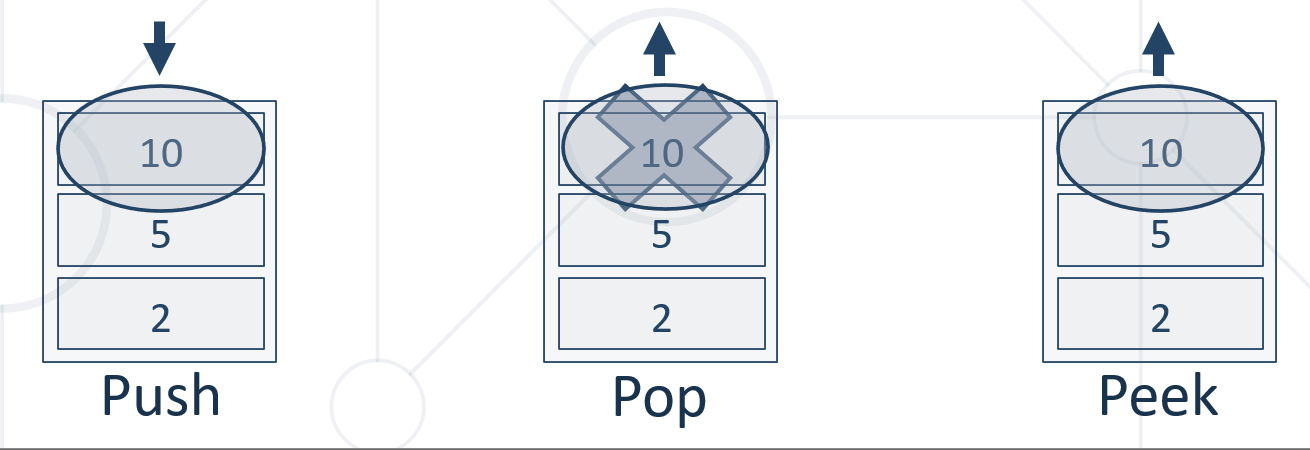
# Java Advanced – Cheat Sheet

# Стек (Stack)

* **LIFO (Last In, First Out)**
* **Последният поставен елемент, излиза първи**
* Визуализация: **под формата на купчина книги**
* Основни методи:
  + **push (елемент)**: добавя елемент на върха на стека
  + **pop()**: премахва и връща елемента от върха на стека
  + **peek()**: връща елементът, който е на върха на стека, но не го премахва
  + **size()**: връща цяло число, което показва колко е размерът на стека (броят на елементите)
  + **isEmpty()**: връща **true**, ако е празен стекът и връща **false**, ако стекът има поне един или повече елементи
  + **contains(елемент)**: връща **true**, ако елементът е наличен в стека и връща **false**, ако елементът го няма в стека



* Примерен код:
  + Създаване на стек:

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

* + Добавяне на данни в стек от конзолата:

**Arrays.stream(scanner.** **nextLine().split("\\s+"))**

**.map(Integer::parseInt).forEach(stack::push);**

* + Отпечатване на данни от стек на конзолата:

**stack.stream().map(String::valueOf)**

**.collect(Collectors.joining(", "));**

* + Допълнителни операции:

**ArrayDeque<Integer> stack = new ArrayDeque<>();  
stack.push(2);  
stack.push(3);  
stack.push(5);  
stack.push(13);  
  
System.*out*.println(stack.peek()); *//13*int lastElement = stack.pop(); *//13*System.*out*.println(lastElement);  
  
System.*out*.println(stack.size()); *//3*System.*out*.println(stack.contains(3)); *//true*System.*out*.println(stack.contains(32)); *//false*System.*out*.println(stack.isEmpty()); *//false*stack.forEach(System.*out*::println);**

* Особености при работа със стек:
  + Винаги проверявайте при pop() дали има елементи в стека
  + Често се използва в случаи, в които трябва да вземем последния въведен елемент

# Опашка (Queue)

* **FIFO (First In, First Out)**
* **Първият добавен елемент излиза първи**
* Визуализация: **под формата на опашка в магазин**
* Основни методи:
  + **add (елемент)**: добавя елемент на края на опашката, хвърля exception при пълна опашка
  + **offer (елемент)**: добавя елемент на края на опашката, връща null при пълна опашка



* + **poll()**: премахва и връща елемента от края на опашката, връща null при празна опашка
  + **remove()**: премахва и връща елемента от края на опашката, хвърля exception при празна опашка



* + **peek()**: връща елементът, който е в края на опашката, но не го премахва



* + **size()**: връща цяло число, което показва колко е размерът на опашката (броят на елементите)
  + **isEmpty()**: връща **true**, ако е празна опашката и връща **false**, ако опашката има поне един или повече елементи
  + **contains(елемент)**: връща **true**, ако елементът е наличен в опашката и връща **false**, ако елементът го няма в опашката
* Примерен код:
  + Създаване на опашка:

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

* + Добавяне на данни в стек от конзолата:

**ArrayDeque<Integer> queue = Arrays.stream(scanner.nextLine().split("\\s+"))**

**.map(Integer::parseInt) .collect(Collectors.toCollection(ArrayDeque::new));**

* + Отпечатване на данни от опашка на конзолата:

**queue.stream().map(String::valueOf)**

**.collect(Collectors.joining(", "));**

* + Допълнителни операции:

**ArrayDeque<Integer> queue = new ArrayDeque<>();  
queue.add(2);  
queue.offer(3);  
queue.add(5);  
queue.offer(13);  
  
System.*out*.println(queue.peek()); *//13*int lastElement = queue.remove(); *//13*System.*out*.println(lastElement);  
  
System.*out*.println(queue.size()); *//2*System.*out*.println(queue.contains(3)); *//true*System.*out*.println(queue.contains(32)); *//false*System.*out*.println(queue.isEmpty()); *//false*queue.forEach(System.*out*::println);**

# Речници (Maps)

* Видове:
  + **HashMap** – не е гарантирана подредбата на записите
  + **LinkedHashMap** – редът на записите се запазва спрямо реда на добавяне
  + **TreeMap** – записите се сортират спрямо стойностите на ключовете в ascending order
* Сортировки:
  + **Sorting map by value (list count)**

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

**map**

**.entrySet()**

**.stream()**

**.sorted((a, b) -> b.getValue().size() - a.getValue().size())**

**.forEach(pair -> {**

**System.out.println(pair.getKey());**

**pair.getValue().forEach(el -> System.out.println(el));**

**});**

* **Sorting map by key ascending**

**map**

**.entrySet()**

**.stream()**

**.sorted(comparingByKey())**

**.collect(**

**toMap(Map.Entry::getKey, Map.Entry::getValue, (e1, e2) -> e2,**

**HashMap::new));**

* **Sorting map by value ascending**

**map**

**.entrySet()**

**.stream()**

**.sorted(comparingByValue())**

**.collect(**

**toMap(Map.Entry::getKey, Map.Entry::getValue, (e1, e2) -> e2,**

**LinkedHashMap::new));**

* **Sorting map by key descending**

**map**

**.entrySet()**

**.stream()**

**.sorted(Collections.reverseOrder(comparingByKey()))**

**.collect(**

**toMap(Map.Entry::getKey, Map.Entry::getValue, (e1, e2) -> e2,**

**HashMap::new));**

* **Sorting map by value descending**

**map**

**.entrySet()**

**.stream()**

**.sorted(Collections.reverseOrder(comparingByValue()))**

**.collect(**

**toMap(Map.Entry::getKey, Map.Entry::getValue, (e1, e2) -> e2,**

**LinkedHashMap::new));**

* Отпечатване:
  + Чрез **stream**

**map.entrySet().stream().forEach(entry -> {**

**System.out.println(entry.getKey()); System.out.println(entry.getValue());**

**});**

* + Чрез **foreach**

**map.forEach((key, value) -> {**

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

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

**});**

**for (Map.Entry<Integer, Integer> entry : map.entrySet()) {**

**System.out.println(entry.getKey());**

**System.out.println(entry.getValue());**

**}**

# Матрици (Multidimensional Arrays)

* Създаване на празна матрица
  + Цели числа: **int[][] matrix = new int[rows][cols];**
  + Дробни числа: **double[][] matrix = new double[rows][cols];**
  + Текстове: **String[][] matrix = new String[rows][cols];**
  + Символи: **char[][] matrix = new char[rows][cols];**
* Движения в матрица
  + Долу: **ред + 1**
  + Горе: **ред – 1**
  + Ляво: **колона – 1**
  + Дясно: **колона + 1**
* Валидиране на позиция в матрица

**ред >= 0 && ред < брой редове && колона >= 0 && колона < брой колони**

* Прочитане на матрица от конзолата:
  + Цели числа

**private static void fillMatrix(int[][] matrix, Scanner scanner) {**

**for (int row = 0; row < matrix.length ; row++) {**

**for (int col = 0; col < matrix.length; col++) {**

**matrix[row][col] = scanner.nextInt();**

**}**

**}**

**}**

* + Дробни числа

**private static void fillMatrix(double[][] matrix, Scanner s) {**

**for (int row = 0; row < matrix.length ; row++) {**

**for (int col = 0; col < matrix.length; col++) {**

**matrix[row][col] = s.nextDouble();**

**}**

**}**

**}**

* + Символи

**private static void fillMatrix(char[][] matrix, Scanner s) {**

**for (int row = 0; row < matrix.length ; row++) {**

**char[] rowData = scanner.nextLine().toCharArray();**

**matrix[row] = rowData;**

**}**

**}**

* + Текстове

**private static void fillMatrix(String[][] matrix, Scanner s) {**

**for (int row = 0; row < matrix.length; row++) {**

**matrix[row] = scanner.nextLine().split("");**

**}**

**}**

* Отпечатване на матрица от конзолата:

**private static void printMatrix(String[][] matrix) {**

**for (int row = 0; row < matrix.length; row++) {**

**for (int col = 0; col < matrix.length; col++) {**

**System.out.print(matrix[row][col]);**

**}**

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

**}**

**}**