Съдържание

1. BASIC SYNTAX, CONDITIONAL STATEMENTS AND LOOPS

1.1. Четене от конзолата

```
int age = Integer.parseInt(scanner.nextLine());                 //стринг с преобразуване към инт
String username = scanner.nextLine();                           // стринг
double price = Double.parseDouble(scanner.nextLine());          // стринг към дабъл
```

1.2. Преобразуване между типове данни

Кастване : **char -> int**

```
char symbol = 'A';
int asciiValue1 = (int) symbol;              // explicit casting, символ към аски код
int asciiValue2 = symbol;                    // implicit casting

String number = "145";
char symbol = number.charAt(0);              //String -> char
String symbolAsText = symbol + "";           // char -> String
int digit = Integer.parseInt(symbolAsText);          // String -> int
double digit = Double.parseDouble(symbolAsText);        String -> Double
int digit = Integer.parseInt(inputNumber.charAt(position) + "")   //позиция от String -> int
```

1.3. Форматирано отпечатване

```
System.out.printf("Name: %s, Age: %d, Grade: %.2f", name, age, averageGrade);
```

%s – String

%d – int

%.2f - double

%n – нов ред

1.4. Други

group.equals("Students") // стрингът group **е еднакъв** на Students

!input.equals("Start") // стрингът input **не е еднакъв** със стрингът Start

number % 2 != 0 // числото **не е четно**, %2 – остатък при делене на две

int lastDigit = number % 10 // **последна цифра на число**

number /= 10; // **премахване на последната цифра** на число

if (Character.isUpperCase(symbol) // проверка **дали стринг започва с главна буква**

if (Character.isLowerCase(symbol) // проверка **дали стринг започва с малка буква**

2. МАСИВИ

2.1. Обща информация

1. масивът е съвкупност от еднотипни елементи
2. масивът има постоянна дължина array.length
3. дължина на масив = максималния брой елементи, които можем да съхраним
4. позиции / индекси -> 0 до последната (array.length - 1)
5. задаване стойност в масив: array[0] = 56;
6. достъпване стойност в масив: array[5]

2.2. Създаване на масив

double[] prices = new double[10]; - празен, с определена предварително дължина

2.3. Пълнене на масив

//вариант 1 за запълване на масив (статичен с предварително зададени елементи)

int [] dates = {4, 5, 6, 7};

//вариант 2 за запълване на масив (празен масив и добавяме елементи)

double[] prices = new double[10];

prices[0] = 34.5;

prices[1] = 23.5;

//вариант 3 за запълване на масив (**елементите се въвеждат от конзолата на отделни редове**)

int n = Integer.parseInt(scanner.nextLine()); - брой на елементите на масива

int [] numbers = new int[n];

for (int position = 0; position < numbers.length; position++) {

numbers[position] = Integer.parseInt(scanner.nextLine());

//вариант 4 за запълване на масив (**елементите са на един ред, разделени с интервал**)

String[] inputs = scanner.nextLine().split(" ");

int [] integerNumbers = Arrays.stream(scanner.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray(); //**int**

double [] decimalNumbers = Arrays.stream(scanner.nextLine().split(")).mapToDouble(Double::parseDouble).toArray(); //**double**

2.4. Обединяване и преобразуване на елементите на масив

String names = "Desi Ivan Georgi Tanya";

System.out.println(String.join(" ", namesArray)); //"Desi Ivan Georgi Tanya" // **обединяване**

System.out.println(Arrays.toString(numbers).replace("[", "").replace("]", "") //**обединяване със замяна на разделител**

"Desi".toCharArray() -> ['D', 'e', 's', 'i'] // Стринг **към символен масив**

2.5. Извеждане на елементите на масив

//използване **for**: има значение позицията на елемента

for (int position = 0; position <= daysOfWeek.length - 1; position++) {

System.out.println(daysOfWeek[position]);

```
}
//използване foreach: няма значение позицията на елемента
for (String day : daysOfWeek) {
System.out.println(day);
}
//отпечтване в обратен ред
for (int position = numbers.length - 1; position >= 0; position--) {System.out.print(numbers[position] + " ");}
```

3. СПИСЪК- LIST

1. Размер на списък

```
System.out.println(numbers.size());    //размер на списъка
```

2. Използване на елемент от списъка

```
numbers.get(1);        // взима елемент от списъка на посочената позиция
```

3. Добавяне на елемент към списъка

```
numbers.add(50);          //добавя елемента в края на списъка -> {50}
numbers.add(5, 23);       //вмъква елемент на дадена позиция; изместваме останалите
numbers.set(1, 45);       //заменя елемента на дадена позиция с дадения елемент
```

4. Премахване на елемент от списъка

```
numbers.remove(Integer.valueOf(50));          //премахва първото срещане дадения елемент от списъка
numbers.remove(1);                            //премахва елемента на дадената позиция
```

5. Отпечатване на списък

for цикъл -> необходими са позициите

```
for (int position = 0; position <= numbers.size() - 1; position++) {
System.out.println(numbers.get(position));
}
```

foreach -> необходими са само с елементите, без да се интересуваме от позициите

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

toString

```
System.out.println(numbers.toString());
```

String.join -> само за лист от текстове

```
List<String> names = new ArrayList<>(Arrays.asList("Ivan", "Georgi", "Pesho"));
System.out.println(String.join(", ", names));  //отпечатва елементите с посочения разделител
```

6. Други методи на list

```
numbers.contains(12);     //проверява дали даден елемент е в листа (по стойност)
numbers.isEmpty();        //проверява дали листът е празен
numbers.clear();          //премахва всички елементи в листа
numbers.indexOf(56);      //връща позицията, на която се намира елемента; връща -1 ако няма такъв елемент
size()                    //извежда броя на елементите
add(element)              //добавя елемент в края
add(index, element)       //добавя елемент на определен индекс
remove(element)           //премахва даден елемент
remove(index)             // премахва елемент на определен индекс
contains(element)         //проверява дали списъкът съдържа определен елемент
set(index, item)          //заменя елемент на определен индекс
```

7. Четене на текст от конзолата

```
//34 56 12 45 87

//Стринг към масив
```

```
int[] numbersArray = Arrays.stream(scanner.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();

//int към лист
List<Integer> numbersList = Arrays.stream(scanner.nextLine()
.split(" ")).map(Integer::parseInt).collect(Collectors.toList());

//Стринг към лист
List<String> namesList = Arrays.stream(scanner.nextLine().split(" ")).collect(Collectors.toList());
```

7. Сортиране

7.1. ascending order -> нарастващ ред

```
Collections.sort(numbers);
```

7.1. descending order -> намаляващ ред

```
Collections.sort(numbers); //ascending order
Collections.reverse(numbers); //обратен ред
```

8. Всички методи на ЛИСТ

void add(int index, E element)	Използва се за вмъкване на посочения елемент на посочената позиция в списък.
boolean add(E e)	Използва се за добавяне на посочения елемент в края на списък.
boolean addAll(Collection<? extends E> c)	Използва се за добавяне на всички елементи в указаната колекция в края на списък.
boolean addAll (int index, Collection<? extends E> c)	Използва се за добавяне на всички елементи в посочената колекция, започвайки от посочената позиция в списъка.
void clear()	Използва се за премахване на всички елементи от този списък.
boolean equals(Object o)	Използва се за сравняване на посочения обект с елементите на списък.
int hashcode()	Използва се за връщане на стойността на хеш кода за списък.
E get(int index)	Използва се за извличане на елемента от определена позиция в списъка.
boolean isEmpty()	Връща true, ако списъкът е празен, в противен случай false.
int lastIndexOf(Object o)	Използва се за връщане на индекса в този списък на последното срещане на посочения елемент или -1, ако списъкът не съдържа този елемент.
Object[] toArray()	Използва се за връщане на масив, съдържащ всички елементи в този списък в правилния ред.
<T> T[] toArray(T[] a)	Използва се за връщане на масив, съдържащ всички елементи в този списък в правилния ред.
boolean contains(Object o)	Връща true, ако списъкът съдържа посочения елемент
boolean containsAll(Collection<?> c)	Връща true, ако списъкът съдържа всички посочени елементи
int indexOf(Object o)	Използва се за връщане на индекса в този списък на първото появяване на посочения елемент или -1, ако списъкът не съдържа този елемент.
E remove(int index)	Използва се за премахване на елемента, присъстващ на посочената позиция в списъка.

boolean remove(Object o)	Използва се за премахване на първото появяване на посочения елемент.
boolean removeAll(Collection<?> c)	Използва се за премахване на всички елементи от списъка.
void replaceAll(UnaryOperator<E> operator)	Използва се за замяна на всички елементи от списъка с посочения елемент.
void retainAll(Collection<?> c)	Използва се за запазване на всички елементи в списъка, които присъстват в указаната колекция.
E set(int index, E element)	Използва се за заместване на посочения елемент в списъка, присъстващ на посочената позиция.
void sort(Comparator<? super E> c)	Използва се за сортиране на елементите от списъка на базата на зададен компаратор.
Spliterator<E> spliterator()	Използва се за създаване на сплитератор върху елементите в списък.
List<E> subList(int fromIndex, int toIndex)	Използва се за извличане на всички елементи в дадения диапазон.
int size()	Използва се за връщане на броя елементи, присъстващи в списъка.

9. Сравнение на ЛИСТ и МАСИВ

1. Създаване

```
int [] array = new int[10];                 //задаване брой на елементите
List<Integer> list = new ArrayList<>();     //няма нужда да задаваме брой елементи
```

2. Брой елементи

```
array.length;           //дължина = брой елементи
list.size();            //размер = брой елементи
```

3. Достъп елементи по позиция

```
array[0];
    list.get(0);
```

4. Добавяне елементи

```
list.add(50);           //добавяне на елемента в скобите в края на списъка
list.add(0, 12);        //вмъкване на елемента на дадения индекс. Елементите след него се преместват.
```

5. Обхождане с foreach

```
for (int number: array) {
    System.out.println(number);
}

for (int number: list) {
    System.out.println(number);
}
```

6. Обхождане с for

```
for (int position = 0; position <= array.length - 1; position++) {
    System.out.println(array[position]);
}

for (int position = 0; position <= list.size() - 1; position++) {
    System.out.println(list.get(position));
}
```

//ПРЕДИМСТВА НА ЛИСТ ПРЕД МАСИВ

//1. ПРЕОРАЗМЕРЯВАНЕ - НЯМА НУЖДА ПРЕДВАРИТЕЛНО ДА ЗНАЕМ БРОЯ НА ЕЛЕМЕНТИ

//2. ПО-ФУНКЦИОНАЛЕН = ПО-ЛЕСНО СЕ МОДИФИЦИРА

4. МЕТОДИ

Определение:

Именуван блок от код, който може да бъде използван по-късно

Дефиниране (деклариране)

Без параметри

```
public static void name () {
System.out.println("Hello!");
}
```

С параметри

```
static void printNumbers(int start, int end) {
for (int i = start; i <= end; i++) {
System.out.printf("%d ", i);
} }
```

Използване

printHello (); - без параметри

printNumbers (6, 12); - с параметри

Видове: връщат стойност и не връщат стойност (void),

Overloading: когато няколко метода са с еднакво име, но с различна сигнатура (входни параметри). Всеки метод може да връща различен тип данни.

5. ОБЕКТИ И КЛАСОВЕ

public class Book { .....}

1. Fields – полета

```
//характеристики - конски сили, марка, цвят
//private - достъпваме само в рамките на класа
//public - достъпваме навсякъде в класовете в проекта

private String title;
private String author;
private double price;
```

2. Constructor – конструктор, носи точно името на класа

```
//конструктори - public методи, чрез които създаваме обекти от класа. Носи точно името на класа.
//1. default constructor -> създава празен обект от класа
//2. custom constructor -> създава обект, на който мога да задам стойности на полетата

public Book (String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }
```

3. Methods - действия

```
public void sell() {
        System.out.printf("Book with title: %s was successfully sold for %.2f.", this.title, this.price);
    }
```

6. АСОЦИАТИВНИ МАСИВИ- MAP

```
//създаване на празен map
Map<String,Double> studentsMap = new TreeMap<>();

//видове:
//1. HashMap -> редът на записите не е гарантиран
//2. LinkedHashMap -> редът на записите се запазва спрямо реда на добавяне
//3. TreeMap -> нарежда записите спрямо ключа в нарастващ ред (ascending order)

//добавяме записи в map
studentsMap.put("Ivan", 5.60);
salariesMap.putIfAbsent("Ivan", 3450.50); //добавя ако такъв ключ няма

// Използване на записи от мапа
map.getKey() - взима ключа
map.getValue() – взима стойността

// Брой на записи в мапа
System.out.println(studentsMap.size());

//премахваме записи от map
studentsMap.remove("Petya");            //премахване по ключ, ако го има
studentsMap.remove("Georgi", 3.40);     //премахване на запис, ако го има

//проверка дали map е празен (size = 0)
System.out.println(studentsMap.isEmpty());

//проверка дали съществува запис с даден ключ или стойност
System.out.println(studentsMap.containsKey("Desi"));
System.out.println(studentsMap.containsValue(5.60));

//премахва всички елементи от map
studentsMap.clear();

//Отпечатване
words = ["kiwi", "orange", "banana"]

//1 начин -> StreamAPI
Arrays.stream(words).forEach(word -> System.out.println(word));                 //метод на Arrays.stream
words.entrySet().forEach(entry -> System.out.printf("%s -> %d\n", entry.getKey(), entry.getValue()));        //метод на мапа

//2 начин -> foreach
for (String word : words) {
System.out.println(word);
}

//Едновременно отпечатване на два мапа (един и същ ключ)
map1.entrySet().forEach(entry -> {
        System.out.println(entry.getKey());
        System.out.println("  HP: " + entry.getValue());
        System.out.println("  MP: " + map2.get(entry.getKey()));
    });

// Map от String и вложен List<Double>
```

- **Създаване :** Map<String, List<Double>> plantRatings = new LinkedHashMap<>();
- **Въвеждане на ключа:** plantRatings.put(name, new ArrayList<>());
- **Въвеждане на стойността (List) :** plantRatings.get(name).add(rating);
- **Изчистване на стойността (List):** plantRatings.get(name).clear();

- **Записване на List от Мапа в отделен List:** List<Double> ratings = plantRatings.get(name);
- **Извеждане на сума (средна стойност на елементите на вложен в Мап List:**
  List<Double> ratings = plantRatings.get(name);
  averageRating = ratings.stream().mapToDouble(Double::doubleValue).sum() / ratings.size();

7. Lambda_StreamAPI

API - Application Program Interface

Stream API - съвкупност от методи върху структури данни

2 вида stream / поточна линия / поток

1. Primitive -> IntStream, DoubleStream -> sum, average, min, max

2. Stream -> Stream<String>, Stream<Integer>

- парсване на елементите на масив input

```
int [] numbers = Arrays.stream(input).mapToInt(e -> Integer.parseInt(e)).toArray();
```

- добавяне в променливата min най-малкия елемент на масива или 23

```
int min = Arrays.stream(numbers).min().orElse(23);
```

-филтрира само елементите по – големи от нула и ги добавя към масив

```
int[] nums = Arrays.stream(scanner.nextLine().split(" ")).mapToInt(e -> Integer.parseInt(e)).filter(n -> n > 0).toArray();
```

-филтрира само думите с четен брой букви и ги добавя към масив

```
String[] words = Arrays.stream(scanner.nextLine().split(" ")).filter(word -> word.length(% 2 == 0).toArray(String[] :: new);
```

8. ОБРАБОТКА НА ТЕКСТОВЕ

String name = "Desislava";

1. дължина на текст = брой символи

```
name.length();
```

2. само главни букви

name.toUpperCase(); - преобразува стринга към **главни букви**

3. само с малки букви

name.toLowerCase(); преобразува стринга към **малки букви**

4. Преобразуване на стринг в масив от символи: "Desislava" -> ['D', 'e', 's', 'i', 's', 'l', 'a', 'v', 'a']

```
char [] symbols = name.toCharArray();
```

5. сравняване на текстове -> true, false

```
System.out.println("Desislava".equals(name));                    // проверява еднакви ли са текстовете
System.out.println("DeSIslava".equalsIgnoreCase(name));          // проверява еднакви ли са текстовете без да прави разлика между малки и главни букви
```

6. достъпване символ от текста

```
name.charAt(0);                          //първия символ
name.charAt(name.length() - 1);          //последния символ
```

7. съдържа определен текст -> true (ако се съдържа), false (ако не се съдържа)

```
name.contains("Des");
```

8. започва с опередален текст -> true (ако започва), false (ако не започва)

```
name.startsWith("De");
```

9. завършва на определен текст -> true (ако завършва), false (ако не завършва)

(name.endsWith("lava");

10. премахва интервалите в началото и края на текста

„ Desislava ".trim();

11. заменя първото срещане на даден текст / символ (да се провери, май заменя всички срещания на текста)

name.replace("va", "ta");

name.replace('v', 'r');

12. заменя всички срещания на текста

name.replaceAll("a", "b");

13. повтаря текста даден брой пъти

name.repeat(5);

14. Подтекст (текст, който е част от друг текст)

name.substring(2); //от посочената позиция до края

name.substring(1, 5); - //Между посочените позиции. Вторият индекс не се включва – до него, но без него. Първият се включва

15. Обхождане на текст

//обходим един текст от първия към последния символ

for (int position = 0; position <= name.length() - 1; position++) {

char Symbol = name.charAt(position);

System.out.println(Symbol);

}

//обходим един текст от последния към първия символ

for (int position = name.length() - 1; position >= 0; position--) {

char Symbol = name.charAt(position);

System.out.println(Symbol);

}

//конкатенация -> долепяне / събиране на два текста

//1. чрез оператора +

String result = "Desi" + " " + "Topuzakova";

//2. чрез метода concat

String concatResult = "Ivan".concat(" ").concat("Ivanov");

//Join - обединяване

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

String[] textsArray = new String[] {"Ivan", "Georgi", "Peter"};

System.out.println(String.join("-", textsArray));

//Split - разделяне

String input = "Ivan-Peter-John-George";

String [] words = input.split("-");

//Input - вмъкване

String partInput = input.substring(0, 4); //position: [0;4)

String partInput2 = input.substring(8); //position: [8; length - 1]

//Searching -> indexOf, lastIndexOf, contains

String fruits = "banana, apple, kiwi, banana, apple";

System.out.println(fruits.indexOf("banana"));

System.out.println(fruits.lastIndexOf("apple"));

System.out.println(fruits.contains("kiwi")); //true

```
System.out.println(fruits.contains("pineapple"));              //false

// Repeat - повтаряне на текст
String animal = "turtle";
System.out.println(animal.repeat(5));

// Replace – замяна на текст
//замяна на всички срещания -> replace
//замяна на първото срещане -> replaceFirst
String lastNamePeople = "Ivanov Ivanov Petrov Georgiev";
lastNamePeople = lastNamePeople.replace("Ivanov", "Petkov");
```

//**Всички методи на String тук:** https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

```
//EXAMPLE
String test = "I am enjoying programming";
System.out.println(test.substring(5, 13));
String text = "enjoying";
int index = test.indexOf(text); //индексът на първата буква на текста = 5
System.out.println(test.substring(index, index + text.length()));
```

9. STRING BUILDER

Използва се винаги, когато има операции върху стрингове – долепяне и др. лесно се модифицира текста, бърз

```
StringBuilder sb = new StringBuilder();              //празен string builder
StringBuilder sb = new StringBuilder("Desi");

//добавяне на текст
sb.append(" Topuzakova");

//използване на текста
String text = sb.toString();

//изтривне на елементи от текст
sb.delete(3, 9);              //изтрива текста между посочените позиции

//обръщане на текст
sb.reverse();

//Дължина на текст
sb.length();

//вмъкване в текст
sb.insert(0, "Ivan");                 //вмъква текста на посочената позиция

//достъпване на символ в текста в StringBuilder
Char char = sb.charAt(0);          //взима символа на посочената позиция
```

//**Всички методи на StringBuilder** тук: https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html

10. REGULAR EXPRESSIONS- REGEX

Основен синтаксис:

[A-Z] - една главна буква (аски код от 65 до 90)

[a-z] - една малка буква (аски код от 97 до 120)

[0-9] - една цифра [0-9] (аски код от 48 до 57)

[A-Za-z] - една буква, която или е малка, или е голяма

[aeiou] - всички гласни букви

[^aeiou] - всички съгласни букви

\w - един символ, който може да е малка буква, главна буква, цифра или _

\W - един символ, различен от малка буква, главна буква, цифра или _

\s - един интервал

\S - един символ, различен от интервал

\d - една цифра [0-9] (аски код от 48 до 57)

\D - един символ, различен от цифра

() -> обособяваме група. Името е автоматично, започва от 1 .....

(?<name>) -> обособяваме група с име

\b -> слагаме граница, която казва, че не искаме да има символи(букви/цифри) преди/след съвпадението, което е открито в текста.

Брой на срещанията:

* -> срещания 0 или безброй много пъти

+ -> срещания 1 или безброй много пъти

? -> срещания 0 или 1 пъти

{число} -> срещания {число} пъти

{число, } -> минимум колко пъти

{число1, число2} -> минимум се среща число1 пъти, максимум се среща число2 брой пъти

() -> обособяваме група

(?<name> шаблон) -> обособяваме група с име

Използване в Java:

//Прочитане на текста от конзолата

String text = scanner.nextLine();

// 1. Създаване на текст на шаблона (критерии за търсене, шаблон. Проверява се в сайта https://regex101.com/)

String regex = "\\+359([-])2\\1\\d{3}\\1\\d{4}\\b";

// 2. Създаване на Java шаблон

Pattern pattern = Pattern.compile(regex);

// 3. Създаване на инструмент с помощта на който ще проверяваме за съвпадения в прочетеният текст от конзолата. Използва се класа Matcher.

Matcher matcher = pattern.matcher(text);

// 4. Създаване на Списък за запазване на откритите съвпадения

List<String> validNumbers = new ArrayList<>();

//5. Проверяване за съвпадения в текста, използват се методи на класа Matcher

- find – връща true ако намери съвпадение и обратно
- group – взима намереното съвпадение

while (matcher.find()){

validNumbers.add(matcher.group())}

System.out.println(String.join(", ", validNumbers))}

//6. Използване на части от съвпаденията

- разделяне на регекса на групи

String regex = "\\b(?<day>\\d{2})([\\.\\-\\/])(?<month>[A-Z][a-z]{2})\\2(?<year>\\d{4})\\b"; //13/Jul/1928

- използване на отделните части на съвпаденията

String wholeMatch = matcher.group(); //"13/Jul/1928"

String day = matcher.group("day"); или String day = matcher.group(1); //"13"

String month = matcher.group("month"); или String month = matcher.group(3); //"Jul"

String year = matcher.group("year"); или String year = matcher.group(4); //"1928"

11. СТЕКОВЕ И ЗАЯВКИ (Stacks and Queues)

11.1 Стекове – последен влязъл, първи излязъл

- създаване: ArrayDeque stack = new ArrayDeque<>();

- добавяне на елемент на върха: stack.push(element);

- използване на елемент: Integer element = stack.peek(); //взима стойността на последния елемент без да го изтрива от стека

- използване на елемент с изтриване: Integer element = stack.pop(); //взима стойността на последния влязъл елемент и го изтрива от стека

- **празен ли е стека:** stack.isEmpty();

- **размер на стека:** stack.size();

- съдържа ли стека определен елемент: stack.contains(2);

- директно пълнене на стек с ред от конзолата:

```
ArrayDeque<Integer> stack = new ArrayDeque<>();
Arrays.stream(scanner.nextLine().split("\\s+")).map(Integer::parseInt).forEach(stack::push);
```

-Обърнато разпечатване на стек

```
List<Integer> reversedStack = new ArrayList<>(stack);
Collections.reverse(reversedStack);
// Стек с цели числа!
// .map(String::valueOf) -> взема едно число и го преобразува на текст
String output = reversedStack.stream().map(String::valueOf).collect(Collectors.joining(", "));
System.out.println(output);
```

11.2 Опашки – първи влязъл, първи излязъл

- създаване: ArrayDeque queue = new ArrayDeque<>(); //създава се по същия начин като стека – структурата данни е една и съща, просто се използват различни методи.

- добавяне на елемент: queue.offer(element);

- използване на елемент с изтриване: element = queue.poll(); // взима стойността на първия влязъл елемент и го изтрива от опашката

- използване на елемент без изтриване: element = queue.peek(); //// взима стойността на първия влязъл елемент без да го изтрива от опашката

- размер на опашката: Integer size = queue.size();

- прехвърляне към масив: Integer[] arr = queue.toArray();

- съдържа ли определен елемент: boolean exists = queue.contains(element);

- празна ли е опашката: queue.isEmpty();

- директно пълнене на опашка с ред от конзолата:

```
ArrayDeque<Integer> queue = new ArrayDeque<>();
Arrays.stream(scanner.nextLine().split("\\s+")).map(Integer::parseInt).forEach(queue::offer);
```

11.3 Приоритетни опашки – първи влязъл, първи излязъл, подредени по определен ред

- създаване: PriorityQueue queue = new PriorityQueue <>();

- останалите методи са аналогични на опашките

12. МНОГОМЕРНИ МАСИВИ (Multidimensional Arrays)

1. Създаване: int[][] matrix = new int[4][4];

2. Прочитане на матрица от конзолата:

```
Scanner scanner = new Scanner(System.in);
int[] input = Arrays.stream(scanner.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
int rowNumber = input[0];
int colNumber = input[1];
int[][] matrix2 = new int[rowNumber][colNumber];

// ВАРИАНТ 1:
for (int row = 0; row < rowNumber; row++) {
    for (int col = 0; col < colNumber; col++) {
        matrix2[row][col] = scanner.nextInt();
```

```
        }
    }

    // ВАРИАНТ 2:
    for (int row = 0; row < rowNumber; row++) {
        int[] currentRow = Arrays.stream(scanner.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
        matrix2[row] = currentRow;
    }
```

3. Обхождане на матрица

```
    for (int row = 0; row < matrix.length; row++) {
        for (int col = 0; col < matrix[row].length; col++) {
            System.out.print(matrix[row][col] + " ");
        }
        System.out.println();
    }
```

4. ГОТОВИ МЕТОДИ ЗА МАТРИЦИ:

4.1. Печатане на матрица:

```
private static void printMatrix(String[][] matrix) {
    for (int row = 0; row < matrix.length; row++) {
        for (int col = 0; col < matrix[row].length; col++) {
            System.out.print(matrix[row][col] + " ");
        }
        System.out.println();
    }
}
```

4.2. Сума на главен диагонал

```
private static int getPrimaryDiagonalSum(int[][] matrix) {
    int sum = 0;
    // Локацията е: индекс на реда == индекс на колона
    for (int index = 0; index < matrix.length; index++) {
        int num = matrix[index][index];
        sum += num;
    }
    return sum;
}
```

4.3. Сума на втори диагонал

```
private static int getSecondaryDiagonalSum(int[][] matrix) {
    int sum = 0;
    // Ред: от последен към 0
    // Колона: 0 докато има редове (всеки път +1)
    int col = 0;
    for (int row = matrix.length - 1; row >= 0; row--) {
        int num = matrix[row][col];
        sum += num;
        col++;
    }
    return sum;
}
```

4.4. Попълване на матрица

```
private static void fillTheMatrix(int[][] matrix, Scanner scanner) {
```

```
    for (int row = 0; row < matrix.length; row++) {
        for (int col = 0; col < matrix[row].length; col++) {
            matrix[row][col] = scanner.nextInt();
        }
    }
}
```

4.5 Завъртане на матрица по посока на часовниковата стрелка

```
    private static char[][] rotateMatrix90(char[][] oldMatrix) {

        // Редовете == броя на колониите на първият ред от старата матрица
        // Колоните == броя на редовете от старата матрица
        int newRows = oldMatrix[0].length;
        int newCols = oldMatrix.length;
        char[][] newMatrix = new char[newRows][newCols];

        // Обхождам СТАРАТА матрица и местя елементите в НОВАТА!
        // Колона: 0 към последна
        // Ред: последн към 0
        for (int col = 0; col < oldMatrix[0].length; col++) {
            int counter = 0;
            for (int row = oldMatrix.length - 1; row >= 0; row--) {
                char symbol = oldMatrix[row][col];
                newMatrix[col][counter] = symbol;
                counter++;
            }
        }
        return newMatrix;
```

4.6 Максимална сума на елемент 3x3

```
private static void fillTheMatrix(int[][] matrix, Scanner scanner) {
        int maxSum = Integer.MIN_VALUE;
        int maxMatrixRow = 0;
        int maxMatrixCol = 0;

for (int row = 0; row < matrix.length - 2; row++) {
    for (int col = 0; col < matrix[row].length - 2; col++) {

        int sum3x3 = matrix[row][col] + matrix[row][col + 1] + matrix[row][col + 2] +
                matrix[row + 1][col] + matrix[row + 1][col + 1] + matrix[row + 1][col + 2] +
                matrix[row + 2][col] + matrix[row + 2][col + 1] + matrix[row + 2][col + 2];

        if (sum3x3 > maxSum){
            maxSum = sum3x3;
            maxMatrixRow = row;
            maxMatrixCol = col;
        }
    }
}
System.out.printf("Sum = %d\n", maxSum);
printMatrix(matrix, maxMatrixRow, maxMatrixCol);
```

4.7. Проверка дали дадена позиция (по индекси) е в рамките на матрицата

```
public static boolean isInBounds(int r, int c, char[][] board) {
    return r >= 0 && r < board.length && c >= 0 && c < board[r].length;
}
```

13. СЕТОВЕ И КАРТИ (Sets and Maps)

```
Set<Integer> numbers = new LinkedHashSet<>();
// new HashSet<>()       -> не пази конкретна подредба на елементите, много бърз
// new TreeSet<>()       -> запазва елементите в нарастващ ред
// new LinkedHashSet<>() -> запазва елементите в реда на тяхно добавяне

// ВАЖНО: Сет -> Пази само уникални елементи
// ВАЖНО: Сет -> Не може да достъпваме елементи по индекс
numbers.add(4);  - добавяне на елемент
numbers.size() - размер
numbers.contains(5) – съдържа ли
```

14. ПОТОЦИ, ФАЙЛОВЕ И ДИРЕКТОРИИ (Streams, Files and Directories)

```
// 1. BufferedReader:
    // Как прочита файл: Ефективно четене на редове, добър за четене на големи файлове поради буферната памет, която използва.
    // Обичайна употреба: Четене на големи текстови файлове ред по ред.
    BufferedReader bufferedReader = new BufferedReader(new FileReader("java-advanced/04-streams-files-and-directories/02-exercises/input.txt"));
    System.out.println(bufferedReader.readLine());

    // 2. Scanner:
    // Как прочита файл: Лесен за използване, но може да бъде по-бавен и не толкова ефективен при големи файлове.
    // Обичайна употреба: Четене от конзолата.
    Scanner scanner = new Scanner(new File("java-advanced/04-streams-files-and-directories/02-exercises/input.txt"));
    System.out.println(scanner.nextLine());

    // 3. Четене с FileReader:
    // Как прочита файл: Прочита файл символ по символ. Не е ефективен за големи файлове.
    // Обичайна употреба: Четене на много малки текстови файлове
    FileReader fileReader = new FileReader("java-advanced/04-streams-files-and-directories/02-exercises/input.txt");
    System.out.println((char) fileReader.read());

    // 4. Четене с Files.readAllLines:
    // Как прочита файл: Чете всички редове наведнъж, лесен за използване, използва BufferedReader
    List<String> allLines = Files.readAllLines(Path.of("java-advanced/04-streams-files-and-directories/02-exercises/input.txt"));
    System.out.println(allLines.get(0));

    // 5. Четене с Files.readString:
    // Как прочита файл: Чете цялото съдържание на файла наведнъж, връща String. Лесен за използване!
    String fullContent = Files.readString(Path.of("java-advanced/04-streams-files-and-directories/02-exercises/input.txt"));
    System.out.println(fullContent);


    // 1. Писане с PrintWriter:
    // Лесен за използване, има добре познати методи като print(), println(), printf()
    // Изисква изрично извикване на flush или close за да се запишат данните, които се опитваме да напишем на файла.
```

```
PrintWriter printWriter = new PrintWriter("output.txt");
printWriter.println("Hello, World!");
printWriter.println("Another line.");
// Затватяме потока от информация към този файл и всички наши промени ще се отразят във файла
printWriter.close();

// 2. Писане с BufferedWriter:
// Ефективен за писане на големи количества данни благодарение на буфериране.
// Изисква изрично извикване на flush или close за да се запишат данните, които се опитваме да напишем на файла.
BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter("output.txt"));
bufferedWriter.write("Hello, World!");
bufferedWriter.newLine(); // Добавя нов ред
bufferedWriter.write("Another line.");
bufferedWriter.close();

// 3. Писане с FileWriter:
    // Прост и директен начин за писане по файлове, като данните ще се записват символ по символ, както става четенето при FileReader класа
    // Не ползва буферна памет, по-малко ефективен за големи файлове
    FileWriter fileWriter = new FileWriter("output.txt");
    fileWriter.write("Hello, World!\n");
    fileWriter.write("Another line.\n");
    fileWriter.close();
```

15. ФУНКЦИИ (Functional Programming)

```
// Imperative/Structured way of programming
    List<String> names = Arrays.asList("Ivan", "Gosho", "Tosho", "Ivana");
    for (String name : names) {
       if (name.startsWith("I")) {
          System.out.println(name);
       }
    }

    // Functional way of programming (Using Lambda expression)
```

```
// Predicate: аргумент -> израз/действие, който връща булева стойност
 // Consumer: аргумент -> извършва се действие без да се връща резулт
// Function: аргумент -> извършва се действие с този аргумент и връща резултат
// Supplier: () -> извършва действие и връща резултат
// BiFunction: (аргумент1, аргумент2) -> извършва се действие с тези два аргумента и връща резултат

names.stream()
  .filter(name -> name.startsWith("I")) // Predicate
  .map(name -> name.toUpperCase()) // Function
  .forEach(name -> System.out.println(name));// Consumer (terminal operation)

Supplier<Integer> RandomNumber = () -> new Random().nextInt();
System.out.println(aRandomNumber.get());

BiFunction<String, String, String> concatenateBiFunction = (name1, name2) -> name1 + " " + name2;
System.out.println(concatenateBiFunction.apply("Viktor", "Aleksandrov"));
```

16. КЛАСОВЕ (Classes)

```
public class Book {

    // 1. Fields: винаги са private
            private String title;
            private String author;
            private double price;

    // 2. Constructor: creates new objects
    public Book(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }

    // 3. Methods: actions
        public void sell() {
            System.out.printf("Book with title %s was sold for %.2f", this.title, this.price);
        }
        public void setTitle(String newTitle) {
            this.title = newTitle;
        }
        public String getTitle() {
            return this.title;
        }
}
```

17. ПАРАМЕТРИЗИРАНИ ТИПОВЕ (Generics)

- Generics Syntax;
- Generic Classes and Interfaces;
- Generic Methods;
- Type Erasure, Type Parameter Bounds.

18. ИТЕРАТОРИ И КОМПАРАТОРИ (Iterators and Comparators)

- Variable Arguments;
- Iterators (Iterator, ListIterator);
- Comperators (Comparable).

19. АБСТРАКЦИИ (Abstraction)

- Project Architecture;
- Code Refactoring;
- Enumerations;
- Static Keyword;
- Java Packages.

20. ЕНКАПСУЛАЦИЯ (Encapsulation)

- Keyword this;
- Access Modifiers;
- Mutable and Immutable Objects;
- Keyword final;
- Validation.

21. НАСЛЕДЯВАНЕ (Inheritance)

- Inheritance;
- Class Hierarchies;
- Accessing Base Class Members;

- Reusing Classes;
- Type of Class Reuse.

22. ИНТЕРФЕЙСИ И АБСТРАКЦИЯ (Interfaces and Abstraction)

- Abstraction;
- Interfaces;
- Abstract Classes;
- Interfaces vs Abstract Classes.

23. ПОЛИМОРФИЗЪМ (Polymorphism)

- Polymorphism;
- Override Methods;
- Overload Methods.

24. SOLID

- Single Responsibility;
- Open/Closed;
- Liskov Substitution;
- Interface Segregation;
- Dependency Inversion.

25. РЕФЛЕКСИЯ И АНОТАЦИЯ (Reflection and Annotation)

- Reflection API;
- Reflacting Annotations.

26. ИЗКЛЮЧЕНИЯ И ПРЕХВАЩАНЕ НА ГРЕШКИ (Exceptions and Error Handling)

- What are Exceptions?;
- Handling Exceptions;
- Raising (Throwing) Exceptions;
- Best Practices;
- Creating Custom Exceptions.

27. ТЕСТВАНЕ (Testing)

- Unit Testing Basics;
- Dependancy Injection;
- Mocking and Mock Objects.

28. Test Driven Development

- Code and Test;
- Test-Driven Development;
- Reasons to use TDD;
- Myths and Misconceptions about TDD.

29. Design Patterns

- Definition of Design Patterns;
- Benefits and Drawbacks;
- Types of Design Patterns.

30. ЛИНЕЙНИ СТРУКТУРИ ОТ ДАННИ – ОБОБЩЕНИЕ

	Масив	Списък	Асоциативни масиви - MAP	Сетове				Текстови
Видове			1. HashMap -> редът на записите не е гарантиран 2. LinkedHashMap -> редът на записите се запазва спрямо реда на добавяне 3. TreeMap -> нарежда записите спрямо ключа в нарастващ ред (ascending order)					
Създаване	int [] array = new int[10]; //задаваме брой на елементите	List<Integer> list = new ArrayList<>(); //няма нужда да задаваме брой елементи	Map<String,Double> studentsMap = new TreeMap<>();	Set<Integer> numbers = new LinkedHashSet<>();			**Създаване**	String string =
Брой елементи	array.length();	list.size();	Map.size();	**Set.size()**			**Брой елементи**	String.length(
Използване на елементи	array[0];	list.get(0);	map.getKey() - взима ключа map.getValue() – взима стойността				**Използване на елементи**	string.charAt(//връща сим посочената п

Добавяне елементи	array[1] = 5;	list.add(50);
		//добавяме елемента в скобите в края на списъка
		list.add(0, 12);
		//вмъквате елемента на дадения индекс
			Map.put("Ivan", 5.60);
			Map.putIfAbsent("Ivan", 3450.50);
			//добавя ако такъв ключ няма
				Set.add()			**Само главни букви**	string.toUppe
								- преобразув
								главни букви

Премахване на елементи		**remove** (int index)	Map.remove("Petya");
			//премахване по ключ, ако го има
			Map.remove("Georgi", 3.40);
			//премахване на запис, ако го има
			studentsMap.clear();
			//премахва всички елементи
							Само малки букви	string.toLowe
								- преобразув
								малки букви

Обхождане с foreach	for (int number: array) {
	System.out.println(number);
	}
		for (int number: list) {
		System.out.println(number);
		}
			for (String word : words) {
			System.out.println(word);
			}
							Обхождане от първия към последния символ	for (int positi
								name.length(
								char currentS
								name.charAt(
								System.out.p
								l);
								}

Обхождане с for	for (int position = 0; position <= array.length - 1; position++) {
	System.out.println(array[position]);
	}
		for (int position = 0; position <= list.size() - 1; position++) {
		System.out.println(list.get(position));
		}
							Обхождане от последния към първия символ	for (int positi
								- 1; position >
								char currentS
								name.charAt(
								System.out.p
								l);
								}

Обхождане със StreamAPI			Arrays.stream(words).forEach(word -> System.out.println(word));
			//метод на Arrays.stream
			words.entrySet().forEach(entry -> System.out.printf("%s -> %d\n", entry.getKey(), entry.getValue()));
			// метод на мапа
							Добавяне Конкатенация -> долепяне	//1. чрез опе
								String result =
								"Topuzakova"
								//2. чрез мет
								String concat
								"Ivan".concat
								").concat("Iva

Проверка дали е празен		list.size() ==0;	Map.isEmpty()	**Set.isEmpty**			**Join - обединяване**	String text = "ca", "ten", "a String[] textsA {"Ivan", "Geor System.out.p textsArray));
Методи							**Split - разделяне**	String text = George"; String [] word
Вмъкване на посочения елемент на посочената позиция в списък.		void **add**(int index, E element)					**Substring**	String text = 4); Между посо String text = От посочена края
Премахване на всички елементи от този списък.		void **clear**()					**Преобразуване в масив от символи:**	char [] symbo name.toChar
Извличане на елемента от определена позиция в списъка.		E **get** (int index)	keySet() -> връща всички ключове от всички записи entrySet() -> връща колекция от всички записи get(ключ) -> връща стойността, която стои срещу дадения ключ				**Сравняване на текстове -> true, false**	Desislava".eq проверява е текстовете "DeSIslava".e me); проверява е текстовете б разлика меж букви
Връща true, ако списъкът е празен, в противен случай false.		boolean **isEmpty**()					**Дали съдържа определен текст**	string.contain -> true (ако (ако не се съ
Връща true, ако списъкът съдържа посочения елемент		boolean **contains**(Object o)	containsKey(key) -> проверява дали в map-а има запис с такъв ключ -> резултат true ако има, false ако няма containsValue(value)				**Дали започва с опеределен текст**	string.startsW -> true (ако не започва)

			-> проверява дали в map-а има запис с такова value -> резултат true ако има, false ако няма					
Премахване на елемента, присъстващ на посочената позиция в списъка.		E **remove**(int index)					**Дали завършва на определен текст**	string.endsW -> true (ако (ако не завъ
Замяна на всички елементи от списъка с посочения елемент.		void **replaceAll**(UnaryOperator<E> operator)					**Премахване на интервалите в началото и края на текста**	(" Desislava
Заместване на посочения елемент в списъка, присъстващ на посочената позиция.		E **set**(int index, E element)					**Заменя първото срещане на даден текст**	name.replace str); Заменя първ
Връщане на броя елементи, присъстващи в списъка.		int **size**()					**Заменя всички срещания на текста**	name.replace str); Заменя стри
Връща позицията, на която се намира елемента; връща -1 ако няма такъв елемент		list.**indexOf**(56));					**Повтаряне на текста даден брой пъти**	name.repeat(
Добавяне на посочения елемент в края на списък.		boolean add(E e)					**Текст, който е част от друг текст**	name.substri //връща под посочената name.substri //връща под първата поз (без нея)
Добавяне на всички елементи в указаната колекция в края на списък.		boolean addAll(Collection<? extends E> c)					**Търсене -> indexOf, lastIndexOf, contains**	string.indexO Връща първ намерено съ string.lastInd Връща после намерено съ fruits.contain Връща true, съвпадение
Добавяне на всички елементи в посочената колекция, започвайки от посочената позиция в списъка.		boolean addAll (int index, Collection<? extends E> c)					**Изтриване**	
Сравняване на посочения обект с елементите на списък.		boolean equals(Object o)					**Обръщане на текст**	
Връща стойността на хеш кода за списък.		int hashcode()						
Връща индекса в този списък на последното срещане на посочения елемент или -1, ако списъкът не съдържа този елемент.		int lastIndexOf(Object o)						
Връща масив, съдържащ всички елементи в този списък в правилния ред.		Object[] toArray()						
Връща масив, съдържащ всички елементи в този списък в правилния ред.		<T> T[] toArray(T[] a)						
Връща true, ако списъкът съдържа всички посочени елементи		boolean containsAll(Collection<?> c)						
Връща индекса в този списък на първото появяване на посочения елемент или -1, ако списъкът не съдържа този елемент.		int indexOf(Object o)						
Премахване на първото появяване на посочения елемент.		boolean remove(Object o)						
Премахване на всички елементи от списъка.		boolean removeAll(Collection<?> c)						
Запазване на всички елементи в списъка, които присъстват в указаната колекция.		void retainAll(Collection<?> c)						
Сортиране на елементите от списъка на базата на зададен компаратор.		void sort(Comparator<? super E> c)						
Създаване на сплитератор върху елементите в списък.		Spliterator<E> spliterator()						
Извличане на всички елементи в дадения диапазон.		List<E> subList(int fromIndex, int toIndex)						