За изпита: 1. Линейна структура от данни, 2. Матрица, 3. Defining classes

1. **Stack and Queue (10.05.2023)**

* От упражненията не съм решил 8-10 (\*)

**Stack (FILO – First In, Last Out)**

**Creating a Stack:**

* ArrayDeque<Integer> stack = new ArrayDeque<>();

**Adding elements at the top of the stack:**

* stack.push(element);

**Removing elements:**

* Integer element = stack.pop();

Картина, която съдържа текст, екранна снимка, софтуер, Мултимедиен софтуер

Описанието е генерирано автоматично**Getting the value of the topmost element:**

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

**Queue (FIFO – First In, First Out)**

**Creating a Queue:**

* ArrayDeque<Integer> queue = new ArrayDeque<>();

**Adding elements at the end of the queue:**

* queue.add(element) – throws exception if queue is full
* queue.offer(element) – returns false if a queue is full

**Removing elements:**

* element = queue.remove(); - throws exception if queue is empty
* element = queue.poll(); - returns null if queue is empty – елемента излиза и се изтрива от опашката (като pop() при стека)

**Check first element**

element = queue.peek();

**Priority Queue**

Картина, която съдържа текст, екранна снимка, софтуер, Компютърна икона

Описанието е генерирано автоматично

Картина, която съдържа текст, екранна снимка, софтуер, Мултимедиен софтуер

Описанието е генерирано автоматично

1. **Multidimensional Arrays (12.05.2023)**

* Не съм решил 7 и 8 задача от лекциите (не съм чел дори условията)
* 9 и 10 задача да реша най-накрая в модула (много са трудни)
* Многомерни масиви (матрици)
* При тях задължително инициализираме редовете (първите квадратни скоби), а числото, указващо номера на колоните (вторите []) можем да го добавим на по-късен етап.

Картина, която съдържа текст, екранна снимка, софтуер, Компютърна икона

Описанието е генерирано автоматично

**Creating a multidimensional array**

* int[][] intMatrix = new int[3][];
* float[][] floatMatrix = new float[8][2];
* String[][][] stringCube = new String[5][5][5];
* *Use new keyword*
* *Must specify the size of at least one dimension*

Литерал – когато инициализираме променлива и веднага и задаваме дадена стойност (String b = “Gosho”)

**Initializing a multidimensional array with values:**

int[][] matrix = {

|  |  |  |
| --- | --- | --- |
|  | **0** | **1** |
| **0** | **1** | **2** |
| **1** | **3** | **4** |

{1, 2, 3, 4}, // row 0 values

{5, 6, 7, 8} // row 1 values

};

* *Matrices are represented by a list of rows*
* *Each row consists of a list of values*

**Getting element value example:**

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

int element = array[1][1]; // element11 = 4

**Setting element value example:**

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;

**Reading a Matrix – Example**

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int rows = Integer.parseInt(scanner.nextLine());

int cols = Integer.parseInt(scanner.nextLine());

int[][] matrix = new int[rows][cols];

for (int row = 0; row < rows; row++) {

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

for (int column = 0; column < cols; column++) {

matrix[row][column] =

Integer.parseInt(inputTokens[column]);

}

}

}

**Четене на масив от числа на един ред (припомняне)**

int[] rowOfMatrix = Arrays.stream(scan.nextLine().split(", ")).mapToInt(Integer::parseInt).toArray();

**Намиране на броя на редовете в матрицата:**

int rows = matrix.length();

**Намиране на броя на колоните в матрицата:**

int cols = matrix[rows].length;

1. **Sets and Maps (16.05.2023)**

**Set**

* Множество от уникални елементи с един и същи тип.
* Работят много бързо
* Add/remove/search
* **HashSet<E>** - не гарантира подредбата на елементите
* **TreeSet<E>** - елементите са подредени във възходящ ред според типа данни
* **LinkedHashSet<E>** - пази елементите в реда, в който са подадени

**Initialization:**

Set<String> hash = new HashSet<String>();

Типа вдясно не е задължителен. Ако не сме го написали, типа се подразбира според написания вляво на инициализацията.

.size()

.isEmpty()

* Можем да ги използваме и за филтриране на елементите. Създаваме нов LinkedHasSet, в скобите на който добавяме името на сета, който искаме да филтрираме.

**HashMap**

.put(key, value);

**Хешираща функция** – взима нещо и го превръща в число

* Маповете са много бързи точно заради хеширащата функция. Затова ще ги използваме много често.

**Колизия (сблъсък)** – когато 2 елемента трябва да бъдат добавени в една и съща клетка на ХешМапа. Тогава се създава свързан списък в съответната клетка. Затова трябва да имаме много добра хешираща функция (за да избегнем тази колизия).

**Превръщане на числа, подадени през конзолата, в LinkedHashSet**

Картина, която съдържа текст, екранна снимка, софтуер, Компютърна икона

Описанието е генерирано автоматично

**LinkedHashMap**

Картина, която съдържа текст, диаграма, План, схематичен

Описанието е генерирано автоматично

**Хеширането** се използва за взимане и поставяне на елементи, а списъка, който се получава вдясно с next и previous, го използваме, за да достъпваме елементи и да принтираме данни.

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

**2 начина за принтиране на Map**

Картина, която съдържа текст, екранна снимка, софтуер, Компютърна икона

Описанието е генерирано автоматично

**Sorting**

Картина, която съдържа текст, екранна снимка, софтуер, дисплей

Описанието е генерирано автоматично

* Сортиращите функции очакват да им бъде върнато число:
* 0 – и двата елемента са равни
* Отрицателно – левият елемент е по-малък от десния и трябва да седи по-наляво в мапа
* Положително – левият елемент е по-голям и трябва да бъдат разменени с десния

**Сортиране във възходящ ред -> left.compareTo(right)**

Картина, която съдържа текст, екранна снимка, софтуер, дисплей

Описанието е генерирано автоматично

**Сортиране в низходящ ред -> right.compareTo(left)**

**Множествено сортиране:**

Картина, която съдържа текст, екранна снимка, софтуер, Компютърна икона

Описанието е генерирано автоматично

* Тук в низходящ ред сортираме по оценки и когато има повече от 1 човек с еднакви оценки, пускаме още едно сортиране – по имена, във възходящ ред.

**firstSet.retainAll(secondSet)** – във firstSet се премахват всички елементи, които ги няма в secondSet

1. **Streams, Files and Directories (20.05.2023)**

Файловите потоци са 2 типа: **FileInputStream** (за **четене** на данни от файла **byte по byte** – бавно действие е това) и **FileOutputStream** (за писане на данни във файла)

FileReader (Scanner) и FileWriter (PrintWriter) – **четене символ по символ**. Следователно тук можем да четем всякакви езици, не само на английски.

**BufferedReader (четене) и BufferedWriter(писане**) – взимаме няколко байта наведнъж

**Files** – клас, който има редица полезни методи за работа с пътища

* readAllLines -> връща списък с всички редове
* write -> пише във файла
* работи пряко Path Paths – представят пътища на нашия компютър
* File класът ни помага за лесна работа с файлове и директории

**Сериализация** – когато искаме да запишем даден обект във файл

* **ПОТОЦИТЕ СЕ ЗАТВАРЯТ СЛЕД ТЯХНОТО ИЗПОЛЗВАНЕ!**

1. **Функционално програмиране**

Да потърся видеото с решението на 8 задача – обяснение за comparator, както и да реша 10-та задача!

**Function<T, R>**

* In Java **Function<T,R>** is an interface that accepts a parameter of type **T** and returns a variable of type **R**
* We use function with **.apply()**

*Function<Integer, Integer> increment = number -> number + 1;*

*int a = increment.apply(5);*

*int b = increment.apply(a);*

**UnaryOperator<T>** - същото като функция, но приема и връща един и същи тип елемент

**Special Functions**

**Consumer<T>**  – приема типа на използвания параметър и **не** връща резултат

* We use a Consumer with **.accept()**:

*Consumer<String> print = message -> System.out.print(message);*

*print.accept("Peter");*

**Supplier<T>** – **не** приема параметър, а типа на **връщания** резултат

* We use a Supplier with .**get():**

*Supplier<Integer> genRandomInt = () -> new Random().nextInt(51);*

*int rnd = genRandomInt.get();*

**Predicate<T>** – приема типа на използвания параметър. В тялото на предиката се описва условие, след изпълнението на което, самият предикат връща true/false.

* We use the Predicate with **.test():**

*Predicate<Integer> isEven = number -> number % 2 == 0;*

*System.out.println(isEven.test(6)); // true*

**BiFunction <T, U, R> –** приема 2 аргумента и връща 3-ти тип данни

Analogically you can use:

**BiConsumer <T, U>**

**BiPredicate <T, U>**

Картина, която съдържа текст, софтуер, Компютърна икона, Мултимедиен софтуер

Описанието е генерирано автоматично

1. **Defining classes**

Garbage collector-а следи за обекти на класове, които не се използват в програмата, за да ги премахне и да не заемат излишна памет.