Java Data Structures] (Extended-CheatSheet)

1. Arrays

- Declare an array: `int[] myArray;`
- Initialize an array: `int[] myArray = {1, 2, 3, 4, 5};`
- Create an array with size: `int[] myArray = new int[5];`
- Access element: `int element = myArray[0];`
- Set element: `myArray[0] = 10;`
- Get array length: `int length = myArray.length;`
- Copy array: `int[] newArray = Arrays.copyOf(myArray, myArray.length);`
- Copy range: `int[] partialArray = Arrays.copyOfRange(myArray, 1, 4);`
- Fill array: `Arrays.fill(myArray, 0);`
- Sort array: `Arrays.sort(myArray);`
- Binary search: `int index = Arrays.binarySearch(myArray, 3);`
- Compare arrays: `boolean isEqual = Arrays.equals(array1, array2);`
- Convert to List: `List<Integer> list = Arrays.asList(myArray);`
- Print array: `System.out.println(Arrays.toString(myArray));`
- Multi-dimensional array: `int[][] matrix = new int[3][3];`

2. ArrayList

- Create an ArrayList: `ArrayList<Integer> list = new ArrayList<>();`
- Create ArrayList with initial capacity: `ArrayList<String> list = new ArrayList<>(10);`
- Create ArrayList from another collection: `ArrayList<String> list = new ArrayList<>(anotherList);`
- Add element: `list.add("element");`
- Add element at index: `list.add(0, "element");`
- Add all elements from another collection: `list.addAll(anotherList);`
- Get element at index: `String element = list.get(0);`
- Set element at index: `list.set(0, "newElement");`
- Remove element: `list.remove("element");`
- Remove element at index: `list.remove(0);`
- Remove all elements from another collection: `list.removeAll(anotherList);`
- Retain all elements from another collection: `list.retainAll(anotherList);`
- Clear all elements: `list.clear();`

- Check if list contains element: `boolean contains = list.contains("element");`
- Get index of element: `int index = list.indexOf("element");`
- Get last index of element: `int lastIndex = list.lastIndexOf("element");`
- Check if list is empty: `boolean isEmpty = list.isEmpty();`
- Get size of list: `int size = list.size();`
- Convert list to array: `Object[] array = list.toArray();`
- Convert list to typed array: `String[] array = list.toArray(new String[0]);`
- Get sublist: `List<String> subList = list.subList(1, 4);`
- Sort list: `Collections.sort(list);`
- Reverse list: `Collections.reverse(list);`
- Shuffle list: `Collections.shuffle(list);`
- Find min element: `String min = Collections.min(list);`
- Find max element: `String max = Collections.max(list);`
- Fill list with element: `Collections.fill(list, "element");`
- Copy list: `ArrayList<String> copy = new ArrayList<>(list);`
- Convert to synchronized list: `List<String> syncList = Collections.synchronizedList(list);`
- Create unmodifiable view of list: `List<String> unmodifiableList = Collections.unmodifiableList(list);`
- Iterate over list: `for (String element : list) { }`
- Iterate with index: `for (int i = 0; i < list.size(); i++) { }`
- Iterate using iterator: `Iterator<String> iter = list.iterator(); while (iter.hasNext()) { }`
- Iterate using listIterator: `ListIterator<String> listIter = list.listIterator();`
- Remove if condition is met: `list.removeIf(element -> element.isEmpty());`
- Replace all elements: `list.replaceAll(String::toUpperCase);`
- For each operation: `list.forEach(System.out::println);`
- Convert to stream: `Stream<String> stream = list.stream();`
- Join elements to string: `String joined = String.join(", ", list);`
- Check if any element satisfies condition: `boolean any = list.stream().anyMatch(String::isEmpty);`
- Check if all elements satisfy condition: `boolean all = list.stream().allMatch(s -> s.length() > 2);`
- Find first element satisfying condition: `Optional<String> first = list.stream().filter(s -> s.startsWith("A")).findFirst();`

2. HashMap

- Create HashMap with initial capacity: `HashMap<String, Integer> map = new HashMap<>(16);`
- Create HashMap from another map: `HashMap<String, Integer> map = new HashMap<>(anotherMap);`
- Put key-value pair: `map.put("key", 1);`
- Put if absent: `map.putIfAbsent("key", 1);`
- Get value by key: `Integer value = map.get("key");`
- Get value by key with default: `Integer value = map.getOrDefault("key", 0);`
- Remove key-value pair: `map.remove("key");`
- Remove key-value pair if value matches: `map.remove("key", 1);`
- Clear all entries: `map.clear();`
- Check if key exists: `boolean containsKey = map.containsKey("key");`
- Check if value exists: `boolean containsValue = map.containsValue(1);`
- Get set of keys: `Set<String> keys = map.keySet();`
- Get collection of values: `Collection<Integer> values = map.values();`
- Get set of entries: `Set<Map.Entry<String, Integer>> entries = map.entrySet();`
- Check if map is empty: `boolean isEmpty = map.isEmpty();`
- Get size of map: `int size = map.size();`
- Replace value for key: `map.replace("key", 2);`
- Replace value if old value matches: `map.replace("key", 1, 2);`
- Merge values: `map.merge("key", 1, Integer::sum);`
- Compute value if absent: `map.computeIfAbsent("key", k -> k.length());`
- Compute value if present: `map.computeIfPresent("key", (k, v) -> v + 1);`
- Compute value: `map.compute("key", (k, v) -> (v == null) ? 1 : v + 1);`
- For each operation: `map.forEach((k, v) -> System.out.println(k + ": " + v));`
- Convert to synchronized map: `Map<String, Integer> syncMap = Collections.synchronizedMap(map);`
- Create unmodifiable view of map: `Map<String, Integer> unmodifiableMap = Collections.unmodifiableMap(map);`
- Iterate over entries: `for (Map.Entry<String, Integer> entry : map.entrySet()) { }`
- Iterate over keys: `for (String key : map.keySet()) { }`
- Iterate over values: `for (Integer value : map.values()) { }`
- Convert to stream: `Stream<Map.Entry<String, Integer>> stream = map.entrySet().stream();`

3. HashSet

- Create a HashSet: `HashSet<String> set = new HashSet<>();`
- Create HashSet with initial capacity: `HashSet<String> set = new HashSet<>(16);`
- Create HashSet from another collection: `HashSet<String> set = new HashSet<>(anotherCollection);`
- Add element: `set.add("element");`
- Remove element: `set.remove("element");`
- Clear all elements: `set.clear();`
- Check if element exists: `boolean contains = set.contains("element");`
- Check if set is empty: `boolean isEmpty = set.isEmpty();`
- Get size of set: `int size = set.size();`
- Add all elements from another collection: `set.addAll(anotherCollection);`
- Remove all elements from another collection: `set.removeAll(anotherCollection);`
- Retain all elements from another collection: `set.retainAll(anotherCollection);`
- Convert set to array: `Object[] array = set.toArray();`
- Convert set to typed array: `String[] array = set.toArray(new String[0]);`
- Iterate over set: `for (String element : set) { }`
- Iterate using iterator: `Iterator<String> iter = set.iterator(); while (iter.hasNext()) { }`
- Remove if condition is met: `set.removeIf(element -> element.isEmpty());`
- For each operation: `set.forEach(System.out::println);`
- Convert to stream: `Stream<String> stream = set.stream();`
- Check if any element satisfies condition: `boolean any = set.stream().anyMatch(String::isEmpty);`
- Check if all elements satisfy condition: `boolean all = set.stream().allMatch(s -> s.length() > 2);`
- Find first element satisfying condition: `Optional<String> first = set.stream().filter(s -> s.startsWith("A")).findFirst();`
- Convert to synchronized set: `Set<String> syncSet = Collections.synchronizedSet(set);`
- Create unmodifiable view of set: `Set<String> unmodifiableSet = Collections.unmodifiableSet(set);`
- Convert to TreeSet (sorted): `TreeSet<String> treeSet = new TreeSet<>(set);`
- Check if set is subset of another set: `boolean isSubset = set.containsAll(anotherSet);`

- Perform union of two sets: `set.addAll(anotherSet);`
- Perform intersection of two sets: `set.retainAll(anotherSet);`
- Perform difference of two sets: `set.removeAll(anotherSet);`

4. LinkedList

- Create a LinkedList: `LinkedList<String> list = new LinkedList<>();`
- Create LinkedList from another collection: `LinkedList<String> list = new LinkedList<>(anotherCollection);`
- Add element: `list.add("element");`
- Add element at index: `list.add(0, "element");`
- Add element at the beginning: `list.addFirst("element");`
- Add element at the end: `list.addLast("element");`
- Remove first occurrence of element: `list.remove("element");`
- Remove element at index: `list.remove(0);`
- Remove first element: `list.removeFirst();`
- Remove last element: `list.removeLast();`
- Get first element: `String first = list.getFirst();`
- Get last element: `String last = list.getLast();` Set element at index:
- `list.set(0, "newElement");`
- Check if list contains element: `boolean contains = list.contains("element");`
- Get index of first occurrence: `int index = list.indexOf("element");`
- Get index of last occurrence: `int lastIndex = list.lastIndexOf("element");`
- Get element at index: `String element = list.get(0);`
- Clear all elements: `list.clear();`
- Check if list is empty: `boolean isEmpty = list.isEmpty();`
- Get size of list: `int size = list.size();`
- Convert list to array: `Object[] array = list.toArray();`
- Convert list to typed array: `String[] array = list.toArray(new String[0]);`
- Get sublist: `List<String> subList = list.subList(1, 4);`
- Add all elements from another collection: `list.addAll(anotherCollection);`
- Add all elements from another collection at index: `list.addAll(1, anotherCollection);`
- Remove all elements from another collection: `list.removeAll(anotherCollection);`
- Retain all elements from another collection: `list.retainAll(anotherCollection);`

- Iterate over list: `for (String element : list) { }`
- Iterate using iterator: `Iterator<String> iter = list.iterator(); while (iter.hasNext()) { }`
- Iterate using list iterator: `ListIterator<String> listIter = list.listIterator();`
- \Iterate in reverse: `Iterator<String> descendingIter = list.descendingIterator();`
- Remove if condition is met: `list.removeIf(element -> element.isEmpty());`
- Replace all elements: `list.replaceAll(String::toUpperCase);`
- For each operation: `list.forEach(System.out::println);`
- Convert to stream: `Stream<String> stream = list.stream();`
- Peek at first element: `String first = list.peek();`
- Peek at last element: `String last = list.peekLast();`
- Poll first element: `String polled = list.poll();`
- Poll last element: `String polled = list.pollLast();`
- Push element onto stack: `list.push("element");`
- Pop element from stack: `String popped = list.pop();`
- Sort list: `Collections.sort(list);`
- Reverse list: `Collections.reverse(list);`
- Shuffle list: `Collections.shuffle(list);`
- Find min element: `String min = Collections.min(list);`
- Find max element: `String max = Collections.max(list);`

5. Stack

- Create Stack: `Stack<String> stack = new Stack<>();`
- Push element: `stack.push("Hello");`
- Pop element: `String popped = stack.pop();`
- Peek top element: `String top = stack.peek();`
- Check if empty: `boolean isEmpty = stack.isEmpty();`
- Get size: `int size = stack.size();`
- Search element: `int position = stack.search("Hello");`
- Clear stack: `stack.clear();`

6. Queue (using LinkedList)

- Create Queue: `Queue<String> queue = new LinkedList<>();`
- Add element: `queue.add("Hello");`
- Offer element: `queue.offer("World");`
- Remove element: `String removed = queue.remove();`

- Poll element: `String polled = queue.poll();`
- Peek front element: `String front = queue.peek();`
- Check if empty: `boolean isEmpty = queue.isEmpty();`
- Get size: `int size = queue.size();`
- Clear queue: `queue.clear();`
- Contains element: `boolean contains = queue.contains("Hello");`

7. PriorityQueue

- Create PriorityQueue: `PriorityQueue<Integer> pq = new PriorityQueue<>();`
- Create with comparator: `PriorityQueue<String> pq = new PriorityQueue<>(Comparator.reverseOrder());`
- Add element: `pq.add(5);`
- Offer element: `pq.offer(3);`
- Remove element: `Integer removed = pq.remove();`
- Poll element: `Integer polled = pq.poll();`
- Peek top element: `Integer top = pq.peek();`
- Check if empty: `boolean isEmpty = pq.isEmpty();`
- Get size: `int size = pq.size();`
- Clear queue: `pq.clear();`
- Contains element: `boolean contains = pq.contains(5);`
- Convert to array: `Object[] array = pq.toArray();`
- Iterator: `Iterator<Integer> it = pq.iterator();`

8. TreeMap

- Create TreeMap: `TreeMap<String, Integer> map = new TreeMap<>();`
- Put key-value pair: `map.put("One", 1);`
- Get value: `Integer value = map.get("One");`
- Remove key-value pair: `map.remove("One");`
- First key: `String firstKey = map.firstKey();`
- Last key: `String lastKey = map.lastKey();`
- Lower key: `String lowerKey = map.lowerKey("One");`
- Higher key: `String higherKey = map.higherKey("One");`
- Floor key: `String floorKey = map.floorKey("One");`
- Ceiling key: `String ceilingKey = map.ceilingKey("One");`
- First entry: `Map.Entry<String, Integer> firstEntry = map.firstEntry();`
- Last entry: `Map.Entry<String, Integer> lastEntry = map.lastEntry();`
- Lower entry: `Map.Entry<String, Integer> lowerEntry = map.lowerEntry("One");`

- Higher entry: `Map.Entry<String, Integer> higherEntry = map.higherEntry("One");`
- Floor entry: `Map.Entry<String, Integer> floorEntry = map.floorEntry("One");`
- Ceiling entry: `Map.Entry<String, Integer> ceilingEntry = map.ceilingEntry("One");`
- Poll first entry: `Map.Entry<String, Integer> firstEntry = map.pollFirstEntry();`
- Poll last entry: `Map.Entry<String, Integer> lastEntry = map.pollLastEntry();`
- Submap: `SortedMap<String, Integer> subMap = map.subMap("A", "D");`
- Headmap: `SortedMap<String, Integer> headMap = map.headMap("D");`
- Tailmap: `SortedMap<String, Integer> tailMap = map.tailMap("D");`
- Descending key set: `NavigableSet<String> descKeys = map.descendingKeySet();`
- Descending map: `NavigableMap<String, Integer> descMap = map.descendingMap();`

9. TreeSet

- Create TreeSet: `TreeSet<String> set = new TreeSet<>();`
- Add element: `set.add("Hello");`
- Remove element: `set.remove("Hello");`
- First element: `String first = set.first();`
- Last element: `String last = set.last();`
- Lower element: `String lower = set.lower("Hello");`
- Higher element: `String higher = set.higher("Hello");`
- Floor element: `String floor = set.floor("Hello");`
- Ceiling element: `String ceiling = set.ceiling("Hello");`
- Poll first: `String first = set.pollFirst();`
- Poll last: `String last = set.pollLast();`
- Subset: `SortedSet<String> subSet = set.subSet("A", "D");`
- Headset: `SortedSet<String> headSet = set.headSet("D");`
- Tailset: `SortedSet<String> tailSet = set.tailSet("D");`
- Descending set: `NavigableSet<String> descSet = set.descendingSet();`
- Iterator: `Iterator<String> it = set.iterator();`
- Descending iterator: `Iterator<String> descIt = set.descendingIterator();`