Синтаксис языка Java

Croc Java School

Комментарии

```
// comment on one line

/* comment on one
 * or more lines
 */

/**
 * Sets the tool tip text.
 *
 * @param text  the text of the tool tip
 */
```

Выражения и блоки

Одна или несколько строк кода, заканчивающихся точкой с запятой, образуют выражение.

```
total = a + b + c
      + d + e + f;
```

Варажения, заключенные в фигурные скобки, образуют блок (или секцию).

```
{
  x = y + 1;
  y = x + 1;
}
```

Вложенные блоки

Блоки могут быть вложенными.

```
{
  int a = 1;
  {
    int b = 2;
  }
  {
    int b = 3;
  }
}
```

Переменные

Переменные

Локальные переменные (определены в методах)

Переменные класса или поля (определены в блоках классов)

Локальные переменные

тип идентификатор;

```
int x;
x = 17;
```

тип идентификатор = значение (выражение);

```
int x = 17;
```

```
int x, y;
int x = 17, y = 99;
```

Инициализация локальных переменных

Переменная должна быть инициализирована до использования в выражении.

```
int x;
int y = x + 2; // error: variable x might not have been initialized
```

```
int x;
x = 1;
int y = x + 2; // ok
```

Константы

```
final double PI = 3.14;

PI = 3.14159; // error
```

Переменные класса (поля)

Блоки с ключевым словом class определяют классы.

```
class MyDate {
  int day = 1;
  int month = 1;
  int year;
}
```

Значения полей по умолчанию

В отличие от локальных переменных у полей есть значения по умолчанию.

`0`	целочисленные типы
`0.0`	вещественные типы
`false`	логический тип
`\u0000`	символьный тип
`null`	ссылочные типы

Идентификаторы

Название переменной, класса или метода.

Может начинаться с буквы (unicode), нижнего подчеркивания (_) или знака доллара ($).

Цифры допускаются за исключением первого символа.

Регистро-зависимы и не ограничены по длине.

Примеры идентификаторов:

- `identifier`
- `userName`
- `user_name`
- `_sys_var1`
- `$change`

Ключевые слова

abstract	continue	for	new	super	while
assert	default	goto	package	switch	
boolean	do	If	private	synchronized	
break	double	implements	protected	this	
byte	else	import	public	throw	
case	enum	instanceof	return	throws	
catch	extends	int	non-sealed	transient	
char	final	interface	short	try	
class	finally	long	static	void	
const	float	native	strictfp	volatile	

Зарезервированные идентификаторы

```
permits
record
sealed
var
yield
```

Зарезервированные литералы

```
true
false
null
```

Система типов

Value-types (примитивы)

- `byte`, `short`, `int`, `long`
- `float`, `double`,
- `boolean`
- `char`

Reference-types (ссылочные или объектные типы)

- экземпляры объектов
- массивы

Примитивы

Целые типы: byte, short, int, long

- Тип для целочисленных литералов по умолчанию — `int`
- Суффикс L или l определяет тип `long`

`2021`	десятичная запись
`2_021`	тоже десятичная запись
`03745`	восьмеричная запись числа 2021 (начинается с нуля)
`0x7E5`	шестнадцатеричная запись
`0b0111_1110_0101`	бинарная запись

Диапазоны значений

Все целые типы в Java знаковые, unsigned в Java нет.

		MIN	MAX
byte	8 bits	-2^7	2^7-1
short	16 bits	-2^{15}	$2^{15}-1$
int	32 bits	-2^{31}	$2^{31}-1$
long	64 bits	-2^{63}	$2^{63}-1$

Представление целых чисел

Положительные: в двоичной форме

`2021: 0000 0000 0000 0000 0000 0111 1110 0101`

Отрицательные: в дополнительном коде (two's complement)

Дополнительный код для отрицательного x = xc : `xc + |x| = 2^N`

где N — размер типа в битах

Как посчитать:

```
xc = ~|x| + 1
```

Дополнительный код (пример)

```
x = -2021
 |x|     : 0000 0000 0000 0000 0000 0111 1110 0101
~|x|     : 1111 1111 1111 1111 1111 1000 0001 1010
~|x| + 1 : 1111 1111 1111 1111 1111 1000 0001 1011 = xc
```

Проверим:

```
int x = -2021;
System.out.println(Integer.toBinaryString(x));
out: 11111111111111111111100000011011
```

Вещественные типы: float, double

- Суффикс D или d определяет тип `double` (по умолчанию)
- Суффикс F или f определяет тип `float`

`3.14`	дробная часть отделяется точкой
`6.02E23`	экспоненциальная запись = 6.02×10^{23}
`2.718F`	суффикс F определяет константу типа `float`
`123.4E+306D`	суффикс D избыточный, но не запрещается
`13.`	дробная часть может быть опущена (13.0)
`.13`	вещественная часть тоже может быть опущена (0.13)

Размер float и double

float	32 bits
double	64 bits

Представление вещественных чисел

float: S (1 бит) E (8 бит) M (23 бита)

S знаковый бит, E смещенная экспонента, M мантисса (целое число)

$f = -1^{S} \times 2^{E - 127} \times (1 + M / 2^{23})$

$M / 2^{23} \in [0, 1)$

Представление вещественных чисел

Специальные значения

+0, -0: E = 0, M = 0

+∞, -∞: E = 255, M = 0

NaN (not-a-number): E = 255, M != 0

Логический тип: boolean

Тип boolean оперирует двумя значениями: “истина” и “ложь”, которые определяются литералами true и false.

Выражение

```
boolean truth = true;
```

определяет переменную `truth` логического типа `boolean` со значением “истина”.

Текстовый тип: char

Целый беззнаковый тип: 16-битный символ Unicode

Литералы заключаются в одинарные кавычки (’A’)

'a'	буква a
'\t'	символ табуляции
'\u????’	символ кодировки Unicode, ???? значение в 16-ричном виде (ровно 4 символа)
‘\u03A6’	греческая буква фи [Ф]

Escape-последовательности

\t	tab
\b	backspace
\n	newline
\r	carriage return
\f	form feed
\’	single quote
\”	double quote
\\	backslash

Текстовый тип: String

Тип String это класс, а не примитивный тип.

Значение задается литералом, заключенным в двойные кавычки.

```
"The quick brown fox jumps over the lazy dog."
```

Примеры:

```
String greeting = "Good Morning!\n";
String errorMessage = "Record Not Found";
```

Текстовые блоки (Java 15)

```
String example = """
     Text
          with
               line breaks""";
```

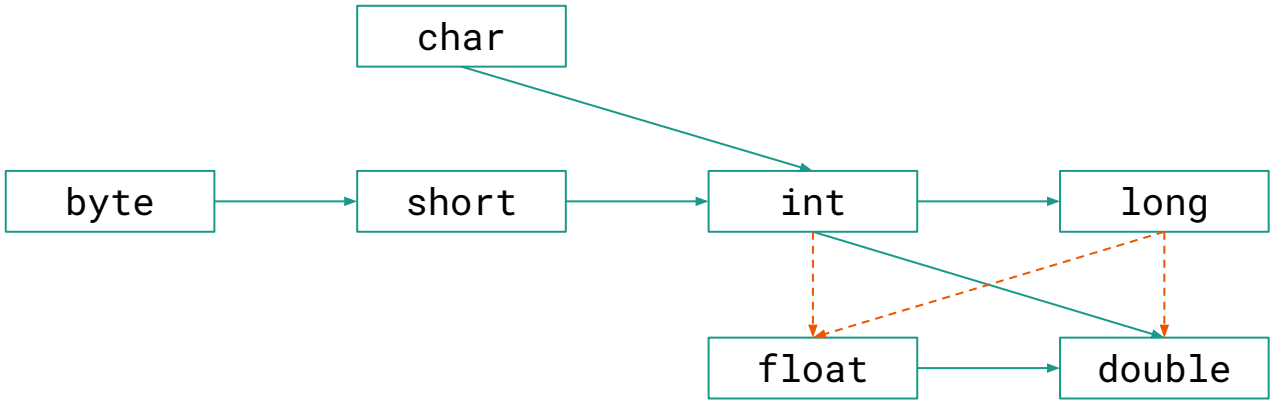
Результат (минимальный отступ отрезается)

```
Text
      with
           line breaks
```

Преобразование типов

Преобразования типов

1. Неявные без потери точности (расширяющие) ->
2. Неявные с потерей точности ->
3. Явные (сужающие)



Преобразования типов (примеры)

```
short a = 16; // ok, compile-time constant
```

```
int v = 16;
short a = v; // error: possible lossy conversion from int to short
```

```
int v = 16;
short a = (short)v; // ok
```

Преобразования типов (примеры)

```
double v = 32.32;
int a = v; // error
```

```
double v = 32.32;
int a = (int)v; // ok: a = 32
```

```
int v = 32;
float a = v; // ok: a = 32.0
```

```
boolean b = (boolean)1; // error
```

Ссылочные типы

Ссылочные типы

Вторая разновидность типов данных после примитивов — ссылочные типы.

Значением переменной ссылочного типа является ссылка на объект (адрес объекта в памяти).

```
public class MyDate {
    private int day = 1;
    private int month = 1;
    private int year = 2000;
    public MyDate(int day, int month, int year) { ... }
    public String toString() { ... }
}
```

Создание объектов

```
MyDate today = new MyDate(14, 10, 2021);
```

При вызове конструктора:

- выделяется необходимая память для полей объекта
- поля инициализируются значениями по умолчанию (или указанными в определении)
- выполняется блок метода конструктора
- оператор new возвращает ссылку на созданный объект

Значение ссылки присваивается переменной.

Создание объекта

```
MyDate today = new MyDate(14, 10, 2021);
```

Объявление переменной резервирует память для сохранение ссылки.

Выделение памяти объекта

```
MyDate today = new MyDate(14, 10, 2021);
```

Оператор new выделяет память для полей нового объекта и инициализирует их значениями по умолчанию.

Инициализация полей

```
MyDate today = new MyDate(14, 10, 2021);
```

В поля записываются значения, указанные в определении класса.

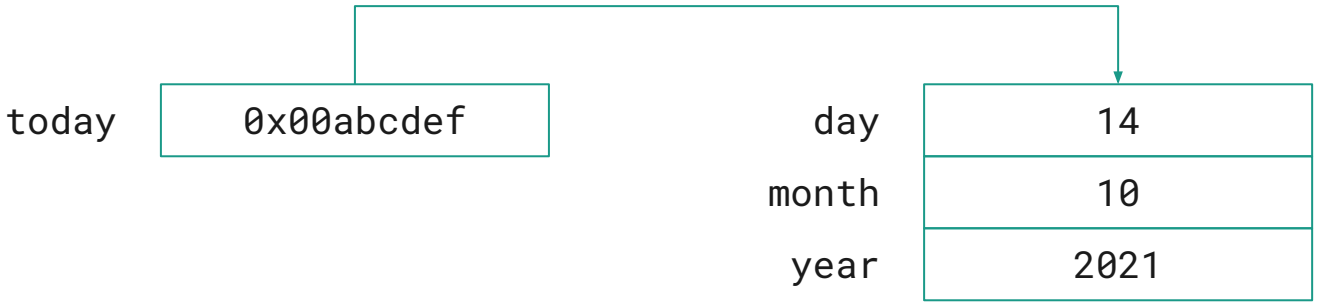
Исполнение блока конструктора

```
MyDate today = new MyDate(14, 10, 2021);
```

Инициализация ссылки

```
MyDate today = new MyDate(14, 10, 2021);
```

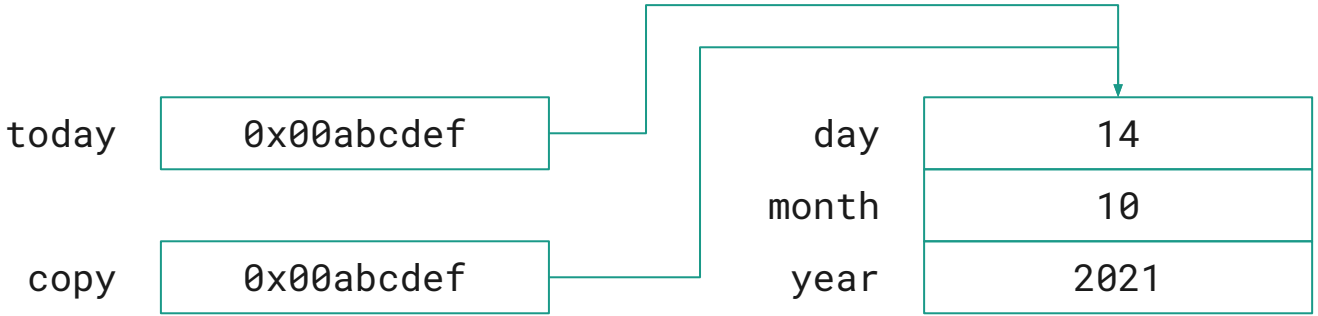
Конструктор возвращает ссылку на сконструированный объект, которая присваивается переменной.



Копирование ссылки

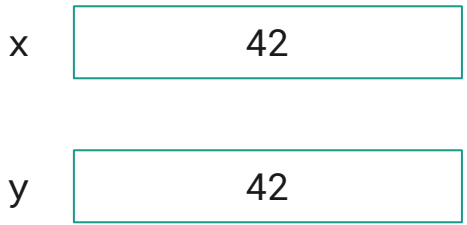
Две переменные могут ссылаться на один и тот же объект.

```
MyDate today = new MyDate(14, 10, 2021);
MyDate copy = today; // копируется только ссылка
```



Копирование значения

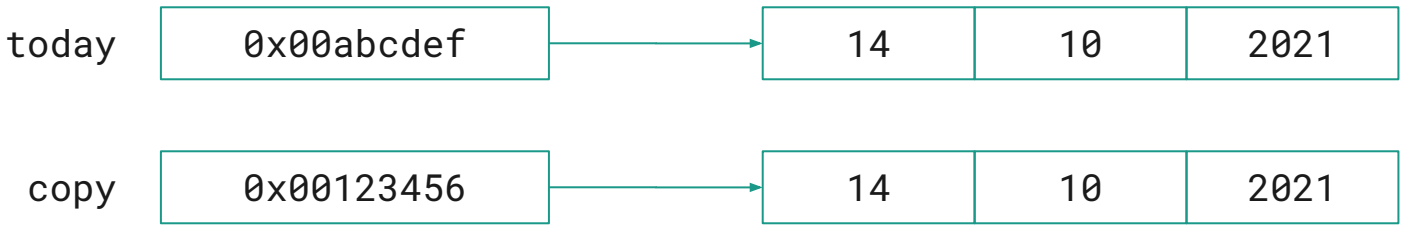
```
int x = 42;
int y = x; // копируется значение
```



Создание копии объекта

Для того, чтобы ссылки указывали на два разных объекта, объект-копию, необходимо сконструировать явно.

```
MyDate today = new MyDate(14, 10, 2021);
MyDate copy = new MyDate(14, 10, 2021); // создается отдельная копия
```



Передача “по значению”

Аргументы в методы в Java всегда передаются по значению.

При передаче объекта в метод в качестве параметра, значение аргумента — ссылка на этот объект, и она копируется.

Поля объекта при этом могут быть изменены в теле метода, но значение ссылки перезаписать нельзя.

Аргументы-примитивы

```
void increment(int v) {
  v = v + 1;
}

int a = 5;
increment(a);
System.out.println(a);

Out: ??
```

Аргументы-примитивы

```
void increment(int v) {
  v = v + 1;
}

int a = 5;
increment(a);
System.out.println(a);

Out: 5
```

Аргументы-ссылки

```
void reset(MyDate date) {
  date = new MyDate(1, 1, 2000);
}

MyDate date = new MyDate(14, 10, 2021);
reset(date);
System.out.println(date);

Out: ??
```

Аргументы-ссылки

```
void reset(MyDate date) {
  date = new MyDate(1, 1, 2000);
}

MyDate date = new MyDate(14, 10, 2021);
reset(date);
System.out.println(date);

Out: 2021-10-14
```

Аргументы-ссылки

```
void reset2(MyDate date) {
  date.day = 1;
  date.month = 1;
  date.year = 2000;
}

MyDate date = new MyDate(14, 10, 2021);
reset2(date);
System.out.println(date);

Out: ??
```

Аргументы-ссылки

```
void reset2(MyDate date) {
  date.day = 1;
  date.month = 1;
  date.year = 2000;
}

MyDate date = new MyDate(14, 10, 2021);
reset2(date);
System.out.println(date);

Out: 2000-01-01
```

Операторы

постфиксные	`a++ a--`
унарные	`++a --a +a -a ~ !`
мультипликативные	`* / %`
аддитивные	`+ -`
сдвиги	`<< >> >>>`
порядок	`< > <= >= instanceof`
равенство	`== !=`
побитовый AND	`&`
побитовый XOR	`^`
побитовый OR	`|`
логический AND	`&&`
логический OR	`||`
тернарный	`? :`
присваивание	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

Приоритеты операторов

См. таблицу на предыдущем слайде.

Переопределяются с помощью скобок.

Операторы с одинаковым приоритетом:

- слева направо
- присваивания: справа налево

Снова преобразование типов

Если типы операторов не совпадают, выполняется неявное расширение типа операторов до общего.

```
1 + 1.0; // double


char c = ‘A’;
byte b = 1;
c + b; // int
```

Деление

```
int a = 7;
System.out.println(a / 0); // ArithmeticException
System.out.println(a / 0.0); // Infinity
```

```
7 / 3 = 2
7.0 / 3 = 2.3333333333333335
```

Сдвиги

```
2021           : 0000 0000 0000 0000 0000 0111 1110 0101
-2021          : 1111 1111 1111 1111 1111 1000 0001 1011
2021 << 3      : 0000 0000 0000 0000 0011 1111 0010 1000
2021 >> 2      : 0000 0000 0000 0000 0000 0001 1111 1001
-2021 >> 2     : 1111 1111 1111 1111 1111 1110 0000 0110
-2021 >>> 2    : 0011 1111 1111 1111 1111 1110 0000 0110
```

$a << k = a \times 2^k$

$a >> k = a \div 2^k$

Короткое замыкание && и ||

P1 && P2: если P1 false, то P2 не выполняется

P1 || P2: если P1 true, то P2 не выполняется

```
String a = null, b = “test”;
if (a != null && a.equals(b)) { // ok
  // ...
}
if (a != null & a.equals(b)) { // NullPointerException
  // ...
}
```

Тернарный оператор

```
int sign = x >= 0 ? 1 : -1;


int sign;
if (x >= 0) {
  sign = 1;
} else {
  sign = -1;
}
```

instanceof

Проверка типа переменной:

- является экземпляром класса
- является экземпляром дочернего класса (по отношению к типу переменной)
- реализует интерфейс

```
class A {}

class B extends A {}

A a = new B();

a instanceof B => true
```

Управляющие конструкции

if-else

```
if (predicate) {
  // исполняется, если выражение predicate истинно
} else {
  // исполняется, если выражение predicate ложно
}

// скобки можно опустить для однострочных выражений,
// но такой записи стоит по возможности избегать
if (a > 0)
    a--;
```

if-else if

```
if (month >= 3 && month <= 5) {
  System.out.println(“Весна”);
} else if (month >= 6 && month <= 8) {
  System.out.println(“Лето”);
} else if (month >= 9 && month <= 11) {
  System.out.println(“Осень”);
} else {
  System.out.println(“Зима”);
}
```

switch

```
switch (month) {
  case 3: case 4: case 5:
    System.out.println(“Весна”);
    break;
  case 6: case 7: case 8:
    System.out.println(“Лето”);
    break;
  case 9: case 10: case 11:
    System.out.println(“Осень”);
    break;
  default:
    System.out.println(“Зима”);
    break;
}
```

Типы switch-выражения:

- целочисленные типы
- текстовые типы: char, String
- типы перечислений

while, do-while

```
while (predicate) {
  // predicate проверяется
  // на истинность перед
  // выполнения тела цикла
}
```

```
while (true) {
  if (!predicate)
    break;
  // ...
}
```

```
do {
  // predicate проверяется
  // на истинность после
  // выполнения тела цикла
} while (predicate);
```

Тело цикла выполнится хотя минимум один раз.

for

```
for (init; predicate; advance) {
  statement;
}
```

```
int[] v = new int[5];
for (int i = 0; i < v.length; i++) {
  v[i] = -1;
}
```

```
init;
while (predicate) {
  statement;
  advance;
}
```

```
int[] v = new int[5];
int i = 0; // отличается от for видимостью
while (i < v.length) {
  v[i] = -1;
  i++;
}
```

for each

```
int[] values = new int[5];
for (int value : values) {
  System.out.println(value);
}
```

break, continue, return

`break`

завершаем исполнение блока, покидаем цикл

`continue`

завершаем исполнение блока, продолжаем цикл

`return`

завершаем исполнение метода

Метки

```
int[][] matrix = {{0, 1}, {1, 0}};

outer:
for (int[] row : matrix} {
  for (int value : row) {
    if (value == 0)
      break outer;
  }
}
```

Coding Conventions

Пакеты

```
com.example.domain
```

Классы, интерфейсы и типы перечислений

```
SavingsAccount
```

Методы

```
getAccount()
```

Переменные

```
currentCustomer
```

Константы

```
HEAD_COUNT
```

Coding Conventions

Расстановка фигурных скобок

```
if (condition) {
    statement1;
} else {
    statement2;
}
```