Содержание

2 Специализация: данные и функции 1
2.1 Данные . . . 1
2.2 Примитивные типы данных . . . 3
2.3 Ссылочные типы данных, массивы . . . 14
2.4 Базовый функционал языка . . . 16
2.5 Функции . . . 19

2. Специализация: данные и функции

В предыдущем разделе

— Краткая история (причины возникновения);
— инструментарий, выбор версии;
— CLI;
— структура проекта;
— документирование;
— некоторые интересные способы сборки проектов.

В этом разделе

Будет рассмотрен базовый функционал языка, то есть основная встроенная функциональность, такая как математические операторы, условия, циклы, бинарные операторы. Далее способы хранения и представления данных в Java, и в конце способы манипуляции данными, то есть функции (в терминах языка называющиеся методами).

— Метод;
— Типизация;
— Переполнение;
— Инициализация;
— Идентификатор;
— Typecasting;
— Массив;

2.1. Данные

2.1.1. Понятие типов

Хранение данных в Java осуществляется привычным для программиста образом: в переменных и константах.

Относительно типизации языки программирования бывают типизированными и нетипизированными (бестиповыми). Нетипизированные языки не представляют большого интереса в современном программировании.

Отсутствие типизации в основном присуще чрезвычайно старым и низкоуровневым языкам программирования, например, Forth и некоторым ассемблерам. Все данные в таких языках считаются цепочками бит произвольной длины и не делятся на типы. Работа с ними часто труднее, при этом часто безтиповые языки работают быстрее типизированных, но описывать с их помощью большие проекты со сложными взаимосвязями довольно утомительно.

Java является языком со **строгой** (также можно встретить термин «**сильной**») **явной статической** типизацией.

- Статическая - у каждой переменной должен быть тип, и этот тип изменить нельзя. Этому свойству противопоставляется динамическая типизация;
- Явная - при создании переменной ей обязательно необходимо присвоить какой-то тип, явно написав это в коде. В более поздних версиях языка (с 9й) стало возможным инициализировать переменные типа `var`, обозначающий нужный тип тогда, когда его возможно однозначно вывести из значения справа. Бывают языки с неявной типизацией, например, Python;
- Строгая(сильная) - невозможно смешивать разнотипные данные. С другой стороны, существует JavaScript, в котором запись `2 + true` выдаст результат 3.

2.1.2. Антипаттерн «магические числа»

Почти во всех примерах, которые используются для обучения, можно увидеть так называемый антипаттерн - плохой стиль для написания кода. Числа, которые находятся справа от оператора присваивания используются в коде без пояснений. Такой антипаттерн называется «магическое число». Магическое, потому что непонятно, что это за число, почему это число именно такое и что будет, если это число изменить.

Так лучше не делать. Заранее нужно сказать, что рекомендуется помещать все числа в коде в именованные константы, которые хранятся в начале файла. Плюсом такого подхода является возможность легко корректировать значения переменных в достаточно больших проектах.

Например, в вашем коде несколько тысяч строк, а какое-то число, скажем, возраст совершеннолетия, число 18, использовалось несколько десятков раз. При использовании приложения в стране, где совершеннолетием считается 21 год вы должны будете перечитывать весь код в поисках магических «18» и исправить их на «21». В этом вопросе будет также важно не запутаться, действительно ли это 18, которые означают совершеннолетие, а не количество карманов в жилетке Анатолия Вассермана[1].

В случае с константой изменить число нужно в одном месте.

[1] мы то знаем, что их 26

Тип	Пояснение	Диапазон
byte	Самый маленький из адресуемых типов, 8 бит, знаковый	[−128, +127]
short	Тип короткого целого числа, 16 бит, знаковый	[−32 768, +32 767]
char	Целочисленный тип для хранения символов в кодировке UTF-8, 16 бит, беззнаковый	[0, +65 535]
int	Основной тип целого числа, 32 бита, знаковый	[−2 147 483 648, +2 147 483 647]
long	Тип длинного целого числа, 64 бита, знаковый	[−9 223 372 036 854 775 808, +9 223 372 036 854 775 807]
float	Тип вещественного числа с плавающей запятой (одинарной точности, 32 бита)	
double	Тип вещественного числа с плавающей запятой (двойной точности, 64 бита)	
boolean	Логический тип данных	true, false

Таблица 1: Основные типы данных в языке Java

2.2. Примитивные типы данных

Все данные в Java делятся на две основные категории: примитивные и ссылочные. Таблица 1 демонстрирует все восемь примитивных типов языка и их размерности. Чтобы отправить на хранение какие-то данные используется оператор присваивания. Присваивание в программировании - это не тоже самое, что математическое равенство, демонстрирующее тождественность, а полноценная операция.

Все присваивания всегда происходят справа налево, то есть сначала вычисляется правая часть, а потом результат вычислений присваивается левой. Исключений нет, именно поэтому в левой части не может быть никаких вычислений.

Шесть из восьми типов имеет диапазон значений, а значит основное их отличие в объёме занимаемой памяти. У `double` и `float` тоже есть диапазоны, но они заключаются в точности представления дробной части. Диапазоны означают, что если попытаться положить в переменную меньшего типа большее значение, произойдёт «переполнение переменной».

2.2.1. Переполнение целочисленных переменных

Чем именно чревато переполнение переменной легче показать на примере (по ссылке - расследование крушения ракеты из-за переполнения переменной)

Переполнение переменных не распознаётся компилятором.

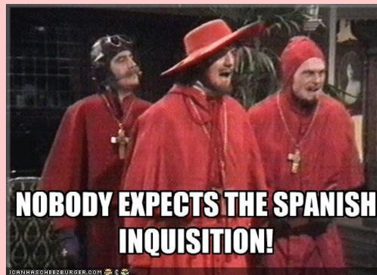
Если создать переменную типа byte, диапазон которого от $[-128, +127]$, и присвоить

этой переменной значение 200 произойдёт переполнение, как если попытаться влить пакет молока в напёрсток.

Переполнение переменной - это ситуация, в которой происходит попытка положить большее значение в переменную меньшего типа.

Важным вопросом при переполнении остаётся следующий: какое в переполненной переменной останется значение? Максимальное, 127? $200 - 127 = 73$? Какой-то мусор? Каждый язык, а зачастую и разные компиляторы одного языка ведут себя в этом вопросе по разному.

В современном мире гигагерцев и терабайтов почти никто не пользуется маленькими типами, но именно из-за этого ошибки переполнения переменных становятся опаснее испанской инквизиции.



2.2.2. Задание для самопроверки

1. Возможно ли объявить в Java целочисленную переменную и присвоить ей дробное значение?
2. Магическое число - это:
   (a) числовая константа без пояснений;
   (b) число, помогающее в вычислениях;
   (c) числовая константа, присваиваемая при объявлении переменной.
3. Переполнение переменной - это:
   (a) слишком длинное название переменной;
   (b) слишком большое значение переменной;
   (c) расширение переменной вследствие записи большого значения.

2.2.3. Бинарное (битовое) представление данных

После разговора о переполнении, нельзя не сказать о том, что именно переполняется. Далее будут представлены сведения которые касаются не только языка Java но и любого другого языка программирования. Эти сведения помогут разобраться в

деталях того как хранится значение переменной в программе и как, в целом, происходит работа компьютерной техники.

Все современные компьютеры, так или иначе работают от электричества и являются примитивными по своей сути устройствами, которые понимают только два состояния: есть напряжение в электрической цепи или нет. Эти два состояния принято записывать в виде 1 и 0, соответственно.

Все данные в любой программе - это единицы и нули. Данные в программе на Java не исключение, удобнее всего это явление рассматривать на примере примитивных данных. Поскольку в компьютере можно оперировать только двумя значениями то естественным образом используется двоичная система счисления.

Десятичное	Двоичное	Восьмеричное	Шестнадцатеричное
00	00000	00	0x00
01	00001	01	0x01
02	00010	02	0x02
03	00011	03	0x03
04	00100	04	0x04
05	00101	05	0x05
06	00110	06	0x06
07	00111	07	0x07
08	01000	10	0x08
09	01001	11	0x09
10	01010	12	0x0a
11	01011	13	0x0b
12	01100	14	0x0c
13	01101	15	0x0d
14	01110	16	0x0e
15	01111	17	0x0f
16	10000	20	0x10

Таблица 2: Представления чисел

Двоичная система счисления это система счисления с основанием два. Существуют и другие системы счисления, например, восьмеричная, но сейчас она отходит на второй план полностью уступая своё место шестнадцатеричной системе счисления. Каждая цифра в десятичной записи числа называется разрядом, аналогично в двоичной записи чисел каждая цифра тоже называется разрядом, но для компьютерной техники этот разряд называется битом.

Одна единица или ноль - это один **бит** передаваемой или хранимой информации.

Биты принято собирать в группы по восемь штук, по восемь разрядов, эти группы называются **байт**. В языке Java возможно оперировать минимальной единицей информации, такой как байт для этого есть соответствующий тип. Диапазон байта, согласно таблицы $[-128, +127]$, то есть байт информации может в себе содержать ровно 256 значений. Само число 127 в двоичной записи это семиразрядное число, все разряды которого единицы (то есть байт выглядит как `01111111`). Последний, восьмой, самый старший бит, определяет знак числа[2]. Достаточно знать формулу расчёта записи отрицательных значений:

1. в прямой записи поменять все нули на единицы и единицы на нули;
2. поставить старший бит в единицу.

Так возможно получить на единицу меньшее отрицательное число, то есть преобразовав 0 получим -1, 1 будет -2, 2 станет -3 и так далее.

Числа бóльших разрядностей могут хранить бóльшие значения, теперь преобразование диапазонов из десятичной системы счисления в двоичную покажет что `byte` это один байт, `short` это два байта, то есть 16 бит, `int` это 4 байта то есть 32 бита, а `long` это 8 байт или 64 бита хранения информации.

2.2.4. Задания для самопроверки

1. Возможно ли число 3000000000 (3 миллиарда) записать в двоичном представлении?
2. Как вы думаете, почему шестнадцатеричная система счисления вытеснила восьмеричную?

2.2.5. Целочисленные типы

Целочисленных типов четыре, и они занимают 1, 2, 4 и 8 байт.

Технически, целочисленых типов пять, но `char` устроен чуть сложнее других, поэтому не рассматривается в этом разделе.

Значения в целочисленных типах могут быть только целые, никак и никогда невозможно присвоить им дробных значений. Про эти типы следует помнить следующее:

— `int` - это самый часто используемый тип. Если сомневаетесь, какой целочисленный тип использовать, используйте `int`;
— все целые числа, которые пишутся в коде - это `int`, даже если вы пытаетесь их присвоить переменной другого типа.

Как `int` преобразуется в меньше типы? Если написать цифрами справа число, которое может поместиться в переменную меньшего типа слева, то статический анализатор кода его пропустит, а компилятор преобразует в меньший тип автоматически (строка 9 на рис. 1).

[2]Здесь можно начать долгий и скучный разговор о схемотехнике и хранении отрицательных чисел с применением техники дополнительного кода.

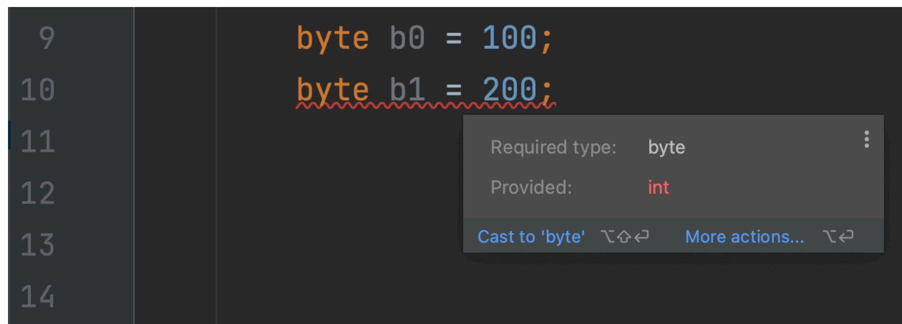


Рис. 1: Присваивание валидных и переполняющих значений

Как видно, к маленькому `byte` успешно присваивается `int`. Если же написать число которое больше типа слева и, соответственно, поместиться не может, среда разработки выдает предупреждение компилятора, что ожидался `byte`, а передан `int` (строка 10 рис 1).

Часто нужно записать в виде числа какое-то значение большее чем может принимать `int`, и явно присвоить начальное значение переменной типа `long`.

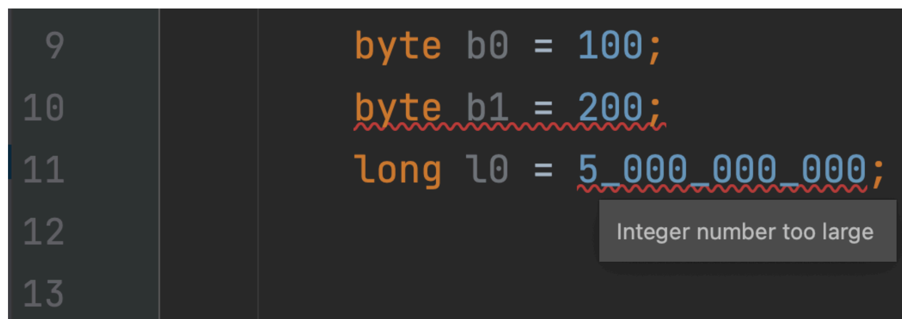


Рис. 2: Попытка инициализации переменной типа `long`

В примере на рис. 2 показана попытка присвоить значение 5000000000 переменной типа `long`. Из текста ошибки ясно, что невозможно положить такое большое значение в переменную типа `int`, а это значит, что справа `int`. Почему большой `int` без проблем присваивается к маленькому байту?

```
 9      byte b0 = 100;
10      byte b1 = 200;
11      long l0 = 5_000_000_000;
12      long l1 = 5_000_000_000L;
13      float f0 = 0.123;
14      float f1 = 0.123f;
```

Рис. 3: Решение проблемы переполнения числовых констант

На рис. 3 продемонстрировано, что аналогичная ситуация возникает с типами `float` и `double`. Все дробные числа, написанные в коде - это `double`, поэтому положить их во `float` без дополнительных усилий невозможно. В этих случаях к написанному справа числу нужно добавить явное указание на его тип. Для `long` пишем `L`, а для `float` - `f`. Чаще всего `L` пишут заглавную, чтобы подчеркнуть, что тип больше, а `f` пишут маленькую, чтобы подчеркнуть, что мы уменьшаем тип. Но регистр в этом конкретном случае значения не имеет, можно писать и так и так.

2.2.6. Числа с плавающей запятой (точкой)

Как видно из таблицы 1, два из восьми типов не имеют диапазонов значений. Это связано с тем, что диапазоны значений флоута и дабла заключаются не в величине возможных хранимых чисел, а в точности этих чисел после запятой.

> Числа с плавающей запятой в англоязычной литературе называются числа с плавающей точкой (от англ. floating point). Такое различие связано с тем, что в русскоязычной литературе принято отделять дробную часть числа запятой, а в европейской и американской - точкой.

Хранение чисел с плавающей запятой[3] работает по стандарту IEEE 754 (1985 г). Для работы с числами с плавающей запятой на аппаратурном уровне к обычному процессору добавляют математический сопроцессор (FPU, floating point unit).

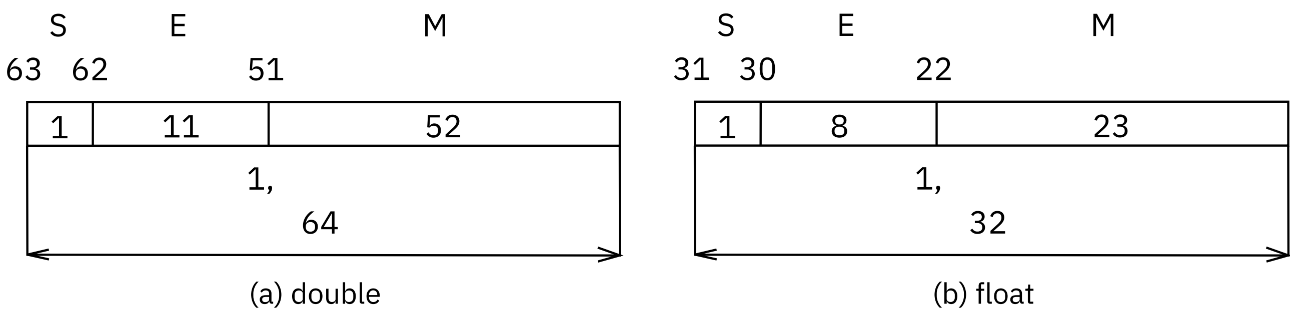


Рис. 4: Типы с плавающей запятой

Рисунок 4 демонстрирует, как распределяются биты в числах с плавающей запятой разных разрядностей, где S - Sign (знак), E - Exponent (8(11) разрядов поля порядка, экспонента), M - Mantissa (23(52) бита мантиссы, дробная часть числа).

Если попытаться уложить весь стандарт в два предложения, то получится примерно следующее: получить число в соответствующих разрядностях возможно по формулам:

$$F_{32} = (-1)^S \times 2^{E-127} \times (1 + \frac{M}{2^{23}})$$

$$F_{64} = (-1)^S \times 2^{E-1023} \times (1 + \frac{M}{2^{52}})$$

[3] хорошо и подробно, но на C, в посте на Хабре.

> Например: $+0,5 = 2^{-1}$ поэтому, число будет записано как
>
> `0_01111110_00000000000000000000000`, то есть знак $= \ 0$, мантисса $= \ 0$, порядок = $127 - 1 = 126$, чтобы получить следующие результаты вычислений:
>
> -1^0 положительный знак, умножить на порядок
> $2^{126-127=-1} = 0,5$ и умножить на мантиссу
> $1 + 0$. То есть, $-1^0 \times 2^{-1} \times (1 + 0) = 0,5$.
> Отсюда становится очевидно, что чем сложнее мантисса и чем меньше порядок, тем более точные и интересные числа мы можем получить.

Возьмём для примера число $-0,15625$, чтобы понять как его записывать, откинем знак, это будет единица в разряде, отвечающем за знак, и посчитаем мантиссу с порядком. Представим число как положительное и будем от него последовательно отнимать числа, являющиеся отрицательными степенями двойки, чтобы получить максимально близкое к нулю значение.

$$2^1 = 2$$
$$2^0 = 1.0$$
$$2^{-1} = 0.5$$
$$2^{-2} = 0.25$$
$$2^{-3} = 0.125$$
$$2^{-4} = 0.0625$$
$$2^{-5} = 0.03125$$
$$2^{-6} = 0.015625$$
$$2^{-7} = 0.0078125$$
$$2^{-8} = 0.00390625$$

Очевидно, что -1 и -2 степени отнять не получится, поскольку мы явно уходим за границу нуля, а вот -3 прекрасно отнимается, значит порядок будет $127 - 3 = 124$, осталось понять, что получится в мантиссе.

Видим, что оставшееся после первого вычитания $(0,15625 - 0,125)$ число - это 2^{-5}. Значит в мантиссе пишем `01` и остальные нули, то есть слева направо указываем, какие степени после -3 будут нужны. -4 не нужна, а -5 нужна.

Получится, что

$$(-1)^1 \times 2^{(124-127)} \times (1 + \frac{2097152}{2^{23}}) = 1,15652$$

или, тождественно,

$$(-1)^1 \times 1,01e-3 =$$
$$1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} =$$
$$1 \times 0,125 + 0 \times 0,0625 + 1 \times 0,03125 =$$
$$0,125 + 0,03125 = 0,15625$$

Так число с плавающей запятой возможно посчитать двумя способами: по приведённой формуле, или последовательно складывая разряды мантиссы умноженные на двойку в степени порядка, уменьшая порядок на каждом шагу.

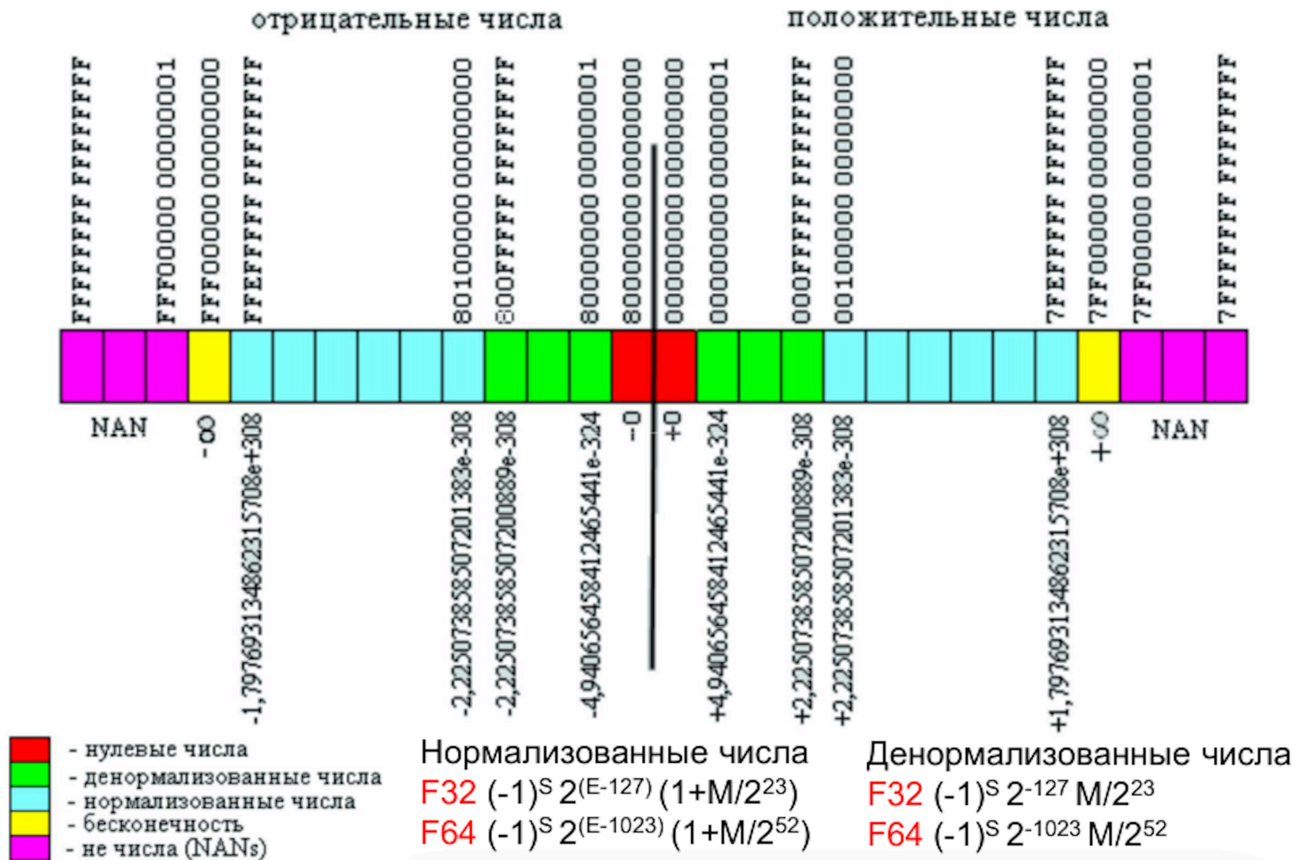


Рис. 5: Особенности работы с числами с плавающей запятой

К особенностям работы чисел с плавающей запятой можно отнести:

— возможен как положительный, так и отрицательный ноль (в целых числах ноль всегда положительный);
— есть огромная зона, отмеченная на рисунке 5, которая являет собой непредставимые числа, слишком большие для хранения внутри такой переменной или настолько маленькие, что мнимая единица в мантиссе отсутствует;
— в таком числе можно хранить значения положительной и отрицательной беско-

нечности;
— при работе с такими числами появляется понятие не-числа, при этом важно помнить, что `NaN != NaN`.

2.2.7. Задания для самопроверки

1. Сколько байт данных занимает самый большой целочисленный тип?
2. Почему нельзя напрямую сравнивать целочисленные данные и числа с плавающей запятой, даже если там точно лежит число без дробной части?
3. Внутри переполненной переменной остаётся значение:
   (a) переданное - максимальное для типа;
   (b) максимальное для типа;
   (c) не определено.

2.2.8. Символы и булевы

Шесть из восьми примитивных типов могут иметь как положительные, так и отрицательные значения, они называются **«знаковые»** типы. В таблице есть два типа, у которых есть диапазон но нет отрицательных значений, это `boolean` и `char`

Булев тип хранит значение `true` или `false`. На собеседованиях иногда спрашивают, сколько места занимает `boolean`. В Java объём хранения не определён и зависит от конкретной JVM, обычно считают, что это один байт.

Тип `char` единственный беззнаковый целочисленный тип в языке, то есть его старший разряд хранит полезное значение, а не признак положительности. Тип целочисленный но по умолчанию среда исполнения интерпретирует его как символ по таблице utf-8 (см фрагмент в таблице 3). В языке Java есть разница между одинарными и двойными кавычками. В одинарных кавычках всегда записывается символ, который на самом деле является целочисленным значением, а в двойных кавычках всегда записывается строка, которая фактически является экземпляром класса `String`. Поскольку типизация строгая, то невозможно записать в `char` строки, а в строки числа.

В Java есть три основных понятия, связанных с данными переменными и использованием значений: объявление, присваивание, инициализация.

Для того чтобы *объявить* переменную, нужно написать её тип и название, также часто вместо названия можно встретить термин идентификатор.
Далее в любой момент можно *присвоить* этой переменной значение, то есть необходимо написать идентификатор использовать оператор присваивания и справа написать значение, которое вы хотите присвоить данной переменной, поставить в конце строки точку с запятой.
Также существует понятие *инициализации* - это когда объединяются на одной строке объявление и присваивание.

dec	hex	val	dec	hex	val	dec	hex	val	dec	hex	val
000	0x00	(nul)	032	0x20	▯	064	0x40	@	096	0x60	‘
001	0x01	(soh)	033	0x21	!	065	0x41	A	097	0x61	a
002	0x02	(stx)	034	0x22	”	066	0x42	B	098	0x62	b
003	0x03	(etx)	035	0x23	#	067	0x43	C	099	0x63	c
004	0x04	(eot)	036	0x24	$	068	0x44	D	100	0x64	d
005	0x05	(enq)	037	0x25	%	069	0x45	E	101	0x65	e
006	0x06	(ack)	038	0x26	&	070	0x46	F	102	0x66	f
007	0x07	(bel)	039	0x27	'	071	0x47	G	103	0x67	g
008	0x08	(bs)	040	0x28	(	072	0x48	H	104	0x68	h
009	0x09	(tab)	041	0x29	)	073	0x49	I	105	0x69	i
010	0x0A	(lf)	042	0x2A	*	074	0x4A	J	106	0x6A	j
011	0x0B	(vt)	043	0x2B	+	075	0x4B	K	107	0x6B	k
012	0x0C	(np)	044	0x2C	’	076	0x4C	L	108	0x6C	l
013	0x0D	(cr)	045	0x2D	-	077	0x4D	M	109	0x6D	m
014	0x0E	(so)	046	0x2E	.	078	0x4E	N	110	0x6E	n
015	0x0F	(si)	047	0x2F	/	079	0x4F	O	111	0x6F	o
016	0x10	(dle)	048	0x30	0	080	0x50	P	112	0x70	p
017	0x11	(dc1)	049	0x31	1	081	0x51	Q	113	0x71	q
018	0x12	(dc2)	050	0x32	2	082	0x52	R	114	0x72	r
019	0x13	(dc3)	051	0x33	3	083	0x53	S	115	0x73	s
020	0x14	(dc4)	052	0x34	4	084	0x54	T	116	0x74	t
021	0x15	(nak)	053	0x35	5	085	0x55	U	117	0x75	u
022	0x16	(syn)	054	0x36	6	086	0x56	V	118	0x76	v
023	0x17	(etb)	055	0x37	7	087	0x57	W	119	0x77	w
024	0x18	(can)	056	0x38	8	088	0x58	X	120	0x78	x
025	0x19	(em)	057	0x39	9	089	0x59	Y	121	0x79	y
026	0x1A	(eof)	058	0x3A	:	090	0x5A	Z	122	0x7A	z
027	0x1B	(esc)	059	0x3B	;	091	0x5B	[	123	0x7B	{
028	0x1C	(fs)	060	0x3C	<	092	0x5C	\	124	0x7C	|
029	0x1D	(gs)	061	0x3D	=	093	0x5D	]	125	0x7D	}
030	0x1E	(rs)	062	0x3E	>	094	0x5E	^	126	0x7E	~
031	0x1F	(us)	063	0x3F	?	095	0x5F	_	127	0x7F	\DEL

Таблица 3: Фрагмент UTF-8 (ASCII) таблицы

2.2.9. Преобразование типов

Java - это язык со строгой статической типизацией, но преобразование типов в ней всё равно есть. Простыми словами, преобразование типов - это когда компилятор видит, что типы переменных по разные стороны присваивания разные, начинает разрешать это противоречие. Преобразование типов бывает явное и неявное.

В разговоре или в сообществах можно услышать или прочитать термины тайпкастинг, кастинг, каст, кастануть, и другие производные от английского typecasting.

Неявное преобразование типов происходит, когда присваиваются числа переменным меньшей размерности, чем `int`. Число справа это `int`, а значит 32 разряда, а слева, например, `byte`, и в нём всего 8 разрядов, но ни среда ни компилятор не поругались, потому что значение в большом `int` не превысило 8 разрядов маленького `byte`. Итак неявное преобразование типов происходит в случаях, когда, «всё и так понятно». В случае, если неявное преобразование невозможно, статический анализатор кода выдаёт ошибку, что ожидался один тип, а был дан другой.

Явное преобразование типов происходит, когда мы явно пишем в коде, что некоторое значение должно иметь определённый тип. Этот вариант приведения типов тоже был рассмотрен, когда к числам дописывались типовые квалификаторы `L` и `f`. Но чаще всего случается, что происходит присваивание переменным не тех значений, которые были написаны в тексте программы, а те, которые получились в результате каких-то вычислений.

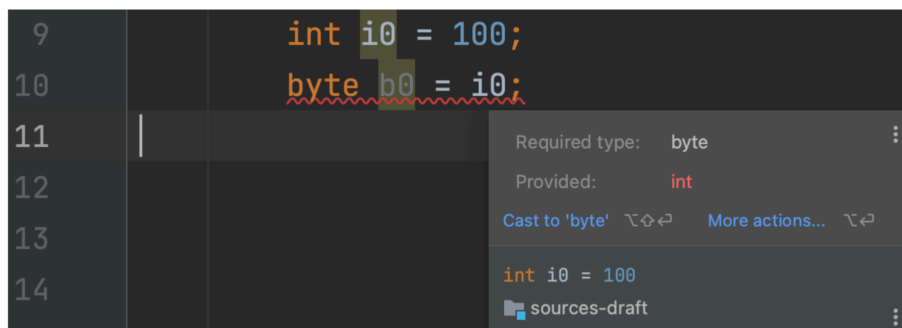


Рис. 6: Ошибка приведения типов

На рис. 6 приведён простейший пример, в котором очевидно, что внутри переменной `i0` содержится значение, не превышающее одного байта хранения, а значит возможно явно сообщить компилятору, что значение точно поместится в `byte`. *Явно преобразовать типы*. Для этого нужно в правой части оператора присваивания перед идентификатором переменной в скобках добавить название типа, к которому необходимо преобразовать значение этой переменной.

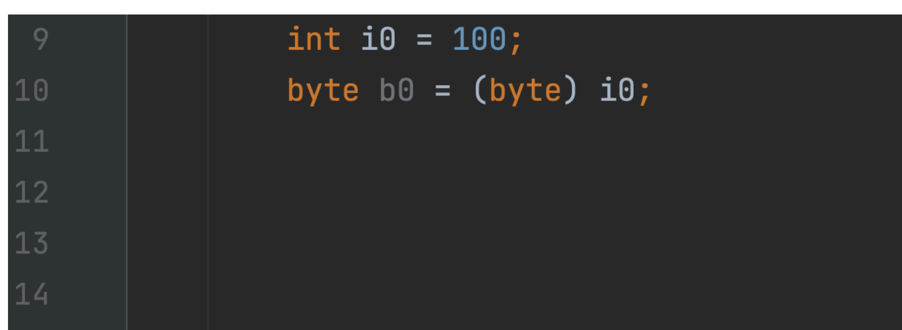


Рис. 7: Верное приведение типов

2.2.10. Константность

Constare - (лат. стоять твёрдо). Константность это свойство неизменяемости. В Java ключевое слово `const` не реализовано, хоть и входит в список ключевых, зарезервированных. Константы создаются при помощи ключевого слова `final`. Ключевое слово файнал возможно применять не только с примитивами, но и со ссылочными типами, методами, классами.

Константа - это переменная или идентификатор с конечным значением.

2.2.11. Задания для самопроверки

1. Какая таблица перекодировки используется для представления символов?
2. Каких действий требует от программиста явное преобразование типов?
3. какое значение будет содержаться в переменной а после выполнения строки int a = 10.0f/3.0f;

2.3. Ссылочные типы данных, массивы

Ссылочные типы данных - это все типы данных, кроме восьми перечисленных примитивных. Самым простым из ссылочных типов является массив. Фактически массив выведен на уровень языка и не имеет специального ключевого слова.

Ссылочные типы отличаются от примитивных местом хранения информации. В примитивах данные хранятся там, где существует переменная и где написан её идентификатор, а по идентификатору ссылочного типа хранится не значение, а ссылка. Ссылку можно представить как ярлык на рабочем столе, то есть очевидно, что непосредственная информация хранится не там, где написан идентификатор. Такое явное разделение идентификатора переменной и данных важно помнить и понимать при работе с ООП.

Массив - это единая, сплошная область данных, в связи с чем в массивах возможно осуществление доступа по индексу

Самый младший индекс любого массива - ноль, поскольку **индекс** - это значение смещения по элементам относительно начального адреса массива. То есть, для получения самого первого элемента нужно сместиться на ноль шагов. Очевидно, что самый последний элемент в массиве из десяти значений, будет храниться по девятому индексу.

Массивы возможно создавать несколькими способами (листинг 1). В общем виде объявление - это тип, квадратные скобки как обозначение того, что это будет массив из переменных этого типа, идентификатор (строка 1). Инициализировать массив можно либо ссылкой на другой массив (строка 2), пустым массивом (строка 3) или заранее

заданными значениями, записанными через запятую в фигурных скобках (строка 4). Присвоить в процессе работы идентификатору возможно только значение ссылки из другого идентификатора или новый пустой массив.

Листинг 1: Объявление массива

```
int[] array0;
int[] array1 = array0;
int[] array2 = new int[5];
int[] array3 = {5, 4, 3, 2, 1};

array2 = {1, 2, 3, 4, 5}; //
```

Никак и никогда нельзя присвоить идентификатору целый готовый массив в процессе работы, нельзя стандартными средствами переприсвоить ряд значений части массива (так называемые слайсы или срезы).

Массивы бывают как одномерные, так и многомерные. Многомерный массив - это всегда массив из массивов меньшего размера: двумерный массив - это массив одномерных, трёхмерный - массив двумерных и так далее. Правила инициализации у них не отличаются. Преобразовать тип массива нельзя никогда, но можно преобразовать тип каждого отдельного элемента при чтении. Это связано с тем, что под массивы сразу выделяется непрерывная область памяти, а со сменой типа всех значений массива эту область нужно будет или значительно расширять или значительно сужать.

Ключевое слово `final` работает только с идентификатором массива, то есть не запрещает изменять значения его элементов.

Если логика программы предполагает создание нижних измерений массива в процессе работы программы, то при инициализации массива верхнего уровня не следует указывать размерности нижних уровней. Это связано с тем, что при инициализации, Java сразу выделяет память под все измерения, а присваивание нижним измерениям новых ссылок на создаваемые в процессе работы массивы, будет пересоздавать области памяти, получается небольшая утечка памяти.

Прочитать из массива значение возможно обратившись к ячейке массива по индексу. Записать в массив значение возможно обратившись к ячейке массива по индексу, и применив оператор присваивания.

```
int i = array[0];
array[1] = 10;
```

В каждом объекте массива есть специальное поле (рис. 8), которое обозначает длину данного массива. Поле находится в классе `__Array__` и является публичной константой.

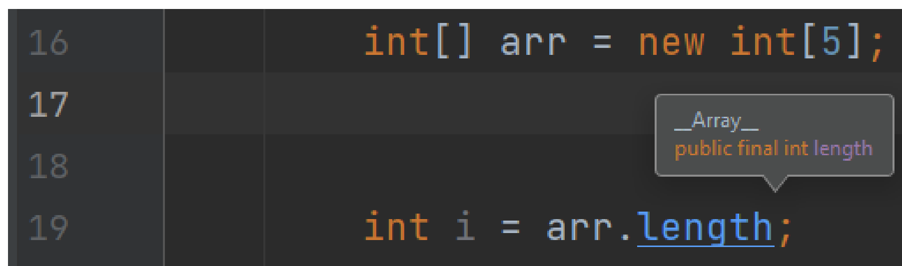


Рис. 8: Константа с длиной массива

2.3.1. Задания для самопроверки

1. Почему индексация массива начинается с нуля?
2. Какой индекс будет последним в массиве из 100 элементов?
3. Сколько будет создано одномерных массивов при инициализации массива 3x3?

2.4. Базовый функционал языка

2.4.1. Математические операторы

Математические операторы работают как и предполагается - складывают, вычитают, делят, умножают, делают это по приоритетам известным нам с пятого класса, а если приоритет одинаков - слева направо. Специального оператора возведения в степень как в пайтоне нет. Единственное, что следует помнить, что оператор присваивания продолжает быть оператором присваивания, а не является математическим равенством, а значит сначала посчитается всё, что слева, а потом результат попробует присвоиться переменной справа. Припоминаем что там за дела с целочисленным делением и отбрасыванием дробной части.

2.4.2. Условия

Условия представлены в языке привычными `if`, `else if`, `else`, «если», «иначе если», «в противном случае», которые являются единым оператором выбора, то есть если исполнение программы пошло по одной из веток, то в другую ветку условия программа точно не зайдёт. Каждая ветвь условного оператора - это отдельный кодовый блок со своим окружением и локальными переменными.

Существует альтернатива оператору `else if` - использование оператора `switch`, который позволяет осуществлять множественный выбор между числовыми значениями. У оператора есть ряд особенностей:

— это оператор, состоящий из одного кодового блока, то есть сегменты кода находятся в одной области видимости. Если не использовать оператор `break`, есть риск «проваливаться» в следующие кейсы;
— нельзя создать диапазон значений;
— достаточно сложно создавать локальные переменные с одинаковым названием для каждого кейса.

2.4.3. Циклы

Циклы представлены основными конструкциями:

— `while () {}`
— `do {} while();`
— `for (;;) {}`

Цикл - это набор повторяющихся до наступления условия действий. `while` - самый простой, чаще всего используется, когда нужно описать бесконечный цикл. `do-while` единственный цикл с постусловием, то есть сначала выполняется тело, а затем принимается решение о необходимости зацикливания, используется для ожидания ответов на запрос и возможного повторения запроса по условию. `for` - классический счётный цикл, его почему-то программисты любят больше всего.

Существует также активно пропагандируемый цикл - `foreach`, работает не совсем очевидным образом, для понимания его работы необходимо ознакомиться с ООП и понятием итератора.

2.4.4. Бинарные арифметические операторы

В современных реалиях мегамощных компьютеров вряд ли кто-то задумывается об оптимизации скорости выполнения программы или экономии занимаемой памяти. Но всё меняется, когда программист впервые принимает сложное решение: запрограммировать микроконтроллер или другой «интернет вещей». Там в вашем распоряжении жалкие пара сотен килобайт памяти, если очень повезёт, в которые нужно не только как-то вложить текст программы и исполняемый бинарный код, но и какие-то промежуточные, пользовательские и другие данные, буферы обмена и обработки. Другая ситуация, в которой нужно начинать «думать о занимаемом пространстве» это разработка протоколов передачи данных, чтобы протокол был быстрый, не передавал по сети большие объёмы данных и быстро преобразовывался. На помощь приходит натуральная для информатики система счисления, двоичная.

Манипуляции двоичными данными представлены в Джава следующими операторами:

— & битовое и;
— | битовое или;
— ~ битовое не;
— ^ исключающее или;
— << сдвиг влево;
— >> сдвиг вправо.

Литеральные «и», «или», «не» уже знакомы по условным операторам. Литеральные операции применяются ко всему числовому литералу целиком, а не к каждому отдельному биту. Их особенность заключается в том, как язык программирования интерпретирует числа.

> В Java в литеральных операциях может участвовать только тип `boolean`, а C++ воспринимает любой ненулевой целочисленный литерал как истину, а нулевой, соответственно, как ложь.

Логика формирования значения при этом остаётся такой же, как и при битовых операциях.

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

A	NOT
0	1
1	0

Таблица 4: Таблицы истинности битовых операторов

Когда говорят о битовых операциях волей-неволей появляется необходимость поговорить о таблицах истинности. В таблице 4 вы видите таблицы истинности для арифметических битовых операций. Битовые операции отличаются тем, что для неподготовленного взгляда они производят почти магические действия, потому что манипулируют двоичным представлением числа.

```
 00000100  4      00000100  4      00000100  4     ~00000100  4
&00000111  7     |00000111  7     ^00000111  7      11111011  -5
 00000100  4      00000111  7      00000011  3
```

Рис. 9: Бинарная арифметика

Число	Бинарное	Сдвиг	Число	Бинарное	Сдвиг
2	000000010	`2 << 0`	128	010000000	`128 >> 0`
4	000000100	`2 << 1`	64	001000000	`128 >> 1`
8	000001000	`2 << 2`	32	000100000	`128 >> 2`
16	000010000	`2 << 3`	16	000010000	`128 >> 3`
32	000100000	`2 << 4`	8	000001000	`128 >> 4`
64	001000000	`2 << 5`	4	000000100	`128 >> 5`
128	010000000	`2 << 6`	2	000000010	`128 >> 6`

Таблица 5: Битовые сдвиги

С битовыми сдвигами работать гораздо интереснее и выгоднее. Они производят арифметический сдвиг значения слева на количество разрядов, указанное справа, в таблице 5 представлены числа, в битовом представлении это одна единственная единица, находящаяся в разных разрядах числа. Это демонстрация сдвига на один разряд влево, и, как следствие, умножение на два. Обратная ситуация со сдвигом вправо, он является целочисленным делением.

- X && Y - литеральная;
- X || Y - литеральная;
- !X - литеральная;
- N << K - $N * 2^K$;
- N >> K - $N/2^K$;
- x & y - битовая. 1 если оба x = 1 и y = 1;
- x | y - битовая. 1 если хотя бы один из x = 1 или y = 1;
- ~x - битовая. 1 если x = 0;
- x ^ y - битовая. 1 если x отличается от y.

2.4.5. Задания для самопроверки

1. Почему нежелательно использовать оператор `switch` если нужно проверить диапазон значений?
2. Возможно ли записать бесконечный цикл с помощью оператора `for`?
3. `2 + 2 * 2 == 2 << 2 >> 1`?

2.5. Функции

Функция - это исполняемый блок кода. Функция, принадлежащая классу называется **методом**.

```
void int method(int param1, int param2) {
    //function body
}

public static void main (String[] args) {
    method(arg1, arg2);
}
```

При объявлении функции в круглых скобках указываются параметры, а при вызове - аргументы.

У функций есть правила именования: функция - это переходный глагол совершенного вида в настоящем времени (вернуть, посчитать, установить, создать), часто снабжаемый дополнением, субъектом действия. Методы в Java пишутся lowerCamelCase. Важно, в каком порядке записаны параметры метода, от этого будет зависеть порядок передачи в неё аргументов. Методы обособлены и их параметры локальны, то есть не видны другим функциям.

Нельзя писать функции внутри других функций.

Все аргументы передаются копированием, не важно, копирование это числовой константы, числового значения переменной или хранимой в переменной ссылке на массив. Сам объект в метод не копируется, а копируется только его ссылка.

Возвращаемые из методов значения появляются в том месте, где метод был вызван. Если будет вызвано несколько методов, то весь контекст исполнения первого метода сохраняется, кладётся (на стек) в стопку уже вызванных методов и процессор идёт выполнять только что вызванный второй метод. По завершении вызванного второго метода, мы снимаем со стека лежащий там контекст первого метода, кладём в него вернувшееся из второго метода значение, если оно есть, и продолжаем исполнять первый метод.

Вызов метода - это, по смыслу, тоже самое, что подставить в код сразу его возвращаемое значение.

Сигнатура метода - это имя метода и его параметры. В сигнатуру метода не входит возвращаемое значение. Нельзя написать два метода с одинаковой сигнатурой.

Перегрузка методов - это механизм языка, позволяющий написать методы с одинаковыми названиями и разными оставшимися частями сигнатуры, чтобы получить единообразие при вызове семантически схожих методов с разнотипными данными.

Практическое задание

1. Написать метод «Шифр Цезаря», с булевым параметром зашифрования и расшифрования и числовым ключом;
2. Написать метод, принимающий на вход массив чисел и параметр n. Метод должен осуществить циклический (последний элемент при сдвиге становится первым) сдвиг всех элементов массива на n позиций;
3. Написать метод, которому можно передать в качестве аргумента массив, состоящий строго из единиц и нулей (целые числа типа `int`). Метод должен заменить единицы в массиве на нули, а нули на единицы и не содержать ветвлений. Написать как можно больше вариантов метода.

Термины, определения и сокращения

Typecasting	преобразование типов переменных в типизированных языках программирования
Идентификатор	идентификатор переменной - название переменной, по которому возможно получить доступ к области памяти, соответствующей этой переменной
Инициализация	одновременной объявление переменной и присваивание ей значения
Массив	структура данных, хранящая набор значений в непрерывной области памяти
Метод	функция в языке программирования, принадлежащая классу
Переполнение	целочи́сленное переполне́ние (англ. integer overflow) — ситуация в компьютерной арифметике, при которой вычисленное в результате операции значение не может быть помещено в тип данных
Типизация	классификация по типам