**Java**

**Инфраструктура**

<https://openjdk.org/>

Downloads: https://jdk.java.net/20/

https://www.oracle.com/java/

Downloads: [https://www.oracle.com/java/technologies/downloads/#jdk18-linux](https://www.oracle.com/java/technologies/downloads/" \l "jdk18-linux)

Установка переменных окружения:

linux:

export JAVA\_HOME=$JAVA\_HOME:/usr/lib/jvm/java-1.6.0-openjdk-i386/bin/java

export PATH=/usr/lib/jvm/jdk-19.0.2/bin:$PATH

**Sublime Text**

Текстовый редактор с подсветкой ключевых слов и специфических конструкций.

Downloads: https://www.sublimetext.com/3

**Структура программы на языке Java**

public class Program {

public static void main (String args[]) {

System.out.println("Java работает!");

}

}

Файл должен иметь имя класса и расширение .java  
  
Компиляция программы:

**javac** Program.java

Выполнение программы:

**java** Program

**Задание 0.**   
Установить Java, Git, Sublime Text, написать программу на языке Java, выводящую Ваши: Имя Фамилия Отчество, номер группы, название института, название университета.

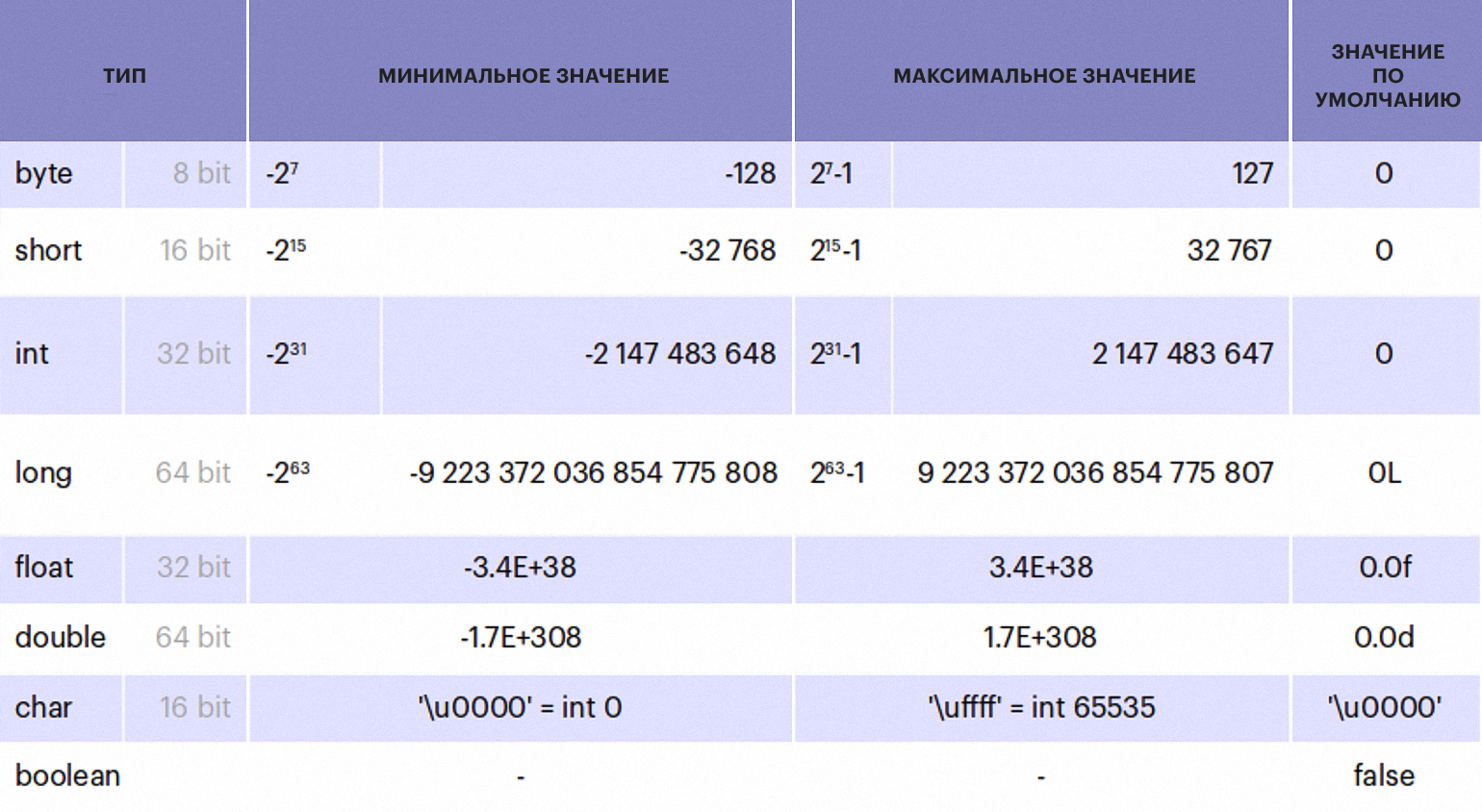
Скомпилировать программу и выполнить ее, используя средства командной строки.

**Идентификаторы в Java:** регистрозависимые, символы и арабские цифры 0—9, причем первым символом идентификатора не может быть цифра (чтобы избежать неопределенности, например, в записи 4e2 : это число 400,0 или имя переменной?)

В набор букв Java обязательно входят прописные и строчные латинские буквы, знак

доллара ( $ ) и знак подчеркивания ( \_ ), а также символы национальных алфавитов.

**Типы Java:** примитивные типы (primitive types) и ссылочные типы (reference types).



**Целые константы:**

- в десятичной форме: +5 , -7 , 12345678 ;

- в двоичной форме, начиная с нуля и латинской буквы b или B : 0b1001 , 0B11011 ;

- в восьмеричной форме, начиная с нуля: 027 , -0326 , 0777 (недопустимы цифры 8 и 9)

- в шестнадцатеричной форме, начиная с нуля и латинской буквы x или X : 0xff0a ,

0xFC2D , 0X45a8 , 0X77FF (здесь строчные и прописные буквы не различаются).

Для улучшения читаемости группы цифр в числе можно разделять знаком подчеркива-

ния: 1\_001\_234 , 0xFC\_2D .

Целые константы хранятся в оперативной памяти в формате типа int.

В конце целой константы можно записать латинскую букву "L" (прописную L или

строчную l ), тогда константа будет сохраняться в длинном формате типа long (см. да-

лее): +25L , -037l , 0xffL , 0XDFDFl .

**Вещественные константы:**

- с фиксированной точкой: 37.25 , -128.678967 , +27.035 ;

- с плавающей точкой: 2.5e34 , -0.345e-25 , 37.2E+4 ; можно писать строчную или про-

писную латинскую букву E ; пробелы и скобки недопустимы.

В конце действительной константы можно поставить букву F или f , тогда константа бу-

дет сохраняться в оперативной памяти в формате типа float (см. далее): 3.5f ,

-45.67F , 4.7e-5f . Можно приписать и букву D (или d ): 0.045D , -456.77889d , означающую

тип double , по умолчанию действительные константы хранятся в формате double .

**Символьные константы:**

- печатные символы, записанные на клавиатуре, просто записываются в апострофах

(одинарных кавычках): 'a' , 'N' , '?' ;

- управляющие и специальные символы записываются в апострофах с обратной на-

клонной чертой, чтобы отличить их от обычных символов:

'\n' — символ перевода строки LF (Line Feed) с кодом ASCII 10;

'\r' — символ возврата каретки CR (Carriage Return) с кодом 13;

'\f' — символ перевода страницы FF (Form Feed) с кодом 12;

'\b' — символ возврата на шаг BS (Backspace) с кодом 8;

'\t' — символ горизонтальной табуляции HT (Horizontal Tabulation) с кодом 9;

'\\' — обратная наклонная черта;

'\"' — кавычка;

'\'' — апостроф;

- код любого символа с десятичной кодировкой от 0 до 255 можно задать, записав его

не более чем тремя цифрами в восьмеричной системе счисления в апострофах после

обратной наклонной черты: '\123' — буква S , '\346' — буква Ж в кодировке CP1251.

Нет смысла использовать эту форму записи для печатных и управляющих символов,

перечисленных в предыдущем пункте, поскольку компилятор сразу же переведет

восьмеричную запись в указанную ранее форму. Наибольший восьмеричный код

'\377' — десятичное число 255;

- код любого символа в кодировке Unicode набирается в апострофах после обратной

наклонной черты и латинской буквы u четырьмя шестнадцатеричными цифрами:

буква S , '\u0416' — буква Ж .

Символы хранятся в формате типа char.

**Строковые константы:**

Заключены в двойные ковычки: "mail.ru"

Конкатенация строковых констант: "https://" + "mail.ru"

**Операции над целыми числами:**

**Арифметические:**

- сложение — + (плюс);

- вычитание — - (дефис);

- умножение — \* (звездочка);

- деление — / (наклонная черта, слэш); (**результат целый!** дробная часть отбрасывается)

- взятие остатка от деления (деление по модулю) — % (процент);

- инкремент (увеличение на единицу) — ++ ;

- декремент (уменьшение на единицу) — --

Результат арифметических операций int. Если хотя бы один из операндов long, то результат будет long

**В Java нет ошибки переполнения целого**

**Логический тип:**

**операции**

И - AND - &

ИЛИ — OR - |

ОТРИЦАНИЕ - NOT - ! (унарная)

ИСКЛЮЧАЮЩЕЕ ИЛИ - XOR - ^

**Сокращенные логические операции:**

var1 && var2

var1 || var2

**Задание 1.**

Вывести таблицы результатов логических операций над всеми возможными комбинациями операндов.

**Условные операторы:**

if ( выражение ) действие\_при\_условии;

if (выражение ) {

действие\_при\_условии;

...

}

if (выражение ) {

действие\_при\_условии;

} else {

}

if (выражение ) {

действие\_при\_условии;

} else if (выражение) {

} else {…}

**Операции сравнения:**

> — больше

>= — больше или равно

< — меньше

<= — меньше или равно

!= — не равно

== — равно

Важно! Результат операции сравнения имеет тип boolean.

boolean c = a > b

**Приведение типов**

Перед выполнением арифметических операций происходит повышение типа:

[char, byte, short]+[char, byte, short] → [int]+[int]

Если хотябы один из операторов имеет тип long, то тип повышается до long.

Повышение происходит добавлением нулевых битов.

Явное приведение типов

(type) var

при этом, происходят преобразования

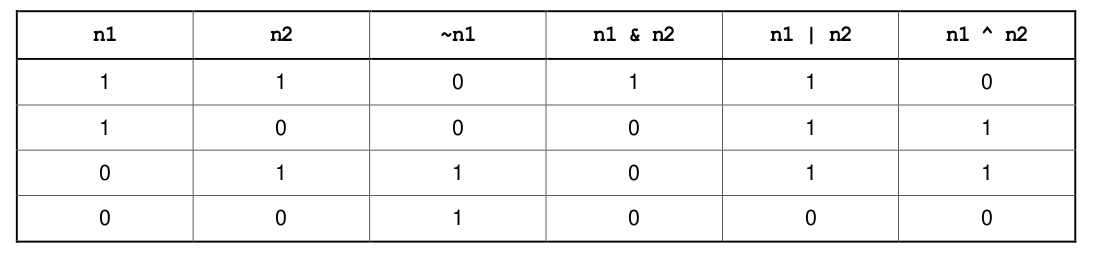
**Побитовые операции**

- дополнение (complement) — ~ (тильда);

- побитовая конъюнкция (bitwise AND) — & ;

- побитовая дизъюнкция (bitwise OR) — | ;

- побитовое исключающее ИЛИ (bitwise XOR) — ^ .

**Сдвиги:**

- сдвиг влево — << ;

- сдвиг вправо — >> ;

- беззнаковый сдвиг вправо — >>> .

беззнаковый сдвиг вправо для отрицательных чисел слева добавляет 0 в количестве позиций сдвига

сдвиг вправо для отрицательных чисел слева добавляет 1 в количестве позиций сдвига

**Приоритет операций**

Операции перечислены в порядке убывания приоритета. Операции на одной строке

имеют одинаковый приоритет.

1. Постфиксные операции ++ и -- .

2. Префиксные операции ++ и -- , дополнение ~ и отрицание ! .

3. Приведение типа (тип) .

4. Умножение \* , деление / и взятие остатка % .

5. Сложение + и вычитание - .

6. Сдвиги: << , >> , >>> .

7. Сравнения: > , < , >= , <= .

8. Сравнения: == , != .

9. Побитовая конъюнкция — & .

10. Побитовое исключающее ИЛИ — ^ .

11. Побитовая дизъюнкция — | .

12. Конъюнкция — && .

13. Дизъюнкция — || .

14. Условная операция — ?: .

15. Присваивания: = , += , -= , \*= , /= , %= , &= , ^= , |= , <<= , >>= , >>>= .

К обычным вещественным числам добавляются еще три значения:

- положительная бесконечность, выражаемая константой POSITIVE\_INFINITY и возни-

кающая при переполнении положительного значения, например в результате опера-

ции умножения 3.0\*6e307 или при делении на нуль;

- отрицательная бесконечность NEGATIVE\_INFINITY , возникающая при переполнении от-

рицательного значения, например в результате операции умножения -3.0\*6e307 или

при делении на нуль отрицательного числа;

- "не число", записываемое константой NaN (Not a Number) и возникающее, например,

при умножении нуля на бесконечность.

**Циклы**:

while (логВыр) {операторы;}

do {операторы;} while (логВыр)

int i = 0;  
  
while (i < 10) {  
 System.*out*.println(i++);  
}  
  
do {  
 System.*out*.println(i++);  
} while (i < 12);

1 .. 11

for (списокВыр1; логВыр; списокВыр2) {операторы;}

списокВыр1 — список выражений через запятую, выполняемых 1 раз,

можно определять переменные, используемые в теле цикла

логВыр — логическое выражение — условие выполнения следующей итерации

списокВыр2 - список выражений через запятую, выполняемых после каждой итерации

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

**Задание 2.**

int b1 = 10, c1 = 20, ind = 2, b2 = 5, big = 20;

1) (b1 + с1) % (++b2 / b1++) .

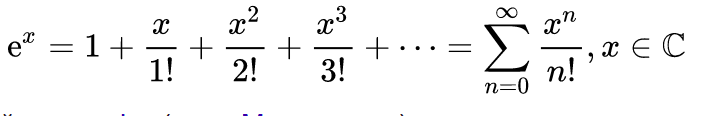
2) (b1 < с1) && (b2 == -99) || (ind >= 0) .

3) (b1 | с1) & (big ^ b1) .

4) (b1<<3 + с1<<2) % (b2>>5 / b1>>>2) .

**Задание 3.**

Вычислить exp(x)



с точностью до 0.00001

**Вывод**

Вывод в консоль:

System.out.print(*String*) — вывод строки в позицию курсора

System.out.println(*String*)— вывод строки в позицию курсора, с переводом курсора на начало новой строки

Аргументом выше перечисленных методов, является строка (точнее выражение со строковым результатом), выводимая в консоль.

System.out.printf(*formatString*, *args*, ….) — форматированный вывод в позицию курсора.

Пример: System.out.println("Значение = " + 2)

*formatString* — строка формата, содержащая некоторый текст и спец.символы форматирования.

*args* — список аргументов различных простых типов, подставляемых в строку форматирования.

Спец.символы: %d — подставляется целое число,

%15d — целое число займет 15 позиций, слева дополнится пробелами, если число менее 15 цифр

%015d — целое число займет 15 позиций, слева дополнится нулями, если число менее 15 цифр

%+d — вывод со знако числа

%o — вывод целого в восьмеричном формате

%x, %X — вывод целого в шеснадцатиричном формате

%a ,%A , %e , %E , %f , %g , %G — вывод вещественных чисел

%a ,%A — в шестнадцатиричном формате

%e , %E — в записи с экспонентой

%f — десятичная запись

%g , %G — универсальный вывод

можно указать точность после запятой: %n.mf — n — кол-во позиций под всё число, m — кол-во знаков после запятой

%с, %С — вывод символа

%s, %S — вывод строки

Примеры

Вывод целого числа:

System.out.printf("Значение %d + %d = %d",2,3,5);

> Значение 2 + 3 = 5

System.*out*.printf("Число PI %20.3f \n",Math.*PI*);

>Число PI 3,142

**Ввод**

Ввод данных с консоли:

Scanner in = new Scanner(System.in);

int num = in.nextInt();

**next()**: считывает введенную строку до первого пробела

**nextLine()**: считывает всю введенную строку

**nextInt()**: считывает введенное число int

**nextDouble()**: считывает введенное число double

**nextBoolean()**: считывает значение boolean

**nextByte()**: считывает введенное число byte

**nextFloat()**: считывает введенное число float

**nextShort()**: считывает введенное число short

**Массивы**

1. Объявление — (declaration)

Тип[] имяПеременной, например, int[] a; int a[];

2. Определение — (instantation)

имяПеременной = new Тип[длинаМассива], например a = new int[5];

3. Инициализация — (initialization)

имяПеременной[индекс] = значение;

имяПеременной = new Тип[] {значение1, … ,значениеN};

Длина массива:

имяПеременной.length

Строки

Класс String

Инициализация строк:

String str1 = "Java";

String str2 = new String(); // пустая строка

String str3 = new String(new char[] {'h', 'e', 'l', 'l', 'o'});

String str4 = new String(new char[]{'w', 'e', 'l', 'c', 'o', 'm', 'e'}, 3, 4);//3 -начальный индекс, 4 -кол-во символов

Длина строки

str.length()

**concat()**: объединяет строки

**valueOf()**: преобразует объект в строковый вид

**join()**: соединяет строки с учетом разделителя

**сompareTo()**: сравнивает две строки

**charAt()**: возвращает символ строки по индексу

**getChars()**: возвращает группу символов

**equals()**: сравнивает строки с учетом регистра

* **equalsIgnoreCase()**: сравнивает строки без учета регистра
* **regionMatches()**: сравнивает подстроки в строках
* **indexOf()**: находит индекс первого вхождения подстроки в строку
* **lastIndexOf()**: находит индекс последнего вхождения подстроки в строку
* **startsWith()**: определяет, начинается ли строка с подстроки
* **endsWith()**: определяет, заканчивается ли строка на определенную подстроку
* **replace()**: заменяет в строке одну подстроку на другую
* **trim()**: удаляет начальные и конечные пробелы
* **substring()**: возвращает подстроку, начиная с определенного индекса до конца или до определенного индекса
* **toLowerCase()**: переводит все символы строки в нижний регистр
* **toUpperCase()**: переводит все символы строки в верхний регистр

Классы обёртки

Integer — число соответствующего типа плюс набор методов.

Integer.parseInt(String)

Double

Игра "Ханойские башни". Есть 3 стержня, 2 из них пустые, а на первый надеты диски разного размера, большие ниже, меньшие выше, как на детской пирамидке. Задача - перенести все диски с 1-го стержня на 3-й, при этом за раз можно переносить только один диск с одного стержня на другой. Кроме того, нельзя класть большие стержни на меньшие. Составить программу для переноса произвольного числа дисков. Пользователь вводит число стержней N, программа перемещает стержни и печатает ходы, откуда куда что переносится.

Рекурсия — вызов метода внутри своего тела

f(x) {

…

f(y)

}

# Условные операторы

## Оператор варианта swithc

switch (выражение){

case констВыр1: оператор1

case констВыр2: оператор2

case констВырN: операторN

default: операторDef

}

выражение - должно иметь тип результата byte , short , int , char и их оболочки, String, Enum

констВыр - константа или константное выражение (не содержит переменных) типа byte , short , int , char и их оболочки, String, Enum

# Тернарный оператор

x = expr\_boolean ? expr1 : expr2

var x;

if (expr\_boolean) {

x = expr1;

} else {

x = exp2;

}

(булевое выражение) ? (выражение для истины) : (выражение для ложного утверждения)

Проверяется булевое выражение, если оно истинно, то выполняется выражение для истины и возвращается его значение, если проверяемое выражение ложно, то возвращается результат выражения для ложного утверждения

выражения expr1 и expr2 должны возвращать значение одного типа.

Пример: модуль числа

double abs(double x) { return x > 0 ? x : -x; }

**Регулярные выражения**

Основные задачи:  
1) Проверка удовлетворения строки заданному шаблону (корректность номера телефона, адреса почты, …);

2) Поиск подстроки, удовлетворяющей заданному шаблону.

Поиск совпадений границ слов или текста:

|  |  |
| --- | --- |
| Метасимвол | Назначение |
| ^ | начало строки |
| $ | конец строки |
| \b | граница слова |
| \B | не граница слова |
| \A | начало ввода |
| \G | конец предыдущего совпадения |
| \Z | конец ввода |
| \z | конец ввода |

Поиск символьных классов

|  |  |
| --- | --- |
| Метасимвол | Назначение |
| \d | цифровой символ |
| \D | нецифровой символ |
| \s | символ пробела |
| \S | непробельный символ |
| \w | буквенно-цифровой символ или знак подчёркивания |
| \W | любой символ, кроме буквенного, цифрового или знака подчёркивания |
| . | любой символ |

Группы символов

|  |  |
| --- | --- |
| Метасимвол | Назначение |
| [абв] | любой из перечисленных (а,б, или в) |
| [^абв] | любой, кроме перечисленных (не а,б, в) |
| [a-zA-Z] | слияние диапазонов (латинские символы от a до z без учета регистра ) |
| [a-d[m-p]] | объединение символов (от a до d и от m до p) |
| [a-z&&[def]] | пересечение символов (символы d,e,f) |
| [a-z&&[^bc]] | вычитание символов (символы a, d-z) |

Количество символов (квантификатор, описывается после группы или символа)

|  |  |
| --- | --- |
| Метасимвол | Назначение |
| ? | один или отсутствует |
| \* | ноль или более раз |
| + | один или более раз |
| {n} | n раз |
| {n,} | n раз и более |
| {n,m} | не менее n раз и не более m раз |

Пример:

"Мой дядя самых честных правил".matches("(.)\*(дядя)(.)\*") = true

(.)\* - любой символ в произвольном количестве

(дядя) — последовательность символов

public static void main(String[] args) {

String text = "Егор Алла Александр";

Pattern pattern = Pattern.compile("А.+а");

Matcher matcher = pattern.matcher(text);

while (matcher.find()) {

System.out.println(text.substring(matcher.start(), matcher.end()));

}

}

**GIT**

Система контроля версий.

Downloads: https://git-scm.com/downloads

Проверяем:  
git --version

настраиваем:

- вообще:

git config name value

- в частности:

git config user.name ksenikeev  
git config user.email [ksenikeev@mail.ru](mailto:ksenikeev@mail.ru)

создаем лоальный репозиторий в текущей директории:

git init

добавляем файл в список индексируемых:  
git add *filename*

коммит (фиксируем изменения индексируемых файлов):

git commit -m "*commit message*"

git push https://*user\_token*@github.com/ksenikeev/*repository*.git