#_ [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 a HashMap: `HashMap<String, Integer> map = new 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();`

10. String

- Create a String: `String str = "Hello, World!";`
- Create String from char array: `String str = new String(new char[]{'H', 'e', 'l', 'l', 'o'});`
- Create String from byte array: `String str = new String(new byte[]{72, 101, 108, 108, 111}, StandardCharsets.UTF_8);`
- Get length of string: `int length = str.length();`
- Get character at index: `char ch = str.charAt(0);`
- Get substring: `String sub = str.substring(0, 5);`
- Concatenate strings: `String concat = str.concat(" How are you?");`
- Check if string contains substring: `boolean contains = str.contains("World");`
- Check if string starts with prefix: `boolean startsWith = str.startsWith("Hello");`
- Check if string ends with suffix: `boolean endsWith = str.endsWith("!");`
- Compare strings: `int result = str.compareTo("Hello");`
- Compare strings ignoring case: `int result = str.compareToIgnoreCase("HELLO");`
- Convert to lowercase: `String lower = str.toLowerCase();`
- Convert to uppercase: `String upper = str.toUpperCase();`
- Trim whitespace: `String trimmed = str.trim();`
- Replace character: `String replaced = str.replace('o', '0');`
- Replace sequence: `String replaced = str.replace("World", "Java");`
- Replace first occurrence: `String replaced = str.replaceFirst("l", "L");`
- Replace all occurrences: `String replaced = str.replaceAll("l", "L");`
- Split string: `String[] parts = str.split(", ");`
- Join strings: `String joined = String.join(", ", "Hello", "World");`
- Check if string is empty: `boolean isEmpty = str.isEmpty();`
- Check if string is blank: `boolean isBlank = str.isBlank();`
- Get index of character: `int index = str.indexOf('o');`
- Get last index of character: `int lastIndex = str.lastIndexOf('o');`
- Get index of substring: `int index = str.indexOf("World");`
- Get last index of substring: `int lastIndex = str.lastIndexOf("o");`
- Convert to char array: `char[] chars = str.toCharArray();`
- Get bytes: `byte[] bytes = str.getBytes();`
- Get bytes with charset: `byte[] bytes = str.getBytes(StandardCharsets.UTF_8);`
- Matches regex: `boolean matches = str.matches("Hello.*");`

- Format string: `String formatted = String.format("Hello, %s!", "World");`
- Repeat string: `String repeated = "Hello".repeat(3);`
- Strip leading spaces: `String stripped = str.stripLeading();`
- Strip trailing spaces: `String stripped = str.stripTrailing();`
- Strip all spaces: `String stripped = str.strip();`
- Convert to int: `int num = Integer.parseInt("123");`
- Convert to long: `long num = Long.parseLong("123");`
- Convert to double: `double num = Double.parseDouble("123.45");`
- Convert to float: `float num = Float.parseFloat("123.45");`
- Convert to boolean: `boolean bool = Boolean.parseBoolean("true");`
- Convert int to String: `String str = String.valueOf(123);`
- Convert long to String: `String str = String.valueOf(123L);`
- Convert double to String: `String str = String.valueOf(123.45);`
- Convert float to String: `String str = String.valueOf(123.45f);`
- Convert boolean to String: `String str = String.valueOf(true);`
- Convert to StringBuilder: `StringBuilder sb = new StringBuilder(str);`
- Convert to StringBuffer: `StringBuffer sb = new StringBuffer(str);`
- Intern string: `String internedStr = str.intern();`
- Compare string references: `boolean isEqual = str1 == str2;`
- Compare string content: `boolean isEqual = str1.equals(str2);`
- Compare string content ignoring case: `boolean isEqual = str1.equalsIgnoreCase(str2);`
- Check if string is palindrome: `boolean isPalindrome = str.equals(new StringBuilder(str).reverse().toString());`
- Get Unicode code point at index: `int codePoint = str.codePointAt(0);`
- Get Unicode code point before index: `int codePoint = str.codePointBefore(1);`
- Count Unicode code points: `int count = str.codePointCount(0, str.length());`
- Get index by Unicode code point: `int index = str.offsetByCodePoints(0, 1);`
- Convert to character stream: `IntStream charStream = str.chars();`
- Convert to code point stream: `IntStream codePointStream = str.codePoints();`
- Check if string contains only digits: `boolean isDigits = str.matches("\\d+");`
- Check if string contains only letters: `boolean isLetters = str.matches("[a-zA-Z]+");`
- Check if string contains only letters and digits: `boolean isAlphanumeric = str.matches("[a-zA-Z0-9]+");`
- Remove leading zeros: `String noLeadingZeros = str.replaceFirst("^0+(?!$)", "");`

- Pad left with zeros: `String padded = String.format("%5s", str).replace(' ', '0');`
- Pad right with spaces: `String padded = String.format("%-5s", str);`
- Convert first character to uppercase: `String capitalized = str.substring(0, 1).toUpperCase() + str.substring(1);`
- Reverse string: `String reversed = new StringBuilder(str).reverse().toString();`
- Check if string is a valid number: `boolean isNumber = str.matches("-?\\d+(\\.\\d+)?");`
- Extract numbers from string: `String numbers = str.replaceAll("[^0-9]", "");`
- Extract letters from string: `String letters = str.replaceAll("[^a-zA-Z]", "");`
- Count occurrences of substring: `int count = (str.length() - str.replace("substring", "").length()) / "substring".length();`
- Truncate string: `String truncated = str.substring(0, Math.min(str.length(), 10));`
- Center string: `String centered = String.format("%" + (padLength + str.length()) / 2 + "s", str);`

Additional Common Operations

- Sort ArrayList: `Collections.sort(arrayList);`
- Binary search ArrayList: `int index = Collections.binarySearch(arrayList, element);`
- Reverse ArrayList: `Collections.reverse(arrayList);`
- Shuffle ArrayList: `Collections.shuffle(arrayList);`
- Find max in Collection: `T max = Collections.max(collection);`
- Find min in Collection: `T min = Collections.min(collection);`
- Fill List: `Collections.fill(list, element);`
- Copy List: `Collections.copy(dest, src);`
- Disjoint Collections: `boolean disjoint = Collections.disjoint(c1, c2);`
- Frequency in Collection: `int freq = Collections.frequency(collection, element);`
- Check if Collection empty: `boolean isEmpty = collection.isEmpty();`
- Convert Collection to Array: `Object[] array = collection.toArray();`
- Add all from Collection: `collection.addAll(otherCollection);`
- Remove all from Collection: `collection.removeAll(otherCollection);`
- Retain all in Collection: `collection.retainAll(otherCollection);`
- Clear Collection: `collection.clear();`

- Get synchronized List: `List<T> syncList = Collections.synchronizedList(list);`
- Get synchronized Set: `Set<T> syncSet = Collections.synchronizedSet(set);`
- Get synchronized Map: `Map<K,V> syncMap = Collections.synchronizedMap(map);`
- Unmodifiable List: `List<T> unmodList = Collections.unmodifiableList(list);`
- Unmodifiable Set: `Set<T> unmodSet = Collections.unmodifiableSet(set);`
- Unmodifiable Map: `Map<K,V> unmodMap = Collections.unmodifiableMap(map);`
- List Iterator: `ListIterator<T> listIt = list.listIterator();`
- Array as List: `List<T> list = Arrays.asList(array);`
- Join ArrayList elements: `String joined = String.join(", ", arrayList);`
- Split String to List: `List<String> list = Arrays.asList(string.split(","));`
- Convert Set to List: `List<T> list = new ArrayList<>(set);`
- Convert List to Set: `Set<T> set = new HashSet<>(list);`
- Map key set to List: `List<K> keyList = new ArrayList<>(map.keySet());`
- Map values to List: `List<V> valueList = new ArrayList<>(map.values());`
- Check if List contains all: `boolean containsAll = list1.containsAll(list2);`
- Get List capacity (ArrayList): `int capacity = ((ArrayList<T>)list).ensureCapacity(minCapacity);`
- Trim ArrayList capacity: `((ArrayList<T>)list).trimToSize();`
- Create immutable List: `List<T> immutableList = List.of(element1, element2, element3);`
- Create immutable Set: `Set<T> immutableSet = Set.of(element1, element2, element3);`
- Create immutable Map: `Map<K,V> immutableMap = Map.of(key1, value1, key2, value2);`
- Stream from Collection: `Stream<T> stream = collection.stream();`
- Parallel stream from Collection: `Stream<T> parallelStream = collection.parallelStream();`
- Filter Collection: `List<T> filtered = list.stream().filter(predicate).collect(Collectors.toList());`
- Map Collection: `List<R> mapped = list.stream().map(function).collect(Collectors.toList());`
- Reduce Collection: `T result = list.stream().reduce(identity, accumulator);`
- Find any match: `Optional<T> any = list.stream().findAny();`
- Find first match: `Optional<T> first = list.stream().findFirst();`
- Check if any match: `boolean anyMatch = list.stream().anyMatch(predicate);`
- Check if all match: `boolean allMatch = list.stream().allMatch(predicate);`

- Check if none match: `boolean noneMatch = list.stream().noneMatch(predicate);`
- Get distinct elements: `List<T> distinct = list.stream().distinct().collect(Collectors.toList());`
- Sort stream: `List<T> sorted = list.stream().sorted().collect(Collectors.toList());`
- Sort stream with comparator: `List<T> sorted = list.stream().sorted(comparator).collect(Collectors.toList());`
- Limit stream: `List<T> limited = list.stream().limit(n).collect(Collectors.toList());`
- Skip elements: `List<T> skipped = list.stream().skip(n).collect(Collectors.toList());`
- Concatenate streams: `Stream<T> concat = Stream.concat(stream1, stream2);`
- Group by: `Map<K,List<T>> grouped = list.stream().collect(Collectors.groupingBy(classifier));`
- Partition by: `Map<Boolean,List<T>> partitioned = list.stream().collect(Collectors.partitioningBy(predicate));`
- Join stream elements: `String joined = list.stream().map(Object::toString).collect(Collectors.joining(", "));`
- Count stream elements: `long count = list.stream().count();`
- Get stream statistics: `IntSummaryStatistics stats = list.stream().mapToInt(mapper).summaryStatistics();`
- Convert to primitive stream: `IntStream intStream = list.stream().mapToInt(mapper);`
- Generate infinite stream: `Stream<T> infinite = Stream.generate(supplier);`
- Create stream of iterates: `Stream<T> iterates = Stream.iterate(seed, operator);`
- Zip streams: `Stream<Pair<T,U>> zipped = StreamZip.zip(stream1, stream2, (t, u) -> new Pair<>(t, u));`
- Flatten nested collections: `List<T> flattened = list.stream().flatMap(Collection::stream).collect(Collectors.toList());`
- Collect to unmodifiable List: `List<T> unmodifiableList = list.stream().collect(Collectors.toUnmodifiableList());`
- Collect to unmodifiable Set: `Set<T> unmodifiableSet = list.stream().collect(Collectors.toUnmodifiableSet());`
- Collect to unmodifiable Map: `Map<K,V> unmodifiableMap = list.stream().collect(Collectors.toUnmodifiableMap(keyMapper, valueMapper));`