

- Этапы разработки ПО
- Что такое алгоритм?
- Что такое пакет?
- Что такое main?
- Какая команда печатает текст на экране консоли?
- Что такое escape-последовательности?

- Прописных и строчных букв латинского алфавита (A-Z, a-z). Прописные и строчные буквы в коде различаются, это свойство называется **чувствительностью к регистру символов** (регистрозависимость).  
Примеры регистрозависимых языков: Java, C++, C#. Примеры регистронезависимых языков: HTML, SQL.

- Цифр от 0 до 9
- Пробельных символов (пробел, горизонтальная табуляция TAB, переход на следующую строку ENTER)
- Специальных символов: , . ; : \_ + - \* / % < > = ^ ? ! & | ~ ( ) { } [ ] @ “ ‘

**Лексема** – это наименьшая неделимая часть языка, которую распознает компилятор. Из лексем состояются все языковые конструкции.

- идентификаторы (identifiers)
- ключевые слова (keywords)
- литералы (literals)
- разделители (separators)
- операторы (operators)

**Идентификаторы** - это имена, которыми обозначаются различные объекты программы, определяемые программистом (переменные, методы, классы и тд.)

Идентификатор обязан быть уникальным. Может состоять из букв латинского алфавита, цифр, символа подчеркивания. Идентификатор не может начинаться с цифры!

Язык Java регистрозависим – большие и маленькие буквы в лексемах различаются:

Name

name

nAmE

- это совершенно разные лексеммы!





Ты чем  
три дня  
занималась?



Выбирала  
имя  
переменной



**Ключевые слова (keywords) - это зарезервированные, служебные слова, которые нельзя использовать в своих целях (например, идентификатор не может быть ключевым словом).**

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

\* зарезервированное слово, не используется

`true`  
`false`  
`null`

Не являются ключевыми словами (это литералы), но также НЕ могут быть использованы в качестве идентификаторов в вашей программе.

**Литерал** - это лексема, жёстко прописанная в коде программы, которая представляет собой фиксированное значение определённого типа.

### Целочисленные литералы

#### Знаковые:

- В десятичной системе (Dec):  
+5  
5  
-5

#### Беззнаковые:

- Восьмеричная система (Oct):  
025
- Шестнадцатеричная система (Hex):  
0x2D 0x2d 0X2D

### Вещественные (дробные) литералы

#### Запись с десятичной точкой:

1.25678  
1.  
.3287

#### Экспоненциальная запись:

2E-4  
1e5  
1E+5

#### Комбинированная запись:

2.6E-2

### Символьные литералы

#### Символ с клавиатуры:

'H'  
'\*'

#### Esc-последовательность:

'\n'  
'\b'  
'\"'

### Логические литералы

true – истина

false - ложь

Литералы позволяют задать в программе значения для числовых, символьных и строковых выражений, а также null-литералов. Всего в Java определены следующие виды литералов:

- целочисленный (integer);
- дробный (floating-point);
- булевский (boolean);
- символьный (character);
- строковый (string);
- null-литерал (null-literal).

**Оператор** – это конструкция языка программирования, которая выполняет определённое действие над аргументами (операндами).

**Операнд** - это аргумент оператора, то есть то значение, над которым оператор выполняет действие.



По количеству операндов операторы делят на:

- **Унарные** – требуют наличия 1 операнда:

-5

level++

- **Бинарные** – требуют 2 операнда:

3 \* 6

2 + 2

- **Тернарный** – состоит из трёх операндов:

int max = a > b ? a : b;

Примеры операторов:

+ - \* / = ++ -- >> <= ==

Операторы отличаются:

- Количеством операндов
- Приоритетом
- Ассоциативностью



[http://pr0java.blogspot.ru/2015/04/java\\_7.html](http://pr0java.blogspot.ru/2015/04/java_7.html)

[http://pr0java.blogspot.com/2015/04/java-2\\_24.html](http://pr0java.blogspot.com/2015/04/java-2_24.html)

- условные операторы (if, switch)
- операторы цикла (while, do while, for)
- операторы безусловного перехода (return, break, continue, throw)
- метки (case, default, user labels)
- операторы-выражения (любое выражение, которое заканчивается точкой с запятой, является оператором).
- операторы-операции (арифметические, логические, поразрядные и операции сравнения)
- блоки

Одни операторы ставятся перед операндами и называются **префиксными**, другие — после, их называют **постфиксными** операторами.

Большинство же операторов ставят между двумя операндами, такие операторы называются **инфиксными** бинарными операторами.

Разделители – это специальные символы, используемые в коде:

“ () ”, “ [] ”, “ {} ”, “ . ”, “ ” ”, “ ” ”, “ ” ”, “ ” ”

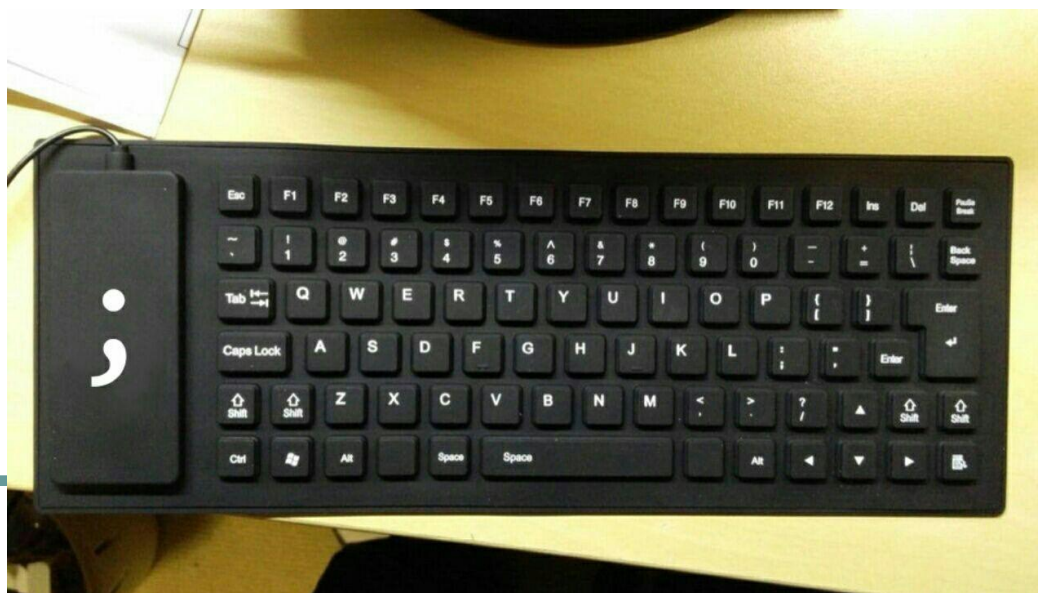
**Выражение** – это комбинация операндов (значений) и операторов, которая всегда имеет определённое результирующее значение. Это значение характеризуется типом данных. В выражении могут присутствовать переменные, литералы, результаты работы методов и тд. Пример выражения:

```
int result = 5 + x / 8 – (3 * number);
```

В результате этого выражения будет **значение** определённого типа данных, которое можно использовать в дальнейшем - например, присвоить переменной.

Почти любая команда (оператор) в языке Java заканчивается точкой с запятой.

```
System.out.println("Hello, world!");
```



Хотя бы в одном из классов должен существовать метод **main()**.

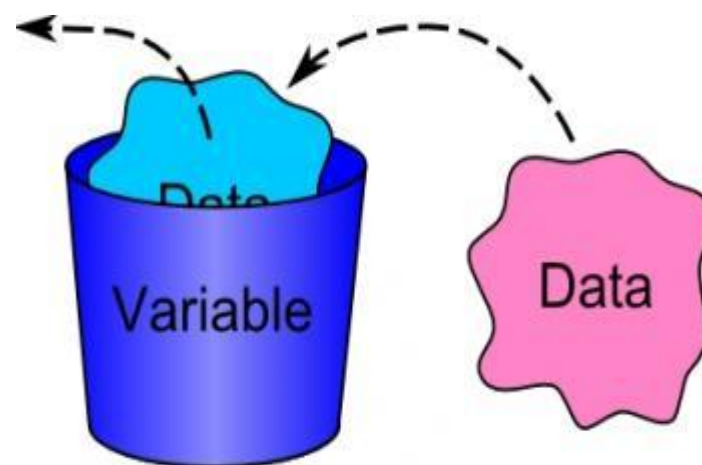
Этот метод вызывается автоматически!

В начале разбираться или пытаться запомнить правильное написание этого метода необязательно – NetBeans всё сгенерирует сам.

**Переменная** – это именованная область в оперативной памяти, которая может хранить и изменять своё значение на протяжении работы программы. Значение характеризуется типом данных. Для того, чтобы использовать переменную в программе, необходимо её **объявить** (создать). Объявление переменной даст компилятору понять, что для этой переменной необходимо выделить память, что данный идентификатор уже будет использоваться, и как с этой переменной работать в дальнейшем.



Переменные – это **своеобразные контейнеры**, которые могут нести в себе числовые, строковые или логические значения.



**Статическая типизация** – тип данных объекта определяется на этапе компиляции.

Если это происходит на этапе выполнения программы — то **динамическая**.

В Java используется статическая типизация, а это значит, что программисту придётся выбирать тип для переменной самостоятельно.

**тип** идентификатор = инициализатор;

**Инициализатор** – это выражение, которое вычисляется в этом месте программы. Им будет инициализирована переменная.

<https://habr.com/post/346214/>



- логический – **boolean**
- целые – **byte, short, int, long**
- вещественные – **float, double**
- символьный – **char**



**INT**

**DOUBLE**

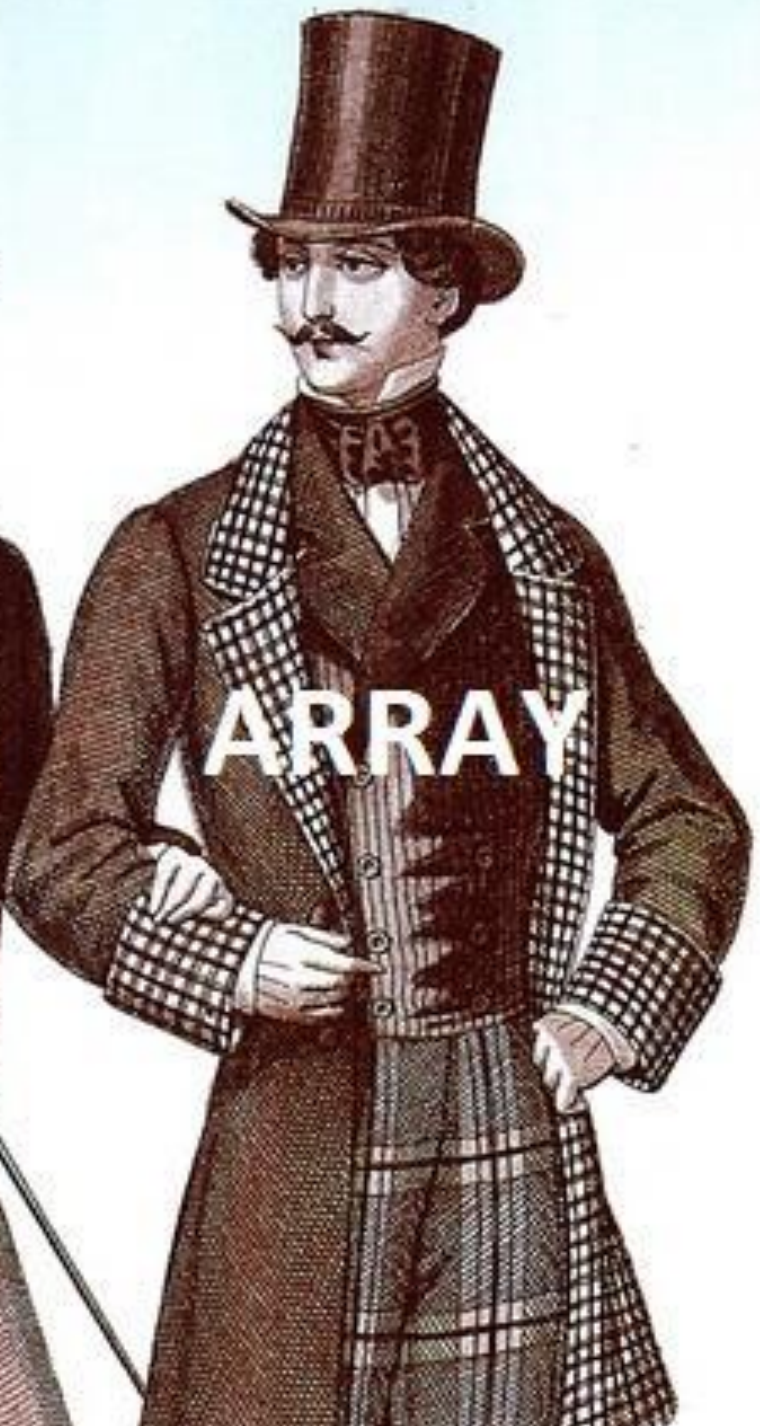
**CHAR**



**OBJECT**



**STRING**



**ARRAY**

**boolean** – **true** или **false**

**byte** (1 байт) – от -128 до 127

**char** (2 байта) – от 0 до 65535

**short** (2 байта) – от -32768 до 32767

**int** (4 байта) – -2 147 483 648 ...

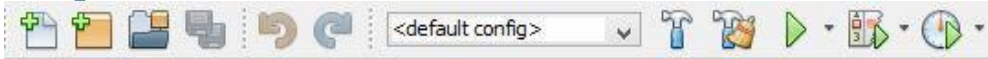
**long** (8 байт) – до  $9 \cdot 10^{18}$  (квинтиллионы)

**float** (4 байта) - 3.4E-38...3.4E38

**double** (8 байт) - 1.7E-308...1.7E308

По стандарту IEEE 754 представление действительных чисел должно записываться в экспоненциальном виде. Это значит, что часть битов кодирует собой **мантиссу** числа, другая часть — показатель **порядка** (степени), и ещё один бит используется для указания знака числа (0 — если число положительное, 1 — если число отрицательное).





Projects x Files Services

- JavaApplication1
- JavaApplication2
  - Source Packages
    - javaapplication2
      - JavaApplication2.java
  - Libraries

```
Start Page x JavaApplication2.java x
Source History
1 package javaapplication2;
2
3 public class JavaApplication2 {
4
5     public static void main(String[] args) {
6
7         System.out.println(0.1 + 0.1 + 0.1 - 0.3);
8
9     }
10 }
11
javaapplication2.JavaApplication2 > main >
```

Output - JavaApplication2 (run) x

```
run:
5.551115123125783E-17
BUILD SUCCESSFUL (total time: 0 seconds)
```

main - Navigator x

Members <empty>

- JavaApplication2
  - main(String[] args)

- **Размер блока памяти**, выделяемый для хранения данных
- **Структуру** этого блока памяти (как в машине будет сохранено, и как машина будет воспринимать данное значение - наличие или отсутствие знакового бита для целых чисел; наличие или отсутствие в числе битов для мантиссы, порядка и знака дробного числа)
- **Диапазон значений**
- **Набор операторов для работы** с этими значениями (например, для строк нельзя использовать оператор «минус», а для дробных чисел нельзя использовать битовые операции)

**тип** идентификатор;

**int** age;

**float** price;

**short** cats, dogs;

**char** answer, symbol;

**boolean** isHungry;

**тип** идентификатор = инициализатор;

**int** age = 35;

**float** price = 28.99f;

**short** cats = 3, dogs = 1;

**char** answer = 'b', symbol = 'x';

**boolean** isHungry = false;

**String** name = "Alex";

**Константа** — это именованная область оперативной памяти, способная сохранить данные, которые потом измениться не смогут.

Константа — это фактически та же переменная, но объявленная с ключевым словом (модификатором) **final**. Также под понятие константы можно подвести понятие **литерала**.

**x = 3;**

**y = x;**

**z = x;**

многократное использование операции присваивания в одном выражении:

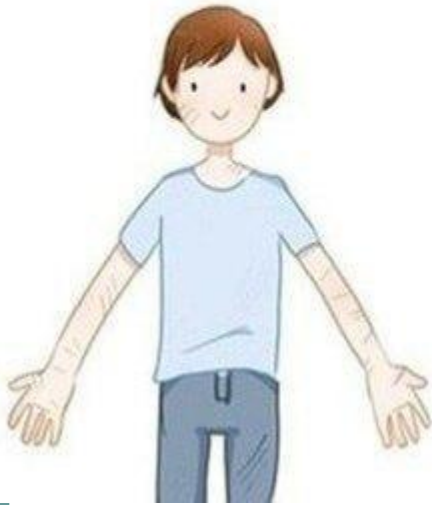
**x = y = z = 0;**

Не рекомендуется так делать!

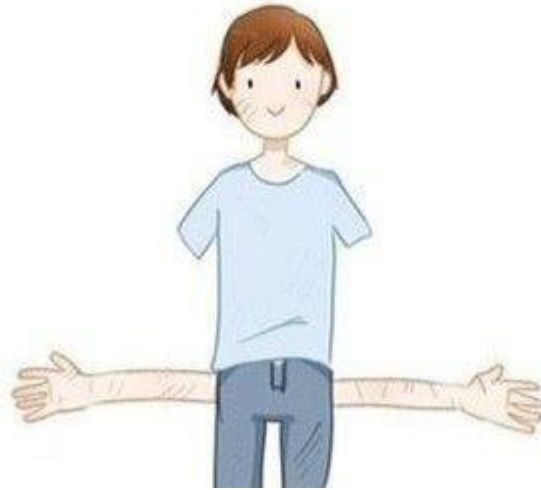
- **инкремент "++"** – увеличивает значение переменной на 1
- **декремент "--"** – уменьшает значение переменной на 1

Для этих операторов существует префиксная и постфиксная форма (практика).

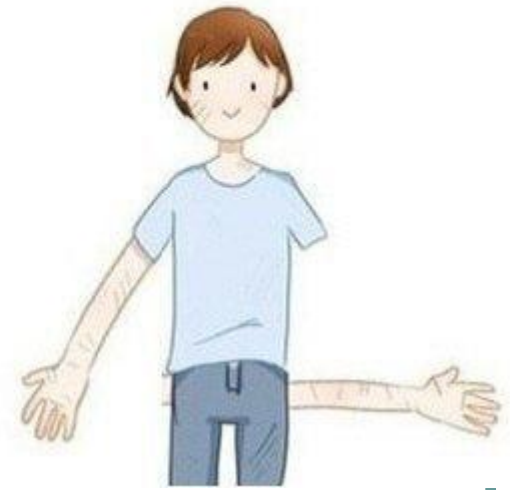
`i++`



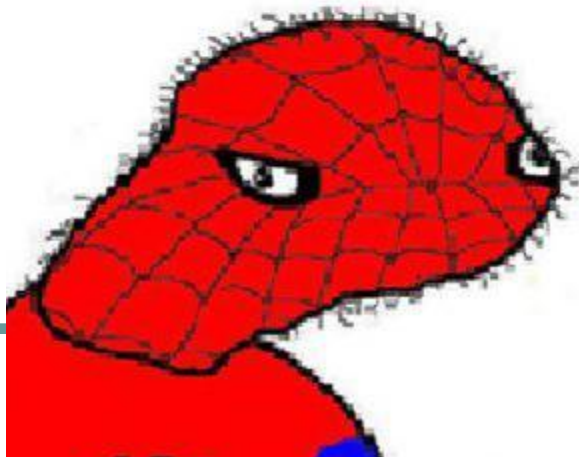
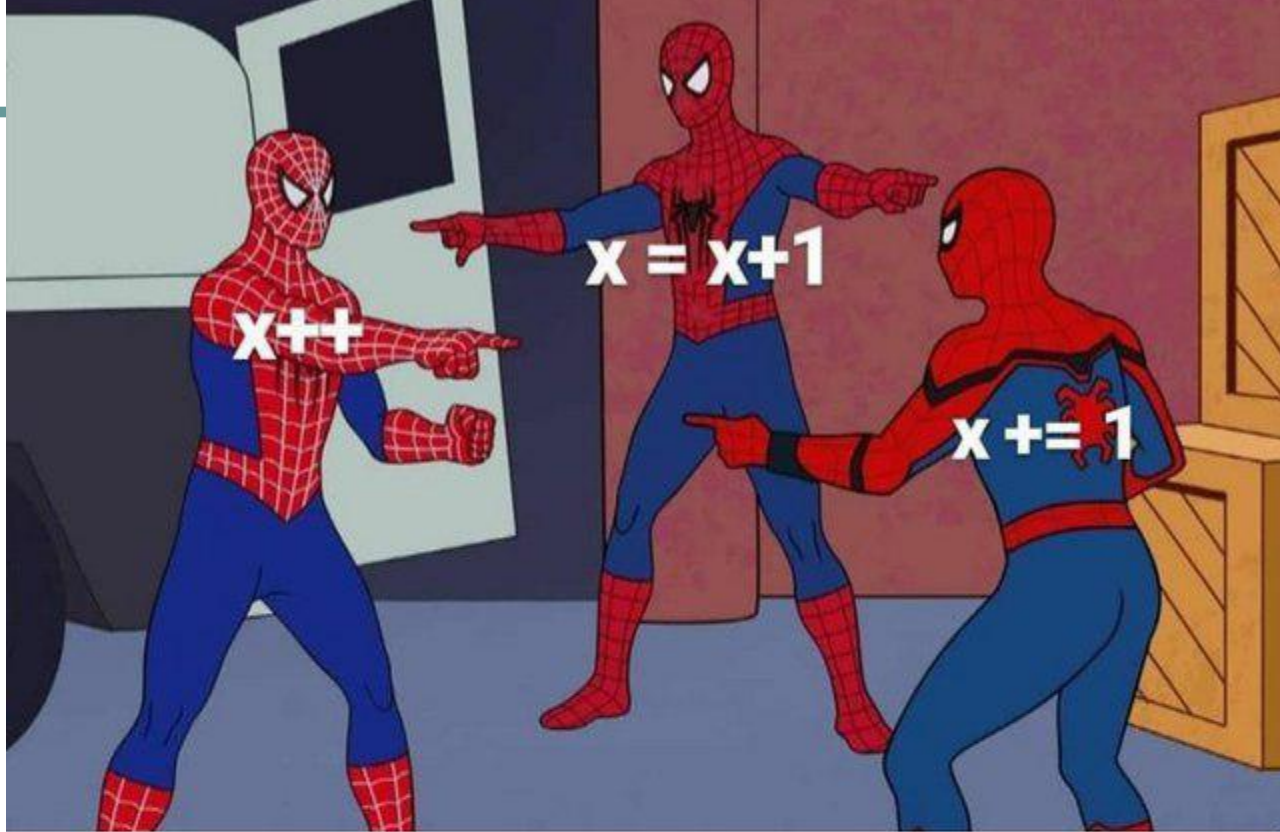
`i=i+1`



`i+=1`







$x -= -1$

- +** сложение
- вычитание
- \*** умножение
- /** деление
- %** остаток от деления
- ++** инкремент
- декремент



```
int a = 5 + 5;  
a = b * c;  
b++;  
a = 10 % 2;
```

## **Единственность цели каждой переменной**

Используйте переменную только с одной целью. Иногда есть соблазн вызвать одну переменную в двух разных местах для решения двух разных задач. Использование "временной" переменной - очень плохая затея.

Из-за использования в нескольких разных ситуациях одной переменной создается впечатление, что задачи связаны, хотя на самом деле это не так. Создавайте уникальные переменные для каждой цели, чтобы сделать код более читабельным и понятным.

У моей соседки **N** кошек. Каждая кошка за день съедает **M** граммов кошачьего корма. Килограмм корма стоит **G** гривен. Сколько денег уходит на кошачий корм в месяц, и за целый год?

