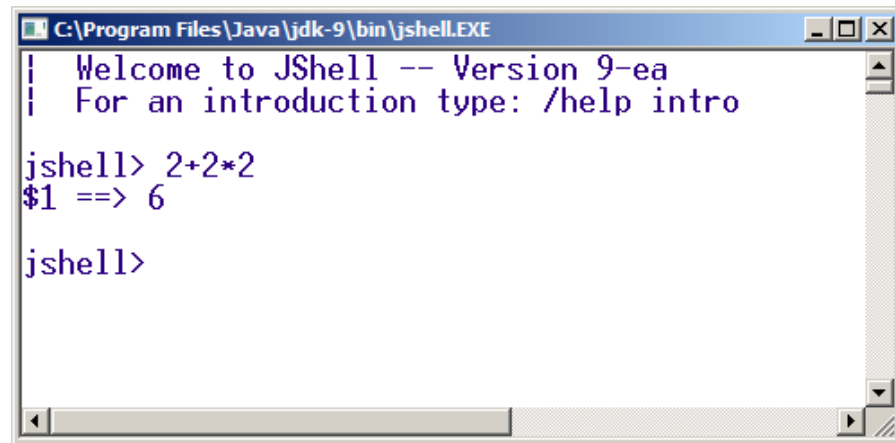


# Базовый синтаксис Java

Тагир Валеев

# JShell (Java 9)

- REPL:
  - Read – считать
  - Evaluate – вычислить
  - Print – напечатать
  - Loop – в цикле



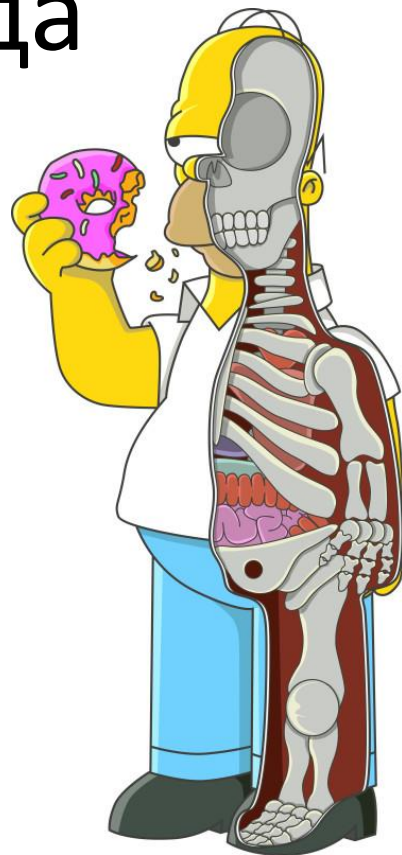
```
C:\Program Files\Java\jdk-9\bin\jshell.EXE
Welcome to JShell -- Version 9-ea
For an introduction type: /help intro

jshell> 2+2*2
$1 ==> 6

jshell>
```

# Анатомия Java-метода

- Code block (блок кода)
- Statement (оператор, предложение, инструкция)
- Expression (выражение)
- Declaration (объявление)
- Expression list (список выражений)
- Resource list (список ресурсов)
- Statement list (список предложений)



```
public static void main(String[] args) {  
    System.out.println("Hello World!");  
}
```

Блок кода  
(реализация метода)

```
public static void main(String[] args) {  
    System.out.println("Hello World!");  
}
```

Предложение  
(предложение-выражение)

```
public static void main(String[] args) {  
    System.out.println("Hello World!");  
}
```

Выражение  
(вызов метода)

```
public static void main(String[] args) {  
    System.out.println("Hello World!");  
}
```

Выражение  
(вызов метода)

```
public static void main(String[] args) {  
    System.out.println("Hello World!");  
}
```

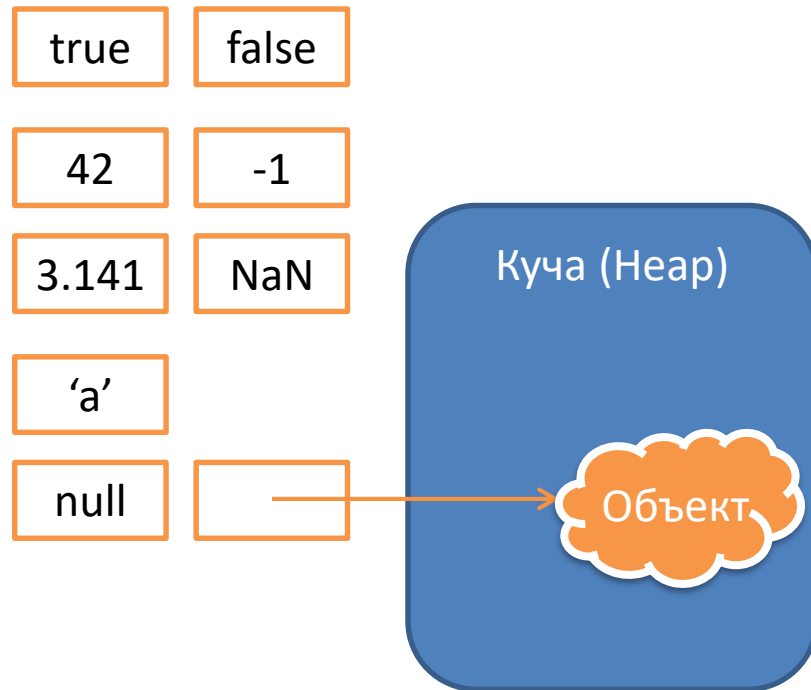
Выражение  
(квалификатор вызова;  
ссылка на поле)

```
public static void main(String[] args) {  
    System.out.println("Hello World!");  
}
```

Выражение  
(аргумент метода;  
строковый литерал)

# Значения и их типы

- ✓ Логическое: boolean
- ✓ Целочисленные: byte, short, int, long
- ✓ Дробные: float, double
- ✓ Символьные (UTF-16 code unit): char
- ✓ Ссылочные



# ~~Chuck Norris~~ value facts

- ✓ Значение есть результат вычисления выражения (за исключением выражения типа `void`)
- ✓ Значения можно сохранять в переменные (поля, константы и т. д.)
- ✓ Значения можно передавать в методы аргументом и принимать в виде параметра
- ✓ При передаче в метод, присваивании значение копируется.
- ✓ Значение нельзя изменить, но можно присвоить в переменную новое значение

# Объявление локальных переменных

[final] Тип имя[ = инициализирующее выражение][, ...];

- ✓ **int** x;
- ✓ **int** x = 5;
- ✓ **final int** x = 5;
- ✓ **final int** x;
- ✓ **int** x, y;
- ✓ **int** x, y = 5;



# ~~Chuck Norris~~ variable facts

- ✓ Локальная переменная видима в блоке кода, в котором объявлена (включая вложенные блоки)
- ✓ Локальная переменная существует в блоке кода, в котором объявлена
- ✓ Во вложенном блоке нельзя объявить переменную с тем же именем (за исключением вложенного объявления класса)
- ✓ Значение `final`-переменной можно присвоить только один раз, но необязательно в одном месте.
- ✓ Никто не покусится на значения ваших параметров и локальных переменных (ни вызванные методы, ни другие потоки)

# Присваивание (выражение)

- ✓ `x = 5;`
- ✓ `x = y;`
- ✓ `x = y = 5;`
- ✓ `x = (y = 5);` // правая ассоциативность
- ✓ `System.out.println(x = 5);`
- ✓ `x += 5;` // составное присваивание

# Логические значения

- ✓ boolean
- ✓ Класс-обёртка (box): Boolean
- ✓ Значения: true, false

Операция	Знак	Сокращённая	Составное присваивание
And (И)	&	&&	&=
Or (ИЛИ)			=
Xor	^, !=		^=
Eq (ЭКВ)	==		
Not	!		^= true

# Таблицы истинности

&	F	T
F	F	F
T	F	T

	F	T
F	F	T
T	T	T

^	F	T
F	F	T
T	T	F

==	F	T
F	T	F
T	F	T

	!
F	T
T	F

# Целые числа

Название	Класс-обёртка	MIN_VALUE	MAX_VALUE
byte	Byte	$-2^7 = 128$	$2^7 - 1 = 127$
short	Short	$-2^{15} = -32\,768$	$2^{15} - 1 = 32\,767$
char	Character	0	$2^{16} - 1 = 65\,535$
int	Integer	$-2^{31} = -2\,147\,483\,648$	$2^{31} - 1 = 2\,147\,483\,647$
long	Long	$-2^{63} \approx 9 \cdot 10^{18}$	$2^{63} - 1 \approx 9 \cdot 10^{18}$

# Целочисленные литералы

- ✓ Десятичное число типа int: 123
- ✓ Десятичное число типа long: 9\_876\_543\_210L
- ✓ Шестнадцатеричное число: 0x1234\_ABCD, 0xFFFF\_FFFF
- ✓ Восьмеричное число: 0123
- ✓ Двоичное число: 0b1010\_1010\_1010\_1010

# Операции над целыми числами

## Арифметические:

✓ Унарный минус:	-		
✓ Сложение:	+	+=	
✓ Вычитание:	-	-=	
✓ Умножение:	*	*=	
✓ Деление:	/	/=	деление на 0: ArithmeticException
✓ Остаток:	%	%=	деление на 0: ArithmeticException
✓ Инкремент:		++	
✓ Декремент:		--	

# А без переполнения?

- ✓ `Math.addExact()`
- ✓ `Math.subtractExact()`
- ✓ `Math.multiplyExact()`
- ✓ `Math.incrementExact()`
- ✓ `Math.decrementExact()`
- ✓ `Math.negateExact()`





# Ещё полезные штуки

- ✓ `Math.abs()`
- ✓ `Math.max()`
- ✓ `Math.min()`
- ✓ `Integer.parseInt()`
- ✓ `Integer.parseUnsignedInt()`
- ✓ `Integer.signum()`

# Операции над целыми числами

## Побитовые:

✓ Дополнение (not):	~	
✓ Побитовое И (and):	&	&=
✓ Побитовое ИЛИ (or):		=
✓ Побитовое исключающее ИЛИ (xor):	^	^=
✓ Сдвиг влево (shl):	<<	<<=
✓ Сдвиг вправо (shr):	>>	>>=
✓ Беззнаковый сдвиг вправо (ushr):	>>>	>>>=

Integer.toBinaryString(), reverse(), bitCount(),  
rotateLeft(), rotateRight(), numberOfLeadingZeros(), ...

# Символьные литералы

Литерал:

```
char c = 'a';
```

```
char c = 'a' + 1;
```

```
char c = 'Ё';
```

```
char c = '\u65e5';
```

```
class Hello {  
    public static void main(String[] args) {  
        // Безобидный комментарий \u000a System.out.println("Bugaga");  
        System.out.println("Hello World");  
    }  
}
```

# Символьные литералы

Литерал:

```
char c = 'a';
```

```
char c = 'a' + 1;
```

```
char c = 'Ё';
```

```
char c = '\u65e5';
```

```
char c = '\\';
```

```
char c = '\n';
```

# char

Литерал:

```
char c = 'a';
```

```
char c = 'a' + 1;
```

```
char c = 'Ё';
```

```
char c = '\u65e5';
```

```
char c = '\\';
```

```
char c = '\n';
```

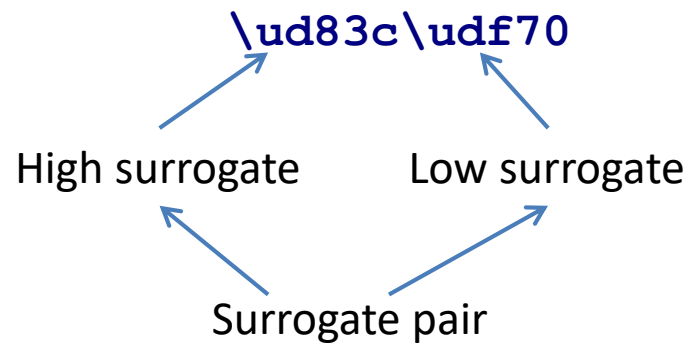
char c = '🍰';

# char

Литерал:

```
char c = 'a';  
char c = 'a' + 1;  
char c = 'Ё';  
char c = '\u65e5';  
char c = '\\';  
char c = '\n';
```

**char** C = '🍰';



# Некоторые термины Unicode

[www.unicode.org/glossary/](http://www.unicode.org/glossary/)

- ✓ Code point – номер символа (0-0x10FFFF = 1114111)
- ✓ Basic Multilingual Plane (BMP) – символы 0-65535 ('uFFFF'), «плоскость 0»
- ✓ Supplementary Plane – плоскости 1-16 (символы 0x10000-0x10FFFF)
- ✓ Surrogate code point – 0xD800-0xDFFF
- ✓ High surrogate code point – 0xD800-0xDBFF
- ✓ Low surrogate code point – 0xDC00-0xDFFF
- ✓ Code unit – минимальная последовательность бит для представления кодовых точек в заданной кодировке
  - ✓ UTF-8: 1 байт, 0..255
  - ✓ UTF-16: 2 байта, 0..65535 (Java!)
  - ✓ UTF-32: 4 байта



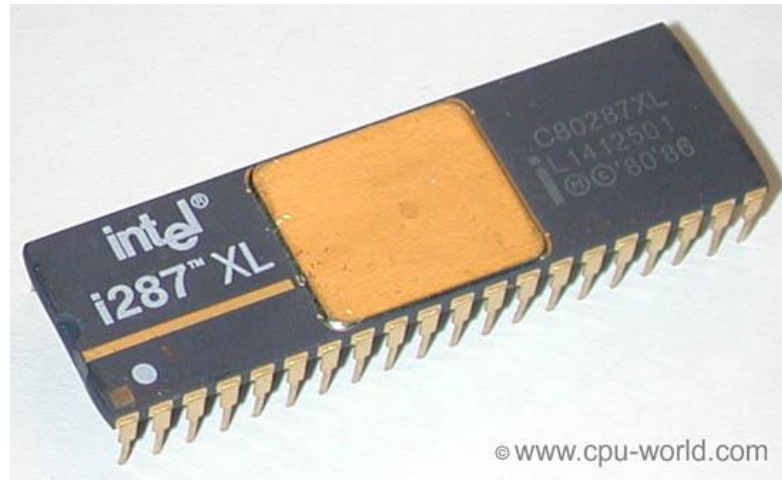
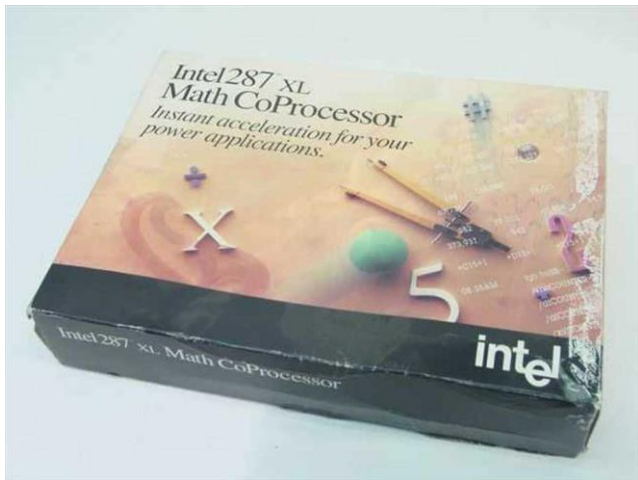
# Операции над символами

- ✓ `Character.toUpperCase('a')`
- ✓ `Character.toUpperCase('dž')`
- ✓ `Character.isAlphabetic('あ')`
- ✓ `Character.isDigit('5')`
- ✓ `Character.getNumericValue('5')`
- ✓ ...

# Числа с плавающей запятой

Floating point numbers

$$1234 = 123.4 \cdot 10^1 = 12.34 \cdot 10^2 = 1.234 \cdot 10^3$$



© www.cpu-world.com



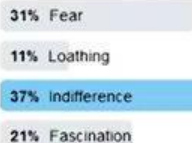
## Survey from 538.0: Floating-point arithmetic vs. spiders

More frightening and more loathsome!



Joseph Darcy  
@jddarcy

What is your primary reaction to spiders?



108 votes • Final results

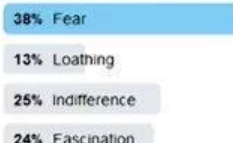
4:15 PM - 29 Jun 2017

*(If you've already overcome arachnophobia,  
level-up to take on tailless whip scorpions!)*



Joseph Darcy  
@jddarcy

What is your primary reaction to floating-point arithmetic?



537 votes • Final results

4:15 PM - 29 Jun 2017

ORACLE®

# Вещественные типы

Тип	Обёртка	MIN_VALUE	MAX_VALUE	Бит	Мантисса	Порядок
float	Float	$\approx 1.4 \cdot 10^{-45}$	$\approx 3.4 \cdot 10^{38}$	32	23	8
double	Double	$\approx 4.9 \cdot 10^{-324}$	$\approx 1.798 \cdot 10^{308}$	64	52	11

# Вещественные литералы

- ✓ Литерал типа double: 1.0
  - ✓ С суффиксом: 1D
- ✓ Литерал типа float: 1F
- ✓ С экспонентой:  $1.6e-19 = 1.6 \cdot 10^{-19}$
- ✓ Шестнадцатеричный:  $0x11P-3 = 0x11 \cdot 2^{-3}$

# Особые числа

- ✓ **+0.0, -0.0**

- ✓ Равны по `==`, но различаются по `toString()`

- ✓ **Infinity (Double.POSITIVE\_INFINITY)**

- ✓ Больше всякого другого числа, положительное

- ✓  $1/\text{Infinity} = 0.0$

- ✓  $\text{Infinity} + 1 = \text{Infinity}$ ;  $\text{Infinity} + \text{Infinity} = \text{Infinity}$

- ✓  $\text{Infinity} - \text{Infinity} = \text{NaN}$

- ✓ **-Infinity (Double.NEGATIVE\_INFINITY)**

- ✓ Меньше всякого другого числа, отрицательное

- ✓  $1/-\text{Infinity} = -0.0$

- ✓ **NaN (Double.NaN)**

- ✓ Не больше, не меньше и не равно никакому числу (в том числе себе)

- ✓ Любая операция с NaN даст NaN

# Операции над вещественными числами

✓ Унарный минус: -

✓ Сложение: + +=

✓ Вычитание: - -=

✓ Умножение: \* \*=

✓ Деление: / /= деление на 0: Double.Infinity/Double.NaN

✓ Остаток: % %= деление на 0: Double.NaN

✓ Инкремент: ++

✓ Декремент: --

Double.isNaN(), isFinite(), isInfinite(), doubleToLongBits()

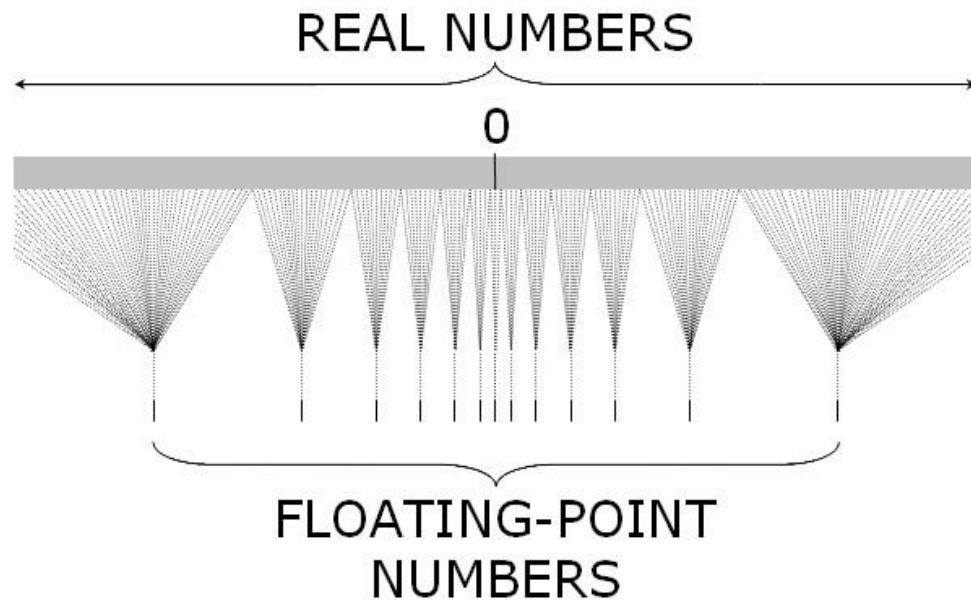
Long.longBitsToDouble()

Math.sin(), cosh(), acos(), atan2(), pow(), floor(), ceil(), round(), log1p(), toRadians()

Math.ulp(), nextUp(), nextDown()

# Машинная точность

```
double x = 0.1;  
x += 0.1;  
x += 0.1;  
System.out.println(x == 0.3);  
System.out.println(x);
```





# Всего 64 бита?



# Можно и больше!

java.math.BigInteger  
java.math.BigDecimal

# Расширяющее преобразование примитивных типов

## Widening primitive conversion

✓ byte → short → int → long → float → double  
char →

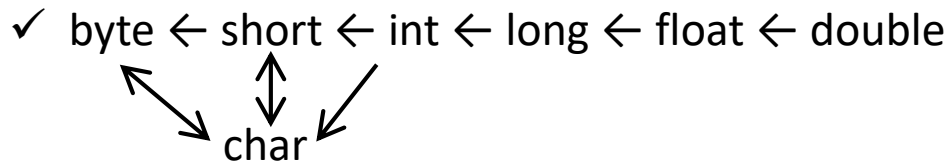
```
short s = b;  
int i = s;  
long l = i;  
float f = l;  
double d = f;
```

### Потеря точности:

- ✓ int → float
- ✓ long → float
- ✓ long → double

# Сжимающее преобразование примитивных типов

## Narrowing primitive conversion



Приведение типа (type cast)

```
float f = (float) d;
```

```
long l = (long) f;
```

```
int i = (int) l;
```

```
short s = (short) i;
```

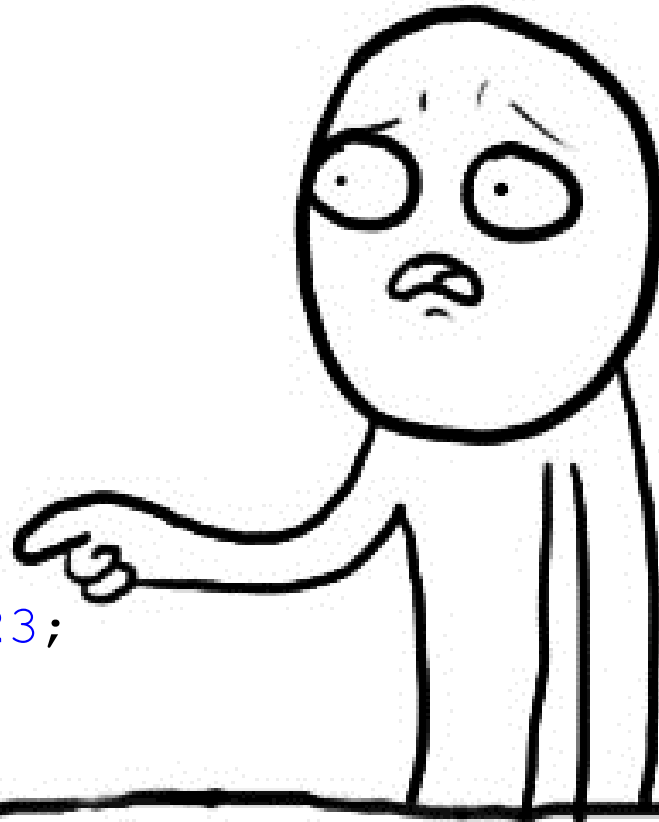
```
char c = (char) s;
```

```
byte b = (byte) c;
```

**Потеря точности:**

всегда

`byte b = 123;`



# Константы времени компиляции

- ✓ Литералы
- ✓ Инициализированные final-переменные примитивных типов и типа String, если инициализатор – константа.
- ✓ Математические, логические, битовые операции над константами
- ✓ Преобразования примитивных типов
- ✓ Конкатенация (сложение) строк
- ✓ Скобки
- ✓ Условный оператор, если все операнды – константы.

```
final int i = 2 + 2 * 2;  
byte b = i;
```

```
byte b = Integer.MAX_VALUE + Integer.MAX_VALUE;
```

```
public static final int MAX_VALUE = 0x7fffffff;
```



```
byte b = Integer.MAX_VALUE + Integer.MAX_VALUE;
```

Numeric overflow in expression [more...](#) (Ctrl+F1)

# Преобразования типов в выражениях

- ✓ Унарная операция, битовый сдвиг, индекс массива: **byte**, **short**, **char** → **int**
- ✓ Бинарная операция (кроме сдвига):
  - ✓ Если любой операнд **double**, другой → **double**
  - ✓ Иначе если любой операнд **float**, другой → **float**
  - ✓ Иначе если любой операнд **long**, другой → **long**
  - ✓ Иначе оба операнда → **int**

# Преобразования типов в выражениях

- ✓ Унарная операция, битовый сдвиг, индекс массива: **byte**, **short**, **char** → **int**
- ✓ Бинарная операция (кроме сдвига):
  - ✓ Если любой операнд **double**, другой → **double**
  - ✓ Иначе если любой операнд **float**, другой → **float**
  - ✓ Иначе если любой операнд **long**, другой → **long**
  - ✓ Иначе оба операнда → **int**

```
double a = Long.MAX_VALUE;  
long b = Long.MAX_VALUE;  
int c = 1;  
System.out.println(a+b+c);  
System.out.println(c+b+a);
```

Ассоциативность сложения?



Не, не слышал

```
byte b = 1;
```

```
b = b + 1;
```

```
b += 1;
```

```
byte b = 1;
```

```
b = b + 1;
```

```
b += 1;
```

```
char c = 'a';
```

```
c *= 1.2;
```

```
System.out.println(c);
```

# Массивы

```
int[] ints;  
byte[] bytes;  
String[] strings;  
int[][] table;
```

# Массивы

```
int[] ints;
```



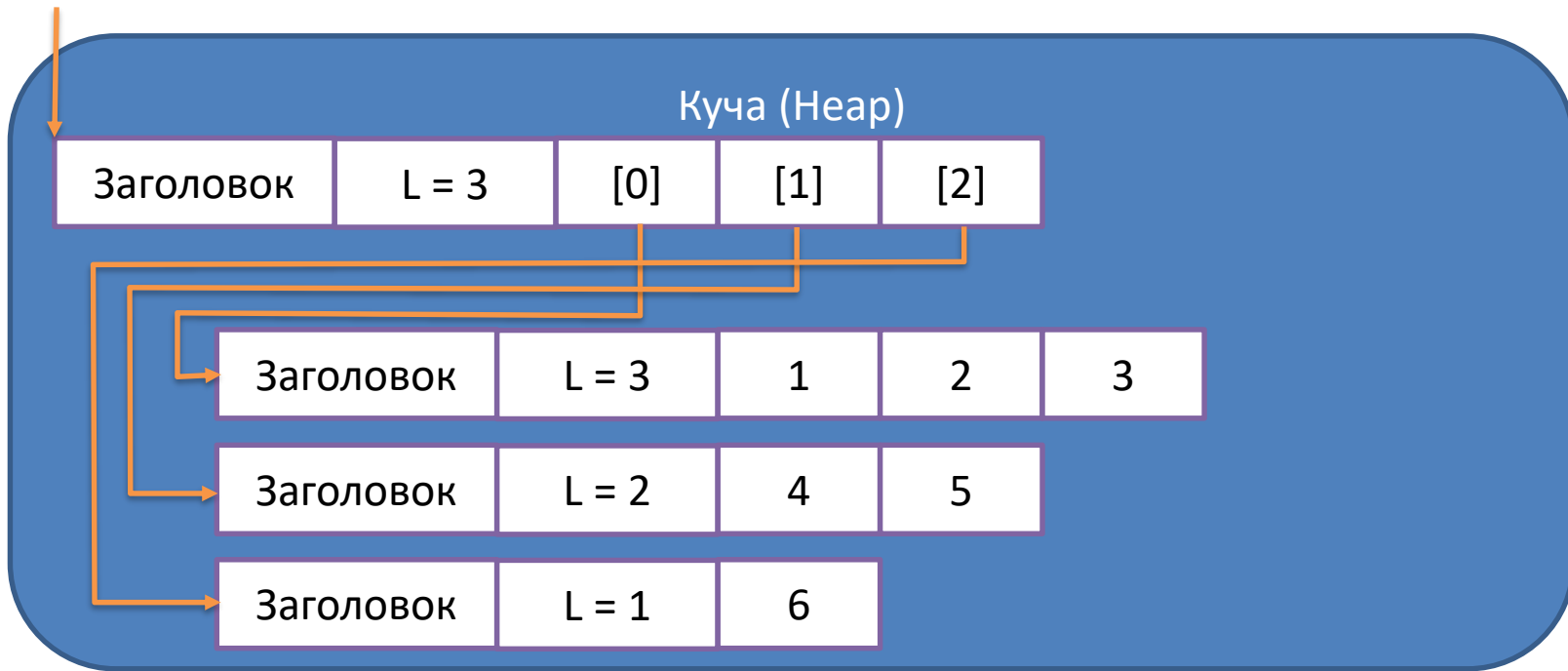


# Массивы: создание

```
int[] ints = new int[10];  
int[] initialized = new int[]{1, 2, 3, 4, 5};  
int[] simpler = {1, 2, 3, 4, 5};  
int[] empty = new int[0];  
int[][] table = new int[2][5];  
int[][] initializedTable = {{1,2,3},{4,5},{6}};
```

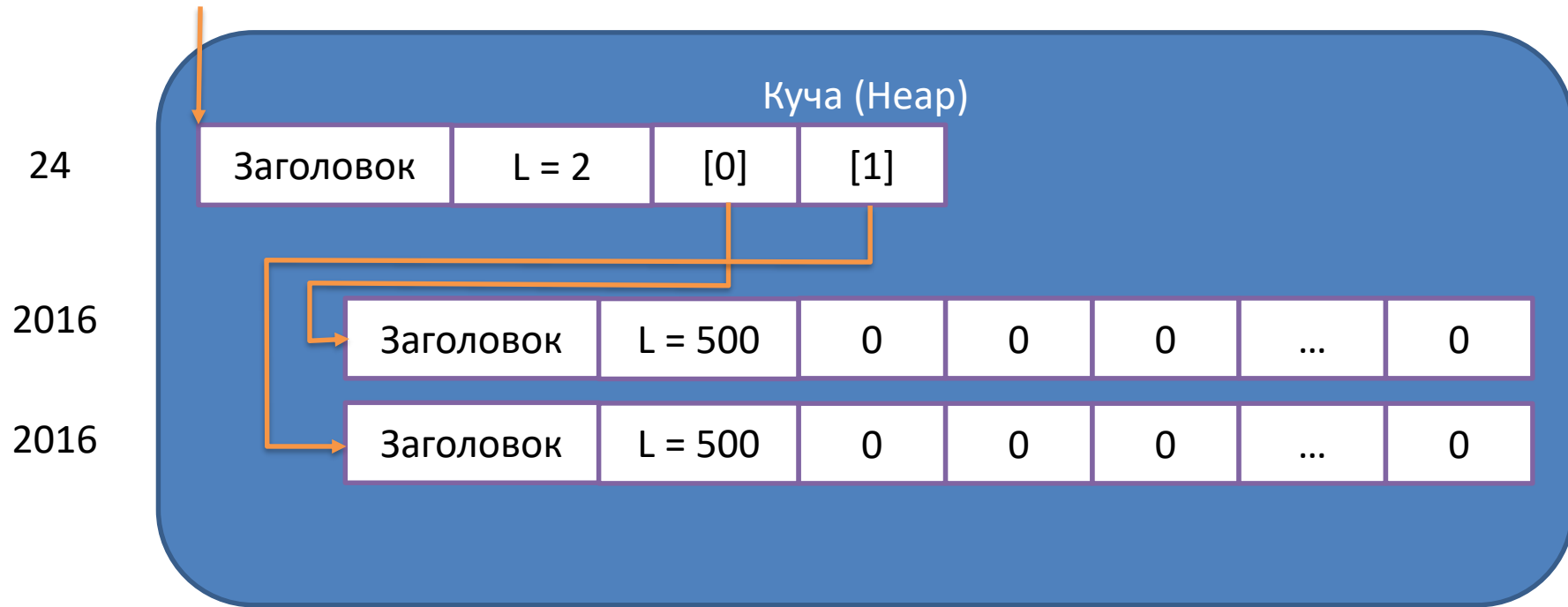
# Массивы

```
int[][] initializedTable = {{1, 2, 3}, {4, 5}, {6}};
```



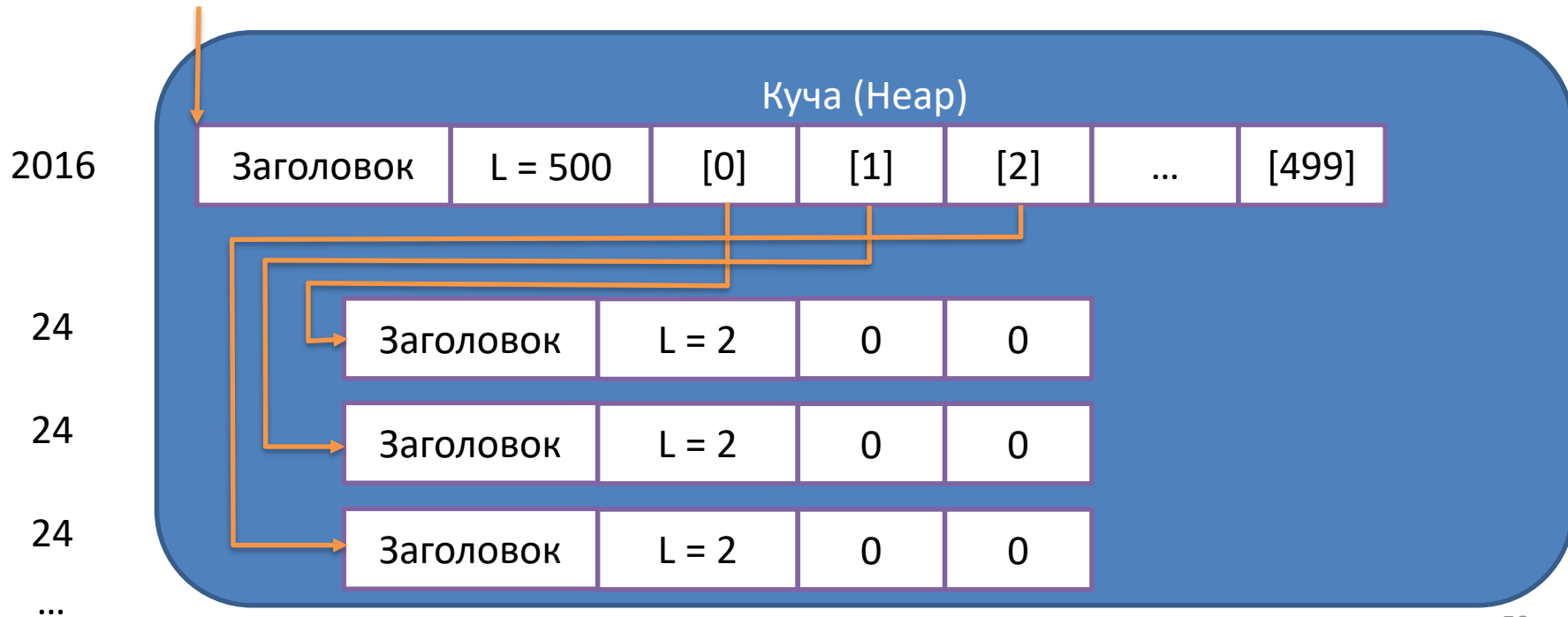
# Массивы

```
int[][] table = new int[2][500]; // 4056 байт, 1.4%
```



# Массивы

```
int[][] table = new int[2][500]; // 14016 байт, 350.4%
```



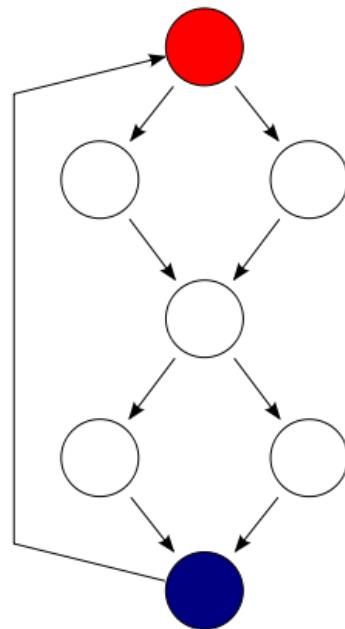
# Массивы: операции

```
int[] ints = {1, 2, 3, 2, 1};  
System.out.println(ints.length);  
System.out.println(ints[0]);  
ints[1] += 1;  
ints[10] = 10; // ArrayIndexOutOfBoundsException  
int[] copy = ints.clone();  
System.out.println(ints == copy); // false  
System.out.println(ints.equals(copy)); // false  
System.out.println(Arrays.equals(ints, copy)); // true  
System.out.println(ints); // [I@3cb5cdba  
System.out.println(Arrays.toString(ints)); // [1, 3, 3, 2, 1]  
Arrays.sort(ints);
```

java.util.Arrays – ваш друг!

# Управляющие конструкции

- ✓ if
- ✓ switch
- ✓ while
- ✓ do ... while
- ✓ for(;;)
- ✓ for-each
- ✓ break;
- ✓ continue;



# Предложение-блок

```
{  
    предложение  
    предложение  
    ...  
}
```

```
{  
    int x = 6;  
    System.out.println(x);  
}  
System.out.println(x);
```

# Пустое предложение





# if

**if** (условие) предложение  
[**else** предложение]

```
if(x > 0) {  
    System.out.println("positive");  
} else {  
    System.out.println("negative or zero");  
}
```

# if

**if** (условие) предложение  
**[else** предложение]

```
if (x > 0) {  
    System.out.println("positive");  
} else if (x == 0) {  
    System.out.println("zero");  
} else {  
    System.out.println("negative");  
}
```

# if

**if** (условие) предложение  
**[else** предложение]

```
if (a)
    if (b)
        System.out.println("a and b both true");
else
    System.out.println("??");
```

# Условный оператор

Тернарный оператор

```
System.out.println(x > 0 ? "positive" : "negative or zero");
```

# Предложение-выбор

switch statement

```
switch (x) {  
    case 1:  
        System.out.println("one");  
        break;  
    case 2:  
        System.out.println("two");  
        break;  
    default:  
        System.out.println("other");  
}
```

Метки: целые числа (включая char), строки, перечисления

# Циклы



<https://www.youtube.com/watch?v=RVeHxUVkW4w>

The Centrifuge Brain Project

# while

**while**(условие) предложение

```
int i = 0;
while (i < 10) {
    System.out.println(i);
    i++;
}
```

# while

**while**(условие) предложение

```
while (true) {  
    System.out.println("Catch me if you can!");  
}
```



# while

**while**(условие) предложение

```
System.out.println("Type 'exit' to exit");  
while(!System.console().readLine().contains("exit"));
```

# do..while

**do** предложение **while**(условие);

```
int i = 0;  
do {  
    System.out.println(i);  
    i++;  
} while (i < 10);
```

# for

**for**(инициализатор; условие; обновление)  
предложение

```
for(int i=0; i<10; i++) {  
    System.out.println(i);  
}
```

# for

**for**(инициализатор; условие; обновление)  
предложение

```
for (int i = 0, length = args.length; i < length; i++) {  
    String arg = args[i];  
    System.out.println(arg);  
}
```

# for

**for**(инициализатор; условие; обновление)  
предложение

```
for (int i = 0; i < args.length; i++) {  
    String arg = args[i];  
    System.out.println(arg);  
}
```

# Расширенный for

enhanced for, for-each

**for**(тип переменная: выражение)  
предложение

```
for (String arg : args) {  
    System.out.println(arg);  
}
```

# break;

```
public static void main(String[] args) {  
    boolean found = false;  
    for (String arg : args) {  
        if (arg.equals("Waldo")) {  
            found = true;  
            break;  
        }  
    }  
    if (found) {  
        System.out.println("Waldo is found!");  
    }  
}
```



# continue;

```
public static void main(String[] args) {  
    int goodArgs = 0;  
    for (String arg : args) {  
        if (arg.contains("bad")) {  
            continue;  
        }  
        goodArgs++;  
    }  
    System.out.println("Found " + goodArgs + " good arguments");  
}
```



# break LABEL;

```
int[][] matrix = getMatrix();  
OUTER:  
for(int[] row : matrix) {  
    for (int element : row) {  
        if (element < 0) {  
            System.out.println("Negative value found!");  
            break OUTER;  
        }  
    }  
}
```

# Итого

- ✓ Изучили из чего состоит метод
- ✓ Научились объявлять и присваивать переменные
- ✓ Поигрались с числами
- ✓ Познакомились с массивами
- ✓ Почувствовали власть над потоком управления