## Закръгляне на числа/видове променливи/прости операции/кастване/цикли

%n – на нов ред

\r\n

System.***out***.printf(**"%.3f%n"**, result); три знака след десетичната запетая

System.oot.println(String.format(**"%.3f%n"**, result))

System.***out***.printf(**"%.0f"**, result); изписва без десетичната запетая

System.***out***.printf(**"%d:%02d"**, finalHour, finalMinutes); - изписва с 1 нула пред числото ако то е едноцифрено

System.***out***.printf(**"%.2f%%%n"**, p1Percents); - двоен процент прави процента

String name = String.format(**"%.3f%n"**, result);

**new** DecimalFormat(**"0.####"**).format(5,33437) – нулата определя нула/число със сигурност, # определя че може да има цифра до 4 знака ако числото е толкова дробно или по-малко от 4ри знака

**“%,d”** – thousand separator или 1,000

Placeholders

%s или %S (String);

%d (int);

%f (double);

%c (char);

%b или %B (boolean) …

Default стойности за реално число:

Is **0.0F** for the **float** type

Is **0.0D** for the **double** type – по подразбиране реално число e double

Floating-point types are:

* + - **float** (±1.5 × 10−45 to ±3.4 × 1038) - 32-bits, precision of 7 digits
    - **double** (±5.0 × 10−324 to ±1.7 × 10308) - 64-bits, precision of 15-16 digits

Default стойности за цяло число:

long number = 1**L**;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **Default Value** | **Min Value** | **Max Value** | **Size** |
| **byte** | 0 | -128 (-27) | 127 (27-1) | 8 bit |
| **short** | 0 | -32768 (-215) | 32767 (215 - 1) | 16 bit |
| **int** | 0 | -2147483648 (-231) | 2147483647 (231 – 1) | 32 bit |
| **long** | 0 | -9223372036854775808  (-263) | 9223372036854775807  (263-1) | 64 bit |

**Very high precision is BigDecimal**

BigDecimal number = new BigDecimal(0);

BigDecimal num = new BigDecimal(sc.nextLine()); **или стринг**

number = number.add(BigDecimal.valueOf(2.5));

number = number.subtract(BigDecimal.valueOf(1.5));

number = number.multiply(BigDecimal.valueOf(2));

number = number.divide(BigDecimal.valueOf(2));

**Има също и BIGInteger**

BigInteger number = new BigInteger();

BigInteger C = A.add(BigInteger.valueOf(val));

B = new BigInteger(“123456789123456789”);

if (a < b) {} // For primitive int

if (A.compareTo(B) < 0) {} // For BigInteger

**int** a = 1;  
**double** b = 2.4, c = 2.4;  
  
option 1) a = a + (**int**)(b);  
option 2) a+= b; *// a = (int)(a + b);*

**char** symbol = sc.nextLine().charAt(0);

String architectName = sc.nextLine();  
**int** numberOfProjects = Integer.*parseInt*(sc.nextLine());

**double** percents = Double.*parseDouble*(scanner.nextLine());

7 / 3 - ако е Integer връща цяло число 2

Math.round(45.67852); //46 – **хвърля int**

Math.ceil(23.45); // 24.0 - хвърля double като изчиства 24,0000000000000000003

Math.floor(45.67); //45.0 - хвърля double като изчиства 45,0000000000000000003

Math.*pow*(2, i); // хвърля Double 2 на коя степен

Кастване – от по-малък в по-голям тип данни – не губим данни (Имплицитно):

double a = 4;

int a = (int)5.66; - връща 5 – експлицитно, и губим данни

Math.PI

**int** maxNumber = Integer.***MIN\_VALUE***;

**int** nMin = Integer.***MAX\_VALUE;***

String a = scanner.nextLine();

String b = scanner.nextLine();

System.out.println(a.**equals**(b)); //True

**!**a.equals(b); - отрицание

System.out.println(a.**equalsIgnoreCase**(b)); //True

**boolean greaterAB = (a > b);**

&& - конюнкция

II – дизюнкция

! – отрицание (!F = T)

!= различно

++a – първо променям стойността и след това използвам променливата на същия ред

а++ - първо се изпълнява на съшия ред със старата стойност, и след това ще се смени стойността на а.

Integer.parseInt(sc.nextLine());

sc.nextInt() – чете на същ или нов ред, ако имаме вход от повече от 1 елемент

В цикъл:

Break; - прекъсва текущия цикъл

Continue; - праща цикъла в следващо завъртане, без да изпълнява останалото текущо тяло на цикъла

Унарни операции – 1 аргумент е нужен

Бинарни операции – два аргумента са нужни

Тернарен оператор – 3 аргумента са нужни

## **Базова работа със String и Char**

Стрингове:

String destination = **""**; //празен стринг

Integer.*parseInt*(**""** + num.charAt(i)) чар плюс стринг прави стринг

String toRepeat = “abc”;

String temp = temp.concat(toRepeat);

String text = scanner.nextLine(); // въвеждаме SoftUni

int length = text.length(); // 7 za string

String current = expression.substring(2, i+1);

От число към String

*String newNumber = number + "";  
String newNumber = Integer.toString(2);*

Integer.*parseInt*(**""** + CharSymbol or number); - правим го на String  
Character.getNumericValue(CharSymbol); - правим го на String

String currentNumber = **""** + i;  
String currentnumber = String.*valueOf*(i);

Махаме всички удивителни от стринга, и го разделяме на масив

String currentInput = sc.nextLine().replaceAll(**"!+"**, **""**);

currentArr = currentInput.split(**""**);

System.***out***.println(Character.*isDigit*(**'3'**)); - връща true – проверка дали даден символ е число

boolean a = Character.*isDigit*(input[i].charAt(0));

**"textbrat"**.toUpperCase();

**if** (a.compareTo(b) >0) – a и b са Strings / низове

String output = input.charAt(input.length() / 2 -1) + **""**+ input.charAt(input.length()/2); - ако първо съберем символите, то ще получим число, а не конкатенация на String

split(“ ”) – един Space

split(“\**\s+**”) – повече Space/Tab/Enter/Return (white space) – пишем \\ за да обозначим само една наклонена \

Понякога, когато използваме **\s+** се получава, че един от разделения елемент и е празен Space “”. Него премахваме с result.remove("");

sc.nextLine().split(**"\\s+"**, 2); - докато види първият WhiteSpace e първата част, останалата част е втората част

.trim() – маха Space символите отпред и отзад на стринга – ако има такива Space-ве.

**Whitespace** – Space,Tab, Enter, Return

## Бързи команди в средата IntelliJ

Alt + Enter – мулти тул в Intelli J / средата за разработка Intellij предлага варианти / замества If със Switch / предлага създаване на метод по Дедукционния метод – от общото към частното

Ctrl + / - слага коментар

Ctrl + Alt + L – форматира автоматично

Ctrl + Shift + V – последните наши копирания в IntelliJ

Ctrl + D – копира същото на нов ред

Shift + F6 - Refactor - сменя името на една променлива навсякъде в програмата

Debbugger – с F8 минаваме на всеки ред, слагаме breakpoint (червена точка), с F9 отиваме до следващ breakpoint (червена точка)

Ctrl + B (Ctrl + click) – къде се използва този метод в кода ИЛИ Go To –> Declaration and Usages

Alt + мишката + стрелки нагоре / надолу – пишем на няколко реда едновременно

Ctrl + Shift + стрелки нагоре надолу – местя маркираното

Ctrl+C – копирам целият ред

Ctrl + X – Cut-ва целият ред

Ctrl + V – Поставя целият ред

Ctrl + P – дава инфо за get / set параметрите при списъци

Fori плюс Tab и ни задава цялото тяло на for цикъла

Alt+Enter и след това Iterate – създава FOREACH цикъл с тази колекция/масив/списък

Ctrl+Alt+M – Extract method – след селекция на даден код, да го сложим в метод този код

Ctrl+Shift+Enter – слага къдрави скоби където е пропуснато

Iter – Прави For each цикъл автоматично

Ctrl + Q – задава ни какво да въвеждаме в скобите

Fn + F8 – при дебъгване

Ctrl + C – копира целият ред

Snippets - Live Template – при маркиране на текст, за бърз достъп след това (от Tools):

Psvm – **public static void** main(String[] args)

Sout – System.***out***.println(**"January"**);

Sc - Scanner sc = **new** Scanner(System.***in***);

## **Масиви / Arrays**

### **4.1. Четене на масив**

**int**[] numbers = **new int**[5]; -задаваме дължина 5 елемента на масива

**int**[] numbers = **new int**[] {1, 2, 3};

String[] day = **new** String[7];

String[] numberString = **new** String[] {**"13"**, **"42"**, **"69"**};

String[] numberStrings = sc.nextLine().split(**" "**); - Четене на ред със стрингове

String[] parts = sc.nextLine().split(**"\\s+"**); - четене на ред от стрингове когато има повече Space разстояние между символите

Четене на масив от числа:

**int**[] numbers = Arrays.*stream*(sc.nextLine().split(**" "**)).mapToInt(e -> Integer.*parseInt*(e)).toArray();

**int**[] numbers = Arrays.*stream*(sc.nextLine().split(**" "**)).mapToInt(Integer::*parseInt*).toArray();

- четене на ред от числа

Четене на масив от числа по дългия начин:

String[] input = sc.nextLine().split(**" "**);  
**int**[] numbers = **new int**[input.**length**];  
**for** (**int** i = 0; i < numbers.**length**; i++) {  
 numbers[i] = Integer.*parseInt*(input[i]);  
}

**char**[] input = sc.nextLine().toCharArray();

### **4.2. Операции с масиви**

Размяна на елементи

**for** (**int** i = 0; i < numbers.**length** / 2 ; i++) {  
 **int** oppositeIndex = numbers.**length** - i -1;  
 **int** oldNumbersI = numbers[i];  
 numbers[i] = numbers[oppositeIndex];  
 numbers[oppositeIndex] = oldNumbersI;  
}

Изменяме даден масив по дължина и стойност

**int**[] num = **new int**[8];

**int**[] condensed = **new int**[5];

**int**[] num = Arrays.*copyOf*(condensed, condensed.**length**); - копираме масив с по-къса дължина от началния – създава се нов масив, който сочи към различно място в паметта

num = condensed; - num е референция към стойността на същия масив condensed 😊

arR = Arrays.*copyOfRange*(arR, index+1, length); - копираме масив след кой индекс и с каква дължина и го присвояваме след това в същият

Конкатениране на 2 масива

String[] both = ArrayUtils.addAll(first, second);

**ИЛИ**

**private static** String[] exchangeArr(String[] arR, **int** index) {  
  
 String[] a = Arrays.*copyOf*(arR, index+1);   
  
 **int** length = arR.**length**;  
 String[] b = Arrays.*copyOfRange*(arR, index+1, length);  
  
  
 String[] both = Stream.*concat*(Arrays.*stream*(b), Arrays.*stream*(a))  
 .toArray(String[]::**new**);  
  
 **return** both;  
  
}

**int**[] numbers = **new int**[8]; **int**[] copyNumbers = numbers; - копираме масив

Подмяна на всички елементи

String[] namesDevils = sc.nextLine().split(**","**);  
**for** (**int** i = 0; i < namesDevils.**length**; i++) {  
 namesDevils[i] = namesDevils[i].replaceAll(**" "**, **""**);  
}

Сортиране

Arrays.*sort*(sumCodes);

### **4.3. Печат/Изход на масив**

String[] numberString = **new** String[] {**"13"**, **"42"**, **"69"**};  
String joined = String.*join*(**"and"**, numberString); - обединява в 1 стринг ***масив от стрингове***

Int[] numbers;

System.***out***.println(numbers); - **отпечатва шлокавица за разлика от списъците, където разпечатва** [6, 6, 3]

#### **4.3.1. FOREACH цикъл - масиви /списъци/колекции/ – read-only е цикъла**

Използва се за обхождане на масиви / листи / колекции

**for** (**int** number: numbers) {  
 System.***out***.println(number);  
}

**for** (**int** i = 0; i < numbers.**length**; i++) {  
 **int** number = numbers[i];  
 System.***out***.println(number);  
}

## **Методи / Methods**

Методите се намират в обект Клас. Когато методите се намират извън обект клас, те се наричат функции. В eзиците Java и C# няма функции, а само методи – т.е. всички подобни изчисления сa в даден клас под формата на Метод/и.

Return; – приключва изпълнението на дадения метод

Методите в Java са **camelCase**

Identation - е разстоянието от началото на лявата страна.

Private – използва се метода само в текущия клас

Public – викаме метод от друг клас в нашия клас

**static int**[] readNextArray(Scanner sc) – връща масив

**static double** mathPower(**double** number, **int** power) – връща число Double

**static void** printInWords(**double** grade) – **връща команда/процес**

**static int** getMax (**int** a, **int** b) – връща число int

**static** String repeatString(String s, **int** repeatCount) – връща стринг

Единствено масиви и обекти и колекции, при използването им в метод променят референтната си стойност в RAM паметта, т.е. реално се променя даден елемент от масива.

Всички останали типове примитивни данни като

* int, float, double, char, Boolean – стойностен тип данни и
* String – референтен тип данни, но го считаме като стойностен тип тъй като не можем да му сменим съдържанието на String-a.

използвани в даден метод, то метода работи с тяхно копие, и оригиналната им стойност не се променя!!!

Сигнатура на даден метод се състои от Име на метода и от параметри на метода.

Типа на метода / на връщаните данни – void, String, int, double, не е част от неговата сигнатура!

**Over-loading методи**: за да е наличен оver-load метод, то метода от един и същи тип (на връщаните данни) и с едно и също име, то неговите параметрите трябва да се различават по поне един от три критерия:

* **Брой параметри**
* **Реда на параметрите**
* **Типа на параметрите**

Или с други думи казано, не може да има два метода с едно и също име, и едни и същи брой, ред и тип параметри, които да връщат различен тип данни/резултат – само 1 такъв метод трябва да съществува и да връща само един тип данни/резултат.

**static int** getMax(**int** a, **int** b) {  
 **if** (a > b) {  
 **return** a;  
 }  
 **return** b;  
}

**Рекурсия единична:**  
**static int** getMax(**int** a, **int** b, **int** c) {  
 **return** *getMax*(*getMax*(a, b), c);  
}

**static int** getMax(Stringa, int b){

}

**static int** getMax(int b, Stringa){

}

--------------------------

**Използване на скенер в метод**

**public static void** main(String[] args) {  
 Scanner sc = **new** Scanner(System.***in***);  
  
 *inBetween*(sc);  
}  
  
**private static void** inBetween(Scanner sc) {  
 **int** firstSymbol = sc.nextLine().charAt(0);  
 **int** secondSymbol = sc.nextLine().charAt(0);  
  
 **for** (**int** i = firstSymbol + 1; i <= secondSymbol -1 ; i++) {  
 System.***out***.print((**char**)i + **" "**);  
 }  
  
}

### **5.1. Масив и метод**

Използването на масив в метод – за да можем да вземем новополучения масив, то трябва да го пишем така:

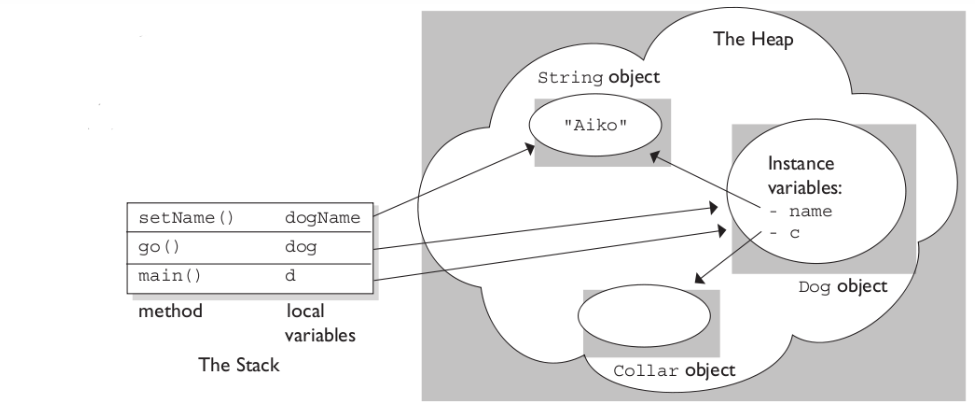
**private static int**[] firstNElements(**int**[] numberArr, **int** count,String evenOrOdds) {  
 **int**[] temp = **new int**[count];  
………………  
 **return** temp;  
}

### **5.2. STACK and HEAP - both are in RAM:**

STACK - static memory allocation – заделя се при стартирането/компилирането на програмата

HEAP - dynamic memory allocation – определя се / изменя се по време на изпълнение на програмата

Variables allocated on the heap have their memory allocated at run time and accessing this memory is a bit slower, but the heap size is only limited by the size of virtual memory . Element of the heap have no dependencies with each other and can always be accessed randomly at any time. You can allocate a block at any time and free it at any time. This makes it much more complex to keep track of which parts of the heap are allocated or free at any given time.



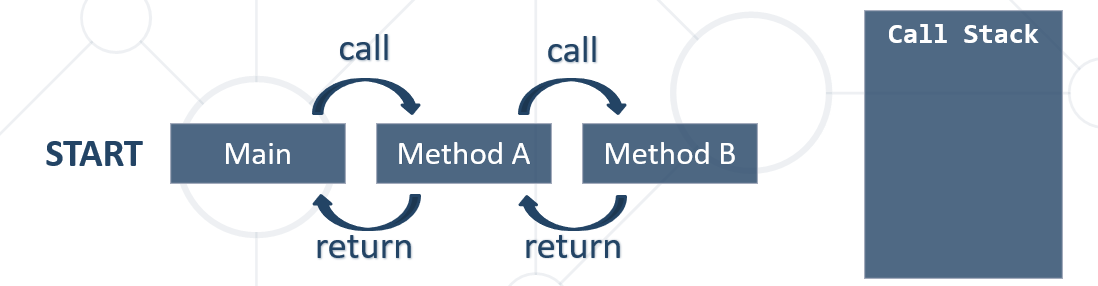
### **5.3. Debugging when Methods**

Stack Frames показва на кое място (първо, второ) в момента на дебъгването/изпълнението на кода се изпълнява даден метод. Комбинацията от методи влизат в състава на Stack-а.

Чрез Frames можем да проследим/проверим по-качествено състоянието на целия стек Stack в даден момент. Ако програмата ни се чупи в даден метод /**Frame**/, то е твърде вероятно грешката да идва от предходнo изпълнения Frame, до който ние имаме достъп!

Един Stack /програма се изпълнява, докато метода main не приключи!

The stack is always reserved in a LIFO order, the most recently reserved block is always the next block to be freed. This makes it really simple to keep track of the stack



## **Списъци / Lists**

Без определен брой елементи. Масивите са конкретен вид списъци – с конкретен брой елементи

Списъците работят по-бавно от масивите!

### **6.1. Четене на лист**

List<Integer> inputNumbers = **new** ArrayList<Integer>();

List<Integer> inputNumbers = **new** ArrayList<Integer>(Arrays.*asList*(1, 5));

ИЛИ

inputNumbers.add(1); inputNumbers.add(5);

Convert a collection into List

List<String> items = Arrays.*stream*(values.split(**" "**)).collect(Collectors.*toList*());

Converts an array into list

String[] data = sc.nextLine().split(**", "**);   
ArrayList<String> racers = **new** ArrayList<>(Arrays.*asList*(data));

List<Double> items = Arrays.stream(**values**.split(" ")).map(Double::parseDouble).collect(Collectors.toList());

List<Integer> firstList = Arrays.*stream*(sc.nextLine().split(**" "**)).map(Integer::*parseInt*).collect(Collectors.*toList*());

**values** e sc.nextLine()

List<Integer> list = **new** ArrayList<>();  
**for** (**int** i = 0; i < n; i++) {  
 **int** number = Integer.*parseInt*(sc.nextLine());  
 list.add(number);  
}

### **6.2. Команди**

**size()** – number of elements in the List<E>

**add(element)** – adds an element to the List<E>

**add(index, element)** – inserts an element to given position

**remove**(object) – removes an element (returns true / false) - ако елементите са тип инт, то **Integer.valueOf()**

**remove**(index) – removes element at index

**contains**(element) – determines whether an element is in the list - ако елементите са тип инт, то **Integer.valueOf()**

**set**(index, item) – replaces the element at the given index

initialList.**addAll**(лист от елементи additionalList) ; - допълнителния лист може да го извикаме с метод от тип List<int>

**int** bombIndex = items.**indexOf**(bombValue); - връща първият индекс, където се среща стойността bombValue в списъка.

Важно – когато работим с **Remove**, винаги е по-добре да премахнем елемента и запишем във времененна променлива; и чак след това да правим други операции със списъка

Също така гледаме дали работим с числа – ако не работим, то можем да използваме String

**while** (number != -1) {  
 inputNumbers.**add**(number);  
 number = sc.nextInt();  
}

**for** (**int** i = inputNumbers.size() - 1; i >= 0; i--) {  
 System.***out***.print(inputNumbers.**get**(i) + **" "**);  
}

**for** (Integer element : inputNumbers) {  
 System.***out***.print(element + **" "**);  
}

inputNumbers.**add**(2, 42); - добавяме нов елемент със стойност 42 на позиция индекс 2, а останалите елементи отиват с една позиция надясно

inputNumbers.**set**(1, 42); - променяме стойността на елемент – на индекс 1 елемента го променяме на стойност 42

inputNumbers.**remove**(1); - с индекс 1 елемента се маха, и останалите елементи се придвижват с един наляво и общия размер на списъка се намаля с 1 елемент.

Премахване по индекс или премахване по стойност при елементи от **int**

nums.remove(Integer.*valueOf*(40)); - премахва елемента/обекта със стойност 40

nums.remove(1); - премахва елемента с индекс 1

List<Integer> items = **new** ArrayList<>();

List<Integer> nums = **new** ArrayList<>();  
**for** (**int** i = 0; i < items.size(); i++)  
 nums.add(Integer.*parseInt*(items.get(i)));

**if** (numbers.size() == 0) е същото като **if** (numbers.isEmpty())

При премахване на елементи

List<Integer> numbers = **new** ArrayList<>(Arrays.*asList*(2, -2, 20, 42, 39, -1));  
**int** i = 0;  
**while** (i < numbers.size()) {  
 **if** (numbers.get(i) < 0) {  
 numbers.remove(i);  
 } **else** {  
 i++;  
 }  
}

**Когато сравняваме списъци, то винаги е препоръчително да използваме equals -дори и когато сравняваме double или int.**

merged.add(secondList.get(0));  
secondList.remove(0);

ИЛИ

merged.add(secondList.remove(0)); remove освен, че изтрива елемента, то връща стойността на изтрития елемент.

2 списъка в растящ ред да се преобразуват на 1 нов списък пак подреден в растящ ред.

**while** (!firstList.isEmpty() || !secondList.isEmpty()) {  
 **if** (firstList.isEmpty()) {  
 merged.add(secondList.get(0));  
 secondList.remove(0);  
 } **else if** (secondList.isEmpty()) {  
 merged.add(firstList.get(0));  
 firstList.remove(0);  
 } **else** {  
 **if** (firstList.get(0) < secondList.get(0)) {  
 merged.add(firstList.get(0));  
 firstList.remove(0);  
 } **else** {  
 merged.add(secondList.remove(0));  
 }  
 }  
}

Пример с addAll:

1)

List<String> result = **new** ArrayList<>();

String[] arr = input.split(**"\\s+"**);  
List<String> listToAdd = Arrays.*asList*(arr); //от масив към списък  
result.addAll(listToAdd);

2)

List<String> result = **new** ArrayList<>();

String[] arr = input.split(**"\\s+"**);  
**for** (String element : arr) {  
 result.add(element);  
}

**Lists<Integer>** num = Lists.*copyOf*(condensed, condensed.**size**); - копираме масив с по-къса дължина от началния – създава се нов списък, който сочи към различно място в паметта

num = condensed; - num е референция към стойността на същия списък condensed

Сортиране на списъци / масиви / стрингове / дати

Collections.sort(names);

Collections.reverse(names);

### **6.3. Печатане/Изход на лист:**

1) List<String> words = **new** ArrayList<> (Arrays.*asList*(**"the"**, **"quick"**, **"brown"**, **"fox"**));  
String joined = String.*join*(**","**, words); - **обединява в един стринг лист/списък само от стрингове**

System.***out***.println(joined);

2)

**private static** String joinElementsByDelimeter(List<Integer> items, String delimeter) {  
 String output = **""**;  
 **for** (Integer element : items) {  
 output += (element + delimeter);  
 }  
 **return** output;  
}

String output = *joinElementsByDelimeter*(numbers, **" "**);  
System.***out***.println(output);

3) List<Integer> numbers;

System.***out***.println(numbers); отпечатва се [6, 6, 3]

4) System.***out***.println(numbers.**toString()**.replaceAll(**"[\\[\\],]"**, **""**)); - прави от всякакви обекти на стринг, и вече от стринг заменя символите ,[ или ] с празен интервал/символ

System.***out***.println(numbers); отпечатва се 6 6 3

## Stack – LIFO (Last In, First Out)

ArrayDeque<Integer> stack = **new** ArrayDeque<>(); - Creating a Stack

stack.push(element); - Adding elements at the top

Integer element = stack.pop();- Removing elements

Integer element = stack.peek(); - Getting the value of the topmost element

**int** size = stack.size();  
**boolean** isEmpty = stack.isEmpty();  
**boolean** exists = stack.contains(2)

Колко операции имаме? По-малко ли са? Check this

ArrayDeque<String> stackMy = **new** ArrayDeque<>();  
System.***out***.println(stackMy.size()); - изпечатва всички елементи на стека

## **Queue / Опашка – FIFO (First In, First Out)**

ArrayDeque<Integer> queue = new ArrayDeque<>(); - Creating a Queue

Add

queue.add(element); - throws exception if queue is full

queue.offer(element); -returns false if queue is full

**for** (String child : children)  
 queue.offer(child);

String[] children = sc.nextLine().split(**" "**);  
ArrayDeque<String> queue = **new** ArrayDeque<>();  
Collections.*addAll*(queue, children); - Правим масива на опашка

Removing elements:

element = queue.remove(); - throws exception if queue is empty

element = queue.poll(); - returns null if queue is empty

element = queue.peek(); - check first element

Integer element = queue.peeк(); - checks the value of the first element  
Integer size = queue.size(); - returns queue size  
Integer[] arr = queue.toArray(); - converts the queue to an array  
**boolean** exists = queue.contains(element); - checks if element is in the queue

ArrayDeque<String> queue = **new** ArrayDeque<>();  
System.***out***.println(queue.size()); - изпечатва всички елементи на опашката

**for** (String child : children)  
 System.out.println(child);

### 5,1, Priority Queue

## **Многомерни масиви / Multidimensional Arrays**

### **6,1, Деклариране**

**int**[][] intMatrix;  
**float**[][] floatMatrix;  
String[][][] strCube;

**int**[][] intMatrix = **new int**[3][]; - поне една променлива трябва да има декларирана при Java  
**float**[][] floatMatrix = **new float**[8][2]; - дължина на всяко измерение  
String[][][] stringCube = **new** String[5][5][5];

**int**[][] array = **new int**[][]{{1, 2}, {3, 4}};

Въвеждане на стойност с цикли 1 – тази версия не е двумерен масив все още, а всеки елемент на първия масив има елементи в себе си.

**int**[][] arr = **new int**[rows][cols];  
**for** (**int** r = 0; r < rows; r++) {  
 arr[r] = Arrays.*stream*(sc.nextLine().split(**" "**)).  
 mapToInt(Integer::*parseInt*).toArray();  
}

Въвеждане на стойност с цикли 2

**int**[][] arr = **new int**[rows][cols];

**for** (**int** r = 0; r < rows; r++) {  
 String[] elements = sc.nextLine().split(**" "**);  
 **for** (**int** c = 0; c < elements.**length**; c++) {  
 **int** number = Integer.*parseInt*(elements[c]);  
 arr[r][c] = number;  
 }  
}

Въвеждане на стойност с цикли 3:

**int**[][] matrix = **new int**[][]{};  
**for** (**int** i = 0; i < rows; i++) {  
 **for** (**int** j = 0; j < cols; j++) {  
 matrix[i][j] = sc.nextInt();  
 }  
}

Въвеждане на стойност с цикли 4:

**int**[][] matrix = **new int**[rows][];  
  
**for** (**int** i = 0; i < rows; i++) {  
 matrix[i] = Arrays.*stream*(sc.nextLine().split(**" "**))  
 .mapToInt(Integer::*parseInt*)  
 .toArray();  
}

Въвеждане на двумерен масив с метод:

**int**[][] matrix = *readMatrix*(rows,cols, sc);

**private static int**[][] readMatrix (**int** rows, **int** col, Scanner sc) {  
 **int**[][] martix = **new int**[rows][col];  
 **for** (**int** r = 0; r < rows; r++) {  
 String[] elements = sc.nextLine().split(**" "**);  
 **for** (**int** c = 0; c < elements.**length**; c++) {  
 **int** number = Integer.*parseInt*(elements[c]);  
 martix[r][c] = number;  
 }  
 }  
  
 **return** martix;  
}

Въвеждане на char

**for** (**int** i = 0; i < rows; i++) {  
 String[] line = sc.nextLine().split(**" "**);  
 **for** (**int** j = 0; j < cols; j++) {  
 firstMatrix[i][j] = line[j].charAt(0);  
 }  
}

Въвеждане на стойност

**int**[][] array = {{1, 2}, {3, 4, 5}}; array.length ще ни изведе 2 в случая  
**int** element = array[1][1]; *// element11 = 4*

**int**[][] array = **new int**[3][4];  
**for** (**int** row = 0; row < array.**length**; row++)  
 **for** (**int** col = 0; col < array[0].**length**; col++)  
 array[row][col] = row + col;

### **6.2. Разни**

Сменяне стойността само на някои елементи от масива (когато правим операции с оригиналния масив не е добре веднага да сменяме стойности)

ArrayList<Integer> values = **new** ArrayList<Integer>();

ArrayList<**int**[]> indexes = **new** ArrayList<>();

values.add(validValue);

Indexes.add(**new int**[]{i, j});

**for** (**int** i = 0; i < values.size(); i++) {  
 matrix[indexes.get(i)[0]][indexes.get(i)[1]] = values.get(i);  
}

### **6.3. Печатане / изход**

Печатане

**for** (**int** row = 0; row < array.**length**; row++) {  
 **for** (**int** col = 0; col < array[row].**length**; col++) {  
 *//array[row][col] = row + col;* System.***out***.print(array[row][col]+ **","**);  
 }  
 System.***out***.println();  
}

Печатане

System.***out***.println(Arrays.*toString*(intMatrix[0]));

**for** (**int** i = 0; i < matrix.**length**; i++) {  
 **for** (**int** j = 0; j < matrix[i].**length**; j++) {  
 System.***out***.print(matrix[i][j] + **" "**);  
 }  
 System.***out***.println();  
}

## **Try Catch конструкция**

**try** {  
 **int** integer = Integer.*parseInt*(input);  
} **catch** (NumberFormatException e){  
 isInt = **false**;  
}

if (tryParseInt(input)) {

Integer.parseInt(input); // We now know that it's safe to parse

}

## **Рекурсия**

Без да принтираме по време на рекурсия

**public class** RecursiveFibonacci {  
 **public static void** main(String[] args) {  
 Scanner sc = **new** Scanner(System.***in***);  
  
 **int** n = Integer.*parseInt*(sc.nextLine());  
  
 **long** fib = *GetFibonacci*(n);  
 System.***out***.println(fib);  
 }  
  
 **private static long** GetFibonacci(**int** n) {  
 **if** (n == 2 || n == 1) {  
 **return** 1L;  
 }  
  
 **if** (n == 0) {  
 **return** 0;  
 }

**return** *GetFibonacci*(n - 1) + *GetFibonacci*(n - 2);  
   
 }  
}

С принтиране по време на рекурсия или записване на рекурсията в масив / лист

**public class** TribonacciSequence {  
 **public static void** main(String[] args) {  
 Scanner sc = **new** Scanner(System.***in***);  
 **int** nNum = Integer.*parseInt*(sc.nextLine());  
 *printTrib*(nNum);  
 }  
  
 **private static void** printTrib(**int** nnum) {//тук можем да боравим с i-ият елемент от рекурсията  
 **for** (**int** i = 1; i <= nnum ; i++) {  
 System.***out***.print(*tribonacciRecursion*(i) + **" "**);  
 }  
 }  
  
 **private static int** tribonacciRecursion(**int** num) {  
 **if** (num == 0) {  
 **return** 0;  
 } **else if** (num == 1 || num == 2) {  
 **return** 1;  
 } **else** {  
 **return** *tribonacciRecursion*(num - 1) + *tribonacciRecursion*(num - 2) + *tribonacciRecursion*(num - 3);  
 }  
 }  
}

**Дължина:**

**.size() – за списък**

**.length – за масив**

**.length() – за стринг**

## **9. Обекти и класове**

Класовете се пишат PascalCase

Random rnd = **new** Random();  
rnd.nextInt(5); *//[0, 4] 5 е границата и тя не се включва*

nextInt() – връща винаги положително число или нула

this. – викаме полето(private variable) или метод когато сме в текущия клас

Alt + Insert (Generate) в системата IntelliJ – автоматично настройва:

Constructor

* Getter
* Setter
* Getter and Setter
* Други

Classes define templates for object and consist of:

* Fields (**private variables**) – store values
* Getters and Setters
* Constructors – it is a kind of Setter (Overloading default constructor; Constructor name is the same as the name of the class)

**public class** Cat {  
 **private** String **name**;  
  
 **public** Cat(String name) { //конструктор  
 **this**.**name** = name;  
 }  
  
 **public** Cat() { //конструктор  
 }  
  
 **public void** setName(String name) { //setter  
 **this**.**name** = name;  
 }

**public** String getName() { //getter  
 **return name**;  
 }  
}

* **Behaviour - Methods**

Objects:

* Hold a set of named values
* Instance of a class - Обектът е инстанция на класа, т.е. е шаблон
* Всеки обект в main метода сочи към някаква референция от паметта

Import ready-to-use packages:

**import java.time.LocalDateTime;**

**import java.util.ArrayList;**

**import java.util.List;**

**import java.util.Scanner;**

Using ***static class members***:

**LocalDateTime today = LocalDateTime.now();**

**double cosine = Math.cos(Math.PI);**

**main** винаги е **static**

Math.abs – статични – не зависи от вътрешното състояние на класа Math

Dice sides6 = Dice.*generateWithSides*(6); - статичен метод – не зависи от вътрешното състояние на класа Dice

Using ***non-static Java classes:***

**Random rnd = new Random();**

**int randomNumber = rnd.nextInt(99);**

Dice d = **new** Dice(); d.setSides(6); - non-static

Когато принтираме, самата функция println вика вградения метод на всеки един клас **toString**

Articles article = **new** Articles(**""**, **""**, **""**);

System.***out***.println(article); = System.***out***.println(article.toString());

@Override – казва ни ако сме объркали името на метода toString  
**public** String toString() {  
 String result = String.*format*(**"%s - %s:%s"**, **this**.**title**, **this**.**content**, **this**.**author**);  
 **return** result;  
}

Пишем Main клас с main метод в package-а – за да може Judge да провери

List<Person> people = **new** ArrayList<>();

people  
 .stream()  
 .filter(p->p.getAge() > 30)  
 .sorted((p1, p2) -> p1.getName().compareTo(p2.getName()))  
 .forEach(p -> System.***out***.println(p));

List<OrderByAge> orderByAgeList = **new** ArrayList<>();

orderByAgeList  
 .stream()  
 .sorted((p1, p2) -> Integer.*compare*(p1.getAge(), p2.getAge())) //сортира възходящо  
 .forEach(p -> System.***out***.println(p));

sorted(Comparator.*comparingInt*(p -> p.getAge()))

**1.** Можем да добавяме в един клас поле от Тип друг клас – в обекта на инстанцията можем да стигнем до инстанцията на всеки от класовете с подготвени getter-и и точка и точка

**public class** Car {  
 **private** String **modelCar**;  
 **private** Engine **connectWithEnginClassModelEngine**;

**public** Engine getConnectWithEnginClassModelEngine() {  
 **return connectWithEnginClassModelEngine**;}

**public void** setConnectWithEnginClassModelEngine(Engine connectWithEnginClassModelEngine) {  
 **this**.**connectWithEnginClassModelEngine** = connectWithEnginClassModelEngine;}

}

**public class** Engine {  
*// model, power, displacement and efficiency* **private** String **modelEngine**;  
 **private** String **powerEngine**;  
 **private** String **displacementEngine**;  
 **private** String **efficiencyEngine**;

}

В main метода на Main класа:

List<Car> carList = **new** ArrayList<>();

Car currCar = **new** Car(tokens[0]);

**for** (Engine engine : engineList) {  
 **if** (engine.getModelEngine().equals(tokens[1])) {  
 currCar.setConnectWithEnginClassModelEngine(engine);  
 }  
}  
carList.add(currCar);

**for** (Car car : carList) {  
 System.***out***.println(car.getModelCar() + **":"**);  
 System.***out***.println(car.getConnectWithEnginClassModelEngine().toString());  
 System.***out***.println(car.toString());  
}

**2.** Или можем да имаме поле от тип List<String> и да достигаме до него и да добавяме на листа елементи с функцията add.

**public class** Team {  
 **private** String **teamName**;  
 **private** String **creatorName**;  
 **private** List<String> **memberName**;  
  
 **public** Team(String teamName, String creatorName) {  
 **this**.**teamName** = teamName;  
 **this**.**creatorName** = creatorName;  
 List<String> temp = **new** ArrayList<>();  
 **this**.**memberName** = temp;  
 }

В метода main – с временна променлива Списък от тип Стринг.

Team newTeam = **new** Team(tokensCreateTeam[1], tokensCreateTeam[0]);  
teamList.add(newTeam);

List<String> newMember = **new** ArrayList<>();  
newMember = **null**;

**for** (**int** i = 0; i < teamList.size(); i++) { *//Pesho->AiNaBira* **if** (tokensJoinMember[1].equals(teamList.get(i).getTeamName())) {  
 isTeamCreated = **true**;  
 newMember = teamList.get(i).getMemberName();  
 newMember.add(tokensJoinMember[1]);  
 teamList.get(i).setMemberName(newMember);  
 **break**;  
 }  
}

Или по този начин:

**public class** Team {  
 **private** String **teamName**;  
**private** List<String> **membersNames** = **new** ArrayList<>();  
  
 String getMember(**int** i) {  
 **return membersNames**.get(i);  
 }

В метода main:

**if** (tokensCreateTeam[0].equals(team.getMember(0)))

## **10. Associative Arrays (Maps), Lambda and Stream API**

### 10.1. Associative Arrays - Collection of Key and Value Pairs – MAPS

#### Описание

Hash означава, че има уникални стойности на по-ниско ниво, и не се налага много много преобразуване на типове данни

**А) HashMap <key, value> HashMap<K, V> - Keys are unique - Uses a hash-table + list**

Map<String, Integer> map = **new** HashMap<>();

map.put(**"Ivan"**, 73);  
map.put(**"Ivan"**, 53);- при опит на пре-записване на Иван, се презаписва само стойността от 73 на 53!

**Произволно записва кой-елемент от листа на кое място е.**

Най-бързия и оптимизиран, но не винаги върши работа.

**B)** **LinkedHashMap <key, value> LinkedHashMap<K, V> -Keys are unique -Keeps the keys in order of addition**

Map<String, Integer> map = **new** LinkedHashMap<>();

Помни последователно ключовете.

при опит на пре-записване на Иван, се презаписва само стойността от 73 на 53!

**C) TreeMap <key, value> TreeMap<K, V> Keys are unique Keeps its keys always sorted Uses a balanced search tree**

Map<String, Integer> map = **new** TreeMap<>();

при опит на пре-записване на Иван, се презаписва само стойността от 73 на 53!

#### Въвеждане

Map<Integer, Integer> map = **new** HashMap<>() {{  
 put(2, 6);  
}};  
**int** result = map.remove(2); //remove(K) връща стойността V на двойката (K, V)

**char**[] input = sc.nextLine().toCharArray();  
LinkedHashMap<Character, Integer> letters = **new** LinkedHashMap<>();  
**for** (**char** letter : input) {  
 letters.putIfAbsent(letter, 0);  
 **int** count = letters.get(letter);  
 letters.put(letter, count + 1);  
}

iter/while {

resources.putIfAbsent(input, 0);  
**int** oldCount = resources.get(input);  
resources.put(input, oldCount + count);

}

**double**[] numbers = Arrays.stream(sc.nextLine().split(**"\\s+"**))  
 .mapToDouble(Double::parseDouble)  
 .toArray();  
Map<Double, Integer> map = **new** TreeMap<>();  
**for** (**double** number : numbers) {  
 **if** (!map.containsKey(number)) {  
 map.put(number, 1);  
 } **else** {  
 map.put(number, map.get(number) + 1);  
 }  
}

HashMap<String, HashMap<String, Integer>> allPlayers = **new** HashMap<>();

allPlayers.putIfAbsent(playerName, **new** HashMap<>());  
allPlayers.get(playerName).putIfAbsent(positionSkill, -1);  
**if** (skillPoints > allPlayers.get(playerName).get(positionSkill)) {  
 allPlayers.get(playerName).put(positionSkill, skillPoints);  
}

#### Методи:

* **put(key, value)** method - airplanes**.put**("Airbus A320", 150);
* **remove(key)** method - airplanes**.remove**("Boeing 737");
* **size() –** взема размера
* **containsKey(key)** method **-** if (**map.containsKey**("Airbus A320"**)**)

За недублиране на елементи:

String[] names = sc.nextLine().split(**"\\s+"**);  
**int**[] points = Arrays.*stream*(sc.nextLine().split(**"\\s+"**))  
 .mapToInt(Integer::*parseInt*)  
 .toArray();  
Map<String, Integer> map = **new** TreeMap<>();  
  
**for** (**int** i = 0; i < names.**length**; i++) {  
 String name = names[i];  
 **int** point = points[i];  
  
 **if** (!map.containsKey(name)) {  
 map.put(name, point);  
 }  
 **if** (map.containsValue(20)) {  
 System.***out***.println(**"Already created"**);  
 }  
}

* **containsValue(value)** method:

map.put("Airbus A320", 150);

System.out.println(**map.containsValue(**150**)**); //true

* **get(K) връща V** – ако ключа го няма, хвърля null (pointer exception) – **иначе връща стойност**

map.**get**(“Ivan”) - връща 53.

**Не гледа по индекс на ключа, а по стойност на ключа**

**for** (String s : map.keySet(name)) {  
 **if** (!map.containsKey(name)) {  
 map.put(name);  
 }  
}

При foreach цикъла, като натиснем iter ни дава предложение дали да създадем:

**for** (String s : map.keySet()) {} – колекция от ключове map.keySet() e от тип Set<String>

**for** (Integer value : map.values()) {} – колекция от стойности

**for** (Map.Entry<String, Integer> keyPlusValuePair : **map.entrySet()**) {} – колекция от двойки

* **entry set –** достъпвам двойката key and value заедно, а не поотделно

Map<String, Double> fruits = new LinkedHashMap<>();

fruits.put("banana", 2.20);

fruits.put("kiwi", 4.50);

for (**Map.Entry<String, Double>** keyValuePair : fruits.**entrySet()**) {

System.out.printf("%s - %.2f%n", keyValuePair.**getKey()**, keyValuePair.**getValue()**);

**}**

Set<String> - уникален лист/списък, в който всички стойности са уникални и не се повтарят

**Стойност от тип Лист от Стринг - добавяне на елемент от листа от стринг + ползване на *putIfAbsent()***

Map<String, List<String>> map = **new** LinkedHashMap<>(); //

map.**putIfAbsent**(word, **new** ArrayList<>()); *//* винаги инициализираме списъка с new ArrayList() за да може да добавяме add *- този ред се прескача реално ако е в цикъл – Да избягвам да ползвам putIfAbsent – защото много пъти се налага допълнителна проверка, и по-добре да си пиша if-ве*  
map.get(word).add(syn);

#### Пример за ForEach вложен цикъл / вложени мапове

LinkedHashMap<**String**, **LinkedHashMap<String, Integer>**> contestsAll = **new** LinkedHashMap<>();  
LinkedHashMap<String, Integer> **standings** = **new** LinkedHashMap<>();  
**for** (***Map.Entry***<String, HashMap<String, Integer>> **contestName** : **contestsAll.entrySet()**) {  
 **for** (***Map.Entry***<String, Integer> userName : **contestName.getValue().entrySet()**) {  
 standings.putIfAbsent(userName.getKey(), 0);  
 standings.put(userName.getKey(), userName.getValue() + standings.get(userName.getKey()));  
 }  
}

### 10.2. Lambda expression – анонимни функции/не си ги пишем ние

x -> x / 2 static int func(int x) { return x / 2; }

x -> x != 0 static boolean func(int x) { return x != 0; }

() -> 42 static int func() { return 42; }

**int**[] numbers = Arrays.*stream*(sc.nextLine().split(**"\\s+"**))  
 .mapToInt(**n -> Integer.*parseInt*(n) / 2**) **– това е Lambda израз**  
 .toArray();

**Lambda** and **Stream API** help collection processing

.mapToInt(x -> Integer.*parseInt*(x)) ==.mapToInt(Integer::*parseInt*) е от тип малкия int (IntStream)

.map(x -> Integer.*parseInt*(x)) ==.map(Integer::*parseInt*) e от бащиния тип Integer (Stream<Integer>)

### 10.3. Stream API

Поток от данни е **stream**

**Lambda** and **Stream API** help collection processing

Arrays.*stream*(sc.nextLine().split(**"\\s+"**))  
 .mapToInt(n -> Integer.*parseInt*(n) / 2);

**.min();**

**.orElse(2); - или върни 2**

**.average();**

**.getAsDouble(); -** ако е OptionalMin\_Max – слагаме накрая и getAsDouble.

**.getAsInt(); -** ако е Optional int

**.get() –** ако е Optional String

**.sum();**

#### Mapping

**map() - manipulates elements in a collection:**

.mapToInt(**p -> Integer.parseInt(p)**) което е същото като .mapToInt(**Integer::parseInt()**)

**Правим нов масив с други елементи**

String[] words = {"abc", "def", "geh", "yyy"};

words = Arrays.stream(words)

.map(w -> w + "yyy")

.toArray(String[]::new);

.collect(Collectors.*toList*());

**.toArray();**

**.collect(Collectors.toList());**

#### Filtering

**.filter**(n -> n > 0);

**.filter(**w -> w.length() % 2 == 0)

int min = Arrays.stream(new int[]{15, 25, 35}).**min().getAsInt();**

int max = nums.stream().mapToInt(Integer::intValue).max().getAsInt();

int max = nums.stream().max(Integer::compareTo).get();

int sum = nums.stream().mapToInt(Integer::intValue).**sum();**

#### Ordering/Sorting

**.sorted**((n1, n2) -> n1.compareTo(n2)) - ascending

**.sorted**((n1, n2) -> n2.compareTo(n1)) - descending

**.limit(3) – ограничи сортировката до първите 3 стойности**

**.sort() – е на масив / Лист, но не и на поток данни stream**

**Sorting Collections by Multiple Criteria**

a.compareTo(b) – когато е бащин Integer или String

Double.*compare(a, b)*

Integer.*compare*(second, first) или second - first; - когато е само int

.sorted((e1, e2) -> {

int res = e2.getValue().**compareTo**(e1.getValue());// сортираме по Value – **compareTo()** връща 1, 0 или -1

if (res == 0)

res = e1.getKey().compareTo(e2.getKey());//ако са равни, то ги сортираме и по Key

return res; })

.forEach(e -> System.out.println(e.getKey() + " " + e.getValue()));

Map<String, List<Integer>> teams = **new** HashMap<>();  
teams.put(**"Sanow"**, Arrays.*asList*(1, 23, 45));  
teams.put(**"Sbnow"**, Arrays.*asList*(19, 39, 29));  
teams.put(**"Acb"**, Arrays.*asList*(45, 23, 12));  
  
teams.entrySet()  
 .stream()  
 .sorted((e1, e2) -> {  
 **if** (e1.getKey().**compareTo**(e2.getKey()) == 0) {  
 **int** sum1 = e1.getValue().stream().mapToInt(x -> Integer.*parseInt*(x+**""**)).sum();  
 **int** sum2 = e1.getValue().stream().mapToInt(x -> Integer.*parseInt*(x+**""**)).sum();  
  
 **return** sum1 - sum2; *//връща 1 0 или -1* }  
 **return** e2.getKey().compareTo(e1.getKey());  
 }  
 )  
 .forEach(e -> { **//functional forEach**  
 System.***out***.println(**"Key :"** + e.getKey());  
 System.***out***.println(**"Values -> "**);  
 e.getValue().sort(Integer::*compare*);  
 **for** (Integer age : e.getValue()) {  
 System.***out***.printf(**"---%d%n"**, age);  
 }  
 });

#### Out/printing

letters  
 **.entrySet()**  
 .forEach(p -> System.***out***.println(String.*format*(**"%c -> %d"**, p.getKey(), p.getValue())));

items  
 .entrySet()  
 .**stream()  
 .sorted**((i1, i2) -> i2.getValue() - i1.getValue())

//.sorted((i1, i2) -> i2.getValue().compareTo(i1.getValue()))

.forEach(i -> System.***out***.println(String.*format*(**"%s: %d"**, i.getKey(), i.getValue())));

resources  
 .forEach((k, v) -> System.***out***.println(String.*format*(**"%s -> %d"**, k, v)));

courses  
 .entrySet()  
 .stream()  
 .sorted((c1, c2) -> {  
 **int** first = c1.getValue().size();  
 **int** second = c2.getValue().size();  
 **return** Integer.*compare*(second, first);  
 })  
 .forEach(c -> {  
 System.***out***.println(String.*format*(**"%s: %d"**,  
 c.getKey(),  
 c.getValue().size()));  
  
 c.getValue()  
 .stream()  
 .sorted((s1, s2) -> s1.compareTo(s2))  
 .forEach(s -> System.***out***.println(String.*format*(**"-- %s"**, s)));  
 });

#### Използване на **final** AtomicInteger

**final** AtomicInteger **br** = **new** AtomicInteger();

В stream-a във forEach частта при разпечатване:

.forEach(s -> {  
 System.***out***.println(String.*format*(**"%s. %s <::> %d"**, **br.incrementAndGet()**, s.getKey(), s.getValue()));  
});

br.set(1);

Също така става и с масив от 1 елемент със стойност 0 - РАБОТИ 😊

.forEach(contnt -> {  
 System.***out***.println(String.*format*(**"%s: %d participants"**, contnt.getKey(), contnt.getValue().size()));  
 Map<String, Integer> students = **new** HashMap<>();  
 students = contnt.getValue();

students  
 .entrySet()  
 .stream()  
 .sorted((………

}

## 11. Text Processing

Strings are immutable (read-only)

Accessible by index (read-only)

Strings use Unicode

### 11.1. Initializing a String

String str = **"Hello, Java"**;

String name = sc.nextLine();

String name = **new** String(**"Pesho"**);

Converting a **string** from and to a **char** **array**:

String str = **new** String(**new char**[] {**'s'**, **'t'**, **'r'**});  
**char**[] charArr = str.toCharArray();

!!! String.*valueOf*(5) *е същото като* (5 + **""**)

### 11.2. Manipulating Strings using the String Class– **масив от Char**, по-малко на брой операции позволява

#### Concatenating

String text = "Hello" + ", " + "world!";

String text = "Hello, ";

text += "John";

Use the concat**()** method

String greet = **"Hello, "**;  
String name = **"John"**;  
String result = greet.concat(name); *// "Hello, John"*

#### Join

Joining Strings

**String.join("", …)** concatenates strings

String t = String.*join*(**""**, **"con"**, **"ca"**, **"ten"**, **"ate"**);

Joining Strings - an array/list of strings

String s = **"abc"**;  
String[] arr = **new** String[3];  
**for** (**int** i = 0; i < arr.**length**; i++) { arr[i] = s; }  
String repeated = String.*join*(**""**, arr); *// "abcabcabc"*

#### Substring

**substring**(int startIndex, int endIndex)

String card = **"10C"**;  
String power = card.substring(0, 2); - взема до endIndex минус 1  
System.***out***.println(power); *// 10*

**substring(**int startIndex**)**

String text = **"My name is John"**;  
String extractWord = text.substring(11);  
System.***out***.println(extractWord); *// John*

#### Searching – indexOf(), lastIndexOf(), contains()

**indexOf()** - returns the first match index or -1

String fruits = **"banana, apple, kiwi, banana, apple"**;  
System.***out***.println(fruits.indexOf(**"banana"**)); *// 0*System.***out***.println(fruits.indexOf(**"orange"**)); *// -1*

**lastIndexOf()** - finds the last occurrence

String fruits = **"banana, apple, kiwi, banana, apple"**;  
System.***out***.println(fruits.lastIndexOf(**"banana"**)); *// 21*System.***out***.println(fruits.lastIndexOf(**"orange"**)); *// -1*

**contains()** - checks whether one string contains another

String text = **"I love fruits."**;  
System.***out***.println(text.contains(**"fruits"**));*// true*System.***out***.println(text.contains(**"banana"**));*// false*

#### Replacing

С използванeто нa **indexOf** и **substring**

String toRemove = sc.nextLine();  
String text = sc.nextLine();  
  
**while** (text.contains(toRemove)) {  
 **int** toRemoveindex = text.indexOf(toRemove);  
 **int** toRemoveLength = toRemove.length();  
  
 text = text.substring(0, toRemoveindex) +  
 text.substring(toRemoveindex + toRemoveLength);

ИЛИ

**replace(match, replacement)** - replaces **all** occurrences

String toRemove = sc.nextLine();  
String text = sc.nextLine();  
text = text.replace(toRemove, **""**);

#### Splitting

**Split** a string by given **pattern**

String[] words = text.split(**", "**);

**Split** by **multiple separators**

String text = **"Hello, I am John."**;  
String[] words = text.split(**"[, .]+"**);

// разделя по който и да е от регулярните изрази(regular expression)

*// "Hello", "I", "am", "John"*

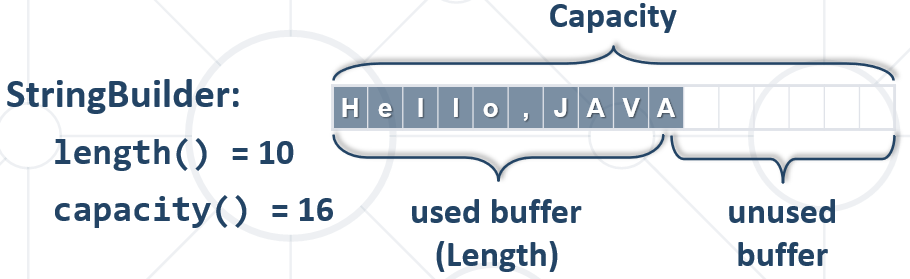
#### **Прохождане на String-a / масивa от чарове**

String message = sc.nextLine();  
**for** (**int** i = 0; i < message.length(); i++) {  
 **char** letter = message.charAt(i);  
}

### 11.2. Using the StringBuilder Class – **списък от Char** – с бонус операции в сравнение работа с нормалния String

**StringBuilder** keeps a buffer space, allocated in advance and **it works a lot quicker**

**Concatenating** strings is a **slow** operation because each iteration **creates** a **new string**



#### **Инициализация**

StringBuilder sb = **new** StringBuilder();  
 sb.append(**"Hello, "**);  
 sb.append(**"John! "**);  
 sb.append(**"I sent you an email."**);  
 System.***out***.println(sb.toString()); *// Hello, John! I sent you an email.*

#### **append()**

- appends the string representation of the argument

StringBuilder sb = **new** StringBuilder();  
sb.append(**"Hello Peter, how are you?"**);

comments.append(String.*format*(**"<div>%n %s%n</div>%n"**, input));

System.***out***.println(comments.toString()); - печата форматираната версия, с нови редови и т.н.

#### **length()**

- holds the length of the string in the buffer

System.***out***.println(sb.length()); *// 25*

#### **setLength(0)**

- removes all characters -това е равно в класа String на String result = **""**;

#### **setCharAt();**

#### **charAt(int index)**

- returns char on index

StringBuilder sb = **new** StringBuilder();  
sb.append(**"Hello Peter, how are you?"**);

System.***out***.println(sb.charAt(1)); //e

#### **insert(int index, String str)**

– inserts a string at the specified character position

sb.insert(11, **" Ivanov"**);  
System.***out***.println(sb); *// Hello Peter Ivanov, how are you?*

#### **replace(int startIndex, int endIndex, String str)**

- replaces the chars in a substring

StringBuilder sb = **new** StringBuilder();

sb.append(**"Hello Peter, how are you?"**);  
sb.replace(6, 11, **"George"**);- премахни нещата от 6ти до 11ти индекс, и на тяхно място сложи George

//**"Hello George, how are you?"**

#### **toString() – изход/печат**

-converts the value of this instance to a String – преобразува **масива от данни StringBuilder** отново на String

String text = sb.toString();  
System.***out***.println(text); //**"Hello George, how are you?"**

#### **delete()**

StringBuilder letters = **new** StringBuilder();  
letters.delete(0, 4); - изтрива от 0вия индекс до 3тия индекс(десния индекс минус 1)

#### **insert()**

letters.insert(2, **"pesho"**);

#### **substring()**

String sub **= substring**(int startIndex, int endIndex);

String sub **= substring(**int startIndex**)**

#### **reverse**()

StringBuilder sb = **new** StringBuilder();

System.***out***.println(sb.**reverse().**toString());

### 11.3. Using the Character Class

Character.*isDigit*(5) – връща true

Character.*isLetter*(**'z'**); - връща true

Character.*isLetterOrDigit*(**'z'**);

Character.isUppercase

## 12. Regular Expressions (RegEx)

### Match text by pattern

<https://regex101.com/>

<https://regexr.com/>

[http://regexone.com](http://regexone.com/)

За група от символи:

**[nvj]** matches any character that is either **n**, **v** or **j**

**[abc][def]** първи символ a b или c; втори символ d, e или f

**[^a]**  matches any character that is not a

**[^abc]**  matches any character that is not a, b or c

**[0-9]** character range matches any digit from **0** to **9**

**[0-9]+** matches non-empty sequence of digits – повече от една цифра последователно записана

**[A-Z][a-z]\*** matches a capital + small letters

**.** matches any character everything except for line terminators

**\.** – символът точка само

**[A-Z][a-z]+ [A-Z][a-z]+** точно една главна буква с повече малки букви плюс space плюс още един път една главна буква с повече малки букви

**John Smith**

* **\w** - matches any **word character** (a-z, A-Z, 0-9, \_) == [A-Za-z0-9\_] - дефиниция за **alphanumeric**
* **\W** - matches any **non-word character** (the opposite of \w)
* **\s** - matches any **white-space** character
* **\S** - matches any **non-white-space**  character (opposite of \s)
* **\d** - matches any **decimal digit** (0-9)
* **\D** - matches any **non-decimal character** (the opposite of \d)
* \b To prevent capturing of letters across new lines, put "\b" at the beginning and at the end of your regex

работи за думи в изречение, когато не гледаме за начало и край на ред с ^(I) am (Svilen)$

matches the empty string at the beginning or end of a word

При завършване на дума/изречение с точка, въпросителен, удивителен, запетая – го взема

При \bFoo – гледа дали преди Foo е точка, запетая, space, удивителен, въпросителен

При Foo\b – гледа дали след Foo е точка, запетая, space, удивителен, въпросителен

\b is zero-width, it doesn't actually match any character

### Quantifiers

**\*** matches the previous element **zero** or more times – мачва колкото се може повече цифри d

**\+\d\* +359885976002 a+b**

**+** matches the previous element **at least ONE** or more times - мачва колкото се може повече цифри d

**\+\d+ +359885976002 a+b**

**?** matches the previous element **zero or one time –** мачва до една цифра d

**\+\d? +359885976002 a+b**

**{3}** matches the previous element exactly 3 times – мачва до 3 цифри d

**\+\d{3} +359885976002 a+b**

**{3,}** matches the previous element exactly minimum 3 times – мачва 3 или повече цифри d

**\+\d{3,} +359885976002 a+b**

**{3,10}** matches the previous element exactly minimum 3 to 10 times – трябва да има поне 3 съвпадение или до 10 съвпадения цифри d

**\+\d{3,} +359885976002 a+b**

**? – optionality ab?c** will match either the strings "abc" or "ac" because the b is considered optional

**\?** plain character ?

**.\*?** matches between zero and unlimited times **(lazy model – as few times as possible, до първото срещане на backreference например)**

**.\*** matches between zero and unlimited times **(greedy model – as more times as possible, до последното срещане на backreference например)**

**(subexpression) -** captures the matched subexpression as numbered group – ограждаме в скоби

**(?:subexpression)** - defines a non-capturing group – група, която не се брои - The parser uses it to match the text, but ignores it later, in the final result.

**(?<name>subexpression)** - defines a named capturing group – ограждаме в скоби и си слагаме име на всяка група

**(?<day>\d{2})-(?<month>\w{3})-(?<year>\d{4}) 22-Jan-2015**

Шаблон за е-мейл – как изглеждат данните, които търся

\w+@[A-Za-z]+\.[A-Za-z]+

[**valid123@email.bg**](mailto:valid123@email.bg)

[**invalid\*name@emai1.bg**](mailto:invalid*name@emai1.bg)

**start and the end of the line** using the special **^** and **$**

**^(I) am (Svilen)$ -** изразът трябва да започне с I и да завърши с Svilen

Note that this is different than the hat used inside a set of bracket **[^...]** for excluding characters, which can be confusing when reading regular expressions.

**Capture** as a group

Правим проверка на всичко, но хващаме в група само това, което е преди точката и разширението

**^(.+)\.pdf$** - всички имена до точката, т.е. без точката и разширението на файла

**Nested groups / capture subgroup**

**(\w+ (\d+)) – хванатите групи на Jan 1987 са** Jan 1987 и 1987

I love (cats|dogs) – regex котки или кучета

I love cats

I love dogs

### Backreferences

**\number** - matches the value of a numbered capture group

<(\w+)[^>]\*>.\*?<\/\**1**>

Вземи **<**

след това вземи една или няколко букви без >

след това вземи **>**

след товавземи който и да е символ много на брой пъти, но lazy model – **при първото срещане** на backreference **.\*?**

след това вземи **</**

след това вземи референция от група 1, може да вземаме от други групи референции

накрая сложи **>**

<b>Regular Expressions</b> are cool!

<p>I am a paragraph</p> … some text after

Hello, <div>I am a<code>DIV</code></div>!

<span>Hello, I am Span</span>

<a href="https://softuni.bg/">SoftUni</a>

### Regex in Java library

#### Basic declarations/operartions

java.util.regex.Pattern

java.util.regex.Matcher

Pattern pattern = Pattern.*compile*(**"a\*b"**);  
Matcher matcher = pattern.matcher(**"aaaab"**);

**boolean** match = matcher.find(); - searches for the next match  
String matchText = matcher.group(); - gets the matched text

System.***out***.println(matcher.groupCount()); - показва колко групи има мачнати от шаблона / групите на шаблона в обикновени скоби ()

String text = **"Andy: 123"**;  
String pattern = **"([A-Z][a-z]+): (?<number>\\d+)"**;  
  
Pattern regex = Pattern.*compile*(pattern);  
Matcher matcher = regex.matcher(text);  
  
System.***out***.println(matcher.find()); *// true !!!* ***– като дадем веднъж find, 2-ри път не може да го намери***System.***out***.println(matcher.group()); *// Andy: 123 - всичко*System.***out***.println(matcher.group(0)); *// Andy: 123 - всичко*System.***out***.println(matcher.group(1)); *// Andy – първа група*System.***out***.println(matcher.group(2)); *// 123 – втора група*System.***out***.println(matcher.group(**"number"**)); *// 123 – група с име number*

Pattern pat = Pattern.*compile*(**"\\b[A-Z][a-z\_]+ [A-Z][a-z]+"**);  
Matcher matcher = pat.matcher(text);  
**while / if** (matcher.find()) {  
 System.***out***.print(matcher.group(0) + **" "**);  
}

Използване за шаблон String.format с променливи стойности

String pattern3 = String.*format*(**"^(%c[^ ]{%d})$"**, letter.getKey(), letter.getValue() - 1);  
Pattern word3Pattern = Pattern.*compile*(pattern3);  
**for** (**int** i = 0; i < word3.**length**-1; i++) {  
 Matcher word3Matcher = word3Pattern.matcher(word3[i]);  
 **if** (word3Matcher.find())

#### Replacing With Regex

To replace **every**/**first** subsequence of the input sequence that matches the pattern with the given replacement string

**replaceAll(String replacement)**

**replaceFirst(String replacement)**

String regex = **"[A-Za-z]+"**;  
String string = **"Hello Java"**;  
Pattern pattern = Pattern.*compile*(regex);  
Matcher matcher = pattern.matcher(string); - резултата от прикагането на   
String res = matcher.replaceAll(**"hi"**); *// hi hi*String res2 = matcher.replaceFirst(**"hi"**); *// hi Java*

#### Splitting with Regex

**split(String pattern)** - splits the text by the pattern

String text = **"1 2 3 4"**;  
String pattern = **"\\s+"**; - отбелязва което и да е от четирите вида whitespace  
String[] tokens = text.split(pattern);

Whitespaces:

**space** (**␣**),

**tab** (**\t**)

**new line** (**\n**)

the carriage return (**\r**)

**\s** will match **any** of the specific whitespaces above

Цяло число или дробно число

**"(-?\\d\*\\.?\\d+)"**

String regex = **"(^|\\s)[a-z0-9][\\.\\\_\\-a-z0-9]\*[a-z0-9]@[a-z0-9][\\.\\-a-z0-9]\*[a-z0-9]\\.[a-z]{2,}"**;  
Pattern pattern = Pattern.*compile*(regex);  
Matcher matcher = pattern.matcher(input);  
**while** (matcher.find()) {  
 sb.append(matcher.group() + **"\n"**);  
}

Какво прави дългия регекс:

**^|\\s** - Искаме да проверим дали има начало на стринг ИЛИ ( "|") дали има празно място (това прави първа група)

**[a-z0-9]** – една малка буква или число

**[\\.\\\_\\-a-z0-9]\*** - дали са точка, долна черта, тире, малка буква или число

**[a-z0-9]** - една малка буква или число

**@** - маймунка

**[a-z0-9]** - една малка буква или число

**[\\.\\-a-z0-9]\* -** дали са точка, тире, малка буква или число

**[a-z0-9] -** - една малка буква или число

**\\.** – точка

**[a-z]{2,}** – две или повече малки букви

<https://www.regular-expressions.info/refadv.html> - Regular Expression Reference: Special Groups

positive lookahead и negative lookahead