод строки)

**%s –** Строковое представление аргумента

**%t –** Время и дата

**%x –** Шестнадцатеричное целое значение аргумента

**%% –** Вставка знака **%**

Пример:

Formatter fmt = new Formatter();

fmt.fоrmаt("Форматировать %s очень просто: %d %f", "с помощью Java", 10, 98.6);

Приведенная кодовая последовательность создает объект Formatter, содержащий следующую строку:

*Форматировать с помощью Java очень просто: 10 98.600000*

В этом примере спецификаторы формата %s, %d и %f замещаются аргументами, следующими за строкой формата. То есть %s заменяется на "с помощью Java", %d на 10, а %f на 98.6. Все остальные символы используются, как есть. Спецификатор параметра %s указывает строку, %d целое число, а %f, означает число с плавающей точкой.

**Указание минимальной ширины поля**

Целое число, помещенное между символом % и кодом преобразования формата, выступает в качестве спецификатора минимальной ширины. Он дополняет вывод пробелами, чтобы обеспечивать заданную минимальную длину. Если строка или число получаются длиннее, чем этот заданный минимум, они будут напечатаны полностью. По умолчанию дополнение осуществляется пробелами. Если вы хотите дополнять нулями, поместите 0 перед спецификатором ширины поля. Например, %05d дополнит число, состоящее из менее чем 5 разрядов, нулями таким образом, чтобы его общая ширина была равна пяти. Спецификатор ширины поля может применяться вместе со всеми спецификаторами формата, кроме %n.

Пример:

Formatter fmt new Formatter();

fmt.format("|%fl%nl%12fl%nl%012f|", 10.12345, 10.12345, 10.12345);

System.out.println(fmt);

Эта программа генерирует следующий вывод:

|10.123450|

| 10.123450|

|00010.123450|

Первая строка отображает число 10,12345 с шириной по умолчанию. Вторая отображает это значение в 12-ти символьном поле. Третья строка отображает значение в 12-ти символьном поле, дополняя его ведущими нулями.

Минимальный модификатор ширины поля часто используется для генерации таблиц, состоящих из строк и столбцов.

**Указание точности**

Спецификатор точности может быть применим к спецификаторам формата %f, %е, %g и %s. Он следует за спецификатором минимальной ширины поля (если таковой имеется) и состоит из точки с последующим целым числом. Его конкретное значение зависит от типа данных, к которому он применяется.

Когда вы применяете спецификатор точности к данным с плавающей точкой с применением спецификаторов преобразования %f или %е, то он определяет количество отображаемых десятичных разрядов. Например, %10.4 f отображает число, по меньшей мере, в 10 символов шириной с четырьмя разрядами после запятой. При использовании %g точность определяет количество значащих десятичных разрядов. Точность по умолчанию составляет 6 знаков после запятой.

В применении к строкам спецификатор точности задает максимальную ширину поля. Например, %5. 7s отображает строку длиной минимум в пять символов и не превышающую семь символов. Если строка длиннее максимальной ширины, конечные символы будут усечены.

**Использование индекса аргументов**

Formatter включает очень удобное средство, позволяющее указать аргумент, к которому должен применяться конкретный спецификатор формата. Обычно порядок аргументов и спецификаторов формата совпадает слева направо. То есть первый спецификатор формата относится к первому аргументу, второй – ко второму и т.д. Однако, используя индекс аргумента, вы можете явно управлять тем, к какому из аргументов относится спецификатор формата.

Индекс аргумента следует сразу за % в спецификаторе формата. Он имеет следующий вид:

n$

Здесь n – индекс нужного аргумента, начиная с 1. Рассмотрим следующий пример.

fmt.format("%3$d%1$d%2$d", 10, 20, 30);

Этот код порождает строку: 30 10 20

В этом примере первый спецификатор формата соответствует 30, второй – 10, а третий – 20.

Одной из выгод индексирования аргументов является то, что оно позволяет повторно использовать аргумент, не указывая его дважды. Например, рассмотрим следующую строку:

fmt.format("%d в шестнадцатеричном формате равно %l$х", 255);

Она порождает следующий результат: *255 в шестнадцатеричном формате равно ff* .

Как видите, аргумент 255 используется с обоими спецификаторами формата.

1. Для хранения значения этого типа достаточно 1 бита, но в реальности память такими порциями не выделяется, поэтому переменные этого типа могут быть по-разному упакованы виртуальной машиной [↑](#footnote-ref-0)