# Java Cheat Sheet for Amazon OA

## Basic Data Structures

HashMap:

```
Map<Integer, String> map = new HashMap<>();
map.put(1, "one");
String val = map.get(1);
boolean exists = map.containsKey(1);
map.remove(1);
```

HashSet:

```
Set<String> set = new HashSet<>();
set.add("hi");
boolean exists = set.contains("hi");
set.remove("hi");
```

ArrayList:

```
List<Integer> list = new ArrayList<>();
list.add(5);
int num = list.get(0);
list.set(0, 10);
list.remove(0);
int size = list.size();
```

## Strings

```
String s = "hello";
int len = s.length();
char c = s.charAt(0);
String sub = s.substring(1, 3);
boolean eq = s.equals("hello");
boolean contains = s.contains("he");
String[] parts = s.split(",");
```

## Arrays

```
int[] arr = {5, 2, 8};
Arrays.sort(arr);
int n = arr.length;
System.out.println(Arrays.toString(arr));
```

## Loops

```
for (int i = 0; i < arr.length; i++) {
System.out.println(arr[i]);
}

for (int num : arr) {
System.out.println(num);
}
```

## Common Patterns

Check if element exists in map:

```
if (map.containsKey(target - num)) {
return new int[] { map.get(target - num), i };
}
```

Sort 2D array by column:

```
Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
```

## Array vs ArrayList

Array:

- Fixed size
- Supports primitives (e.g., int[])
- Fastest access
- Limited built-in methods

ArrayList:

- Dynamic size
- Only supports objects (e.g., Integer)
- Has many methods: add, remove, contains, size, get
- Can convert to array: list.toArray(new Integer[0]);

## ■ Java PriorityQueue (Min Heap) Summary

■ What is PriorityQueue?
- Java's PriorityQueue is a heap-based data structure that allows fast access to the smallest (min-heap) or

■ Creating a Min Heap:

```
PriorityQueue<Map.Entry<Integer, Integer>> minHeap =
    new PriorityQueue<>((a, b) -> a.getValue() - b.getValue());
```

■ Common Methods:
offer(x): adds element x to the heap (same as add)
poll(): removes and returns the smallest element (top of the heap)
peek(): returns the smallest element without removing it
size(): returns the current size of the heap
isEmpty(): returns true if heap is empty

■ Why use Min Heap in 'Top K Frequent Elements'?
- We track the k most frequent elements.
- Heap always removes the least frequent when size exceeds k.

## ■ Problem: Top K Frequent Elements

Plain English:
1. Count frequencies using a HashMap.
2. Use Min Heap to keep top k frequent elements.
3. For each entry in map:
   - If heap.size() < k -> add
   - Else if freq > min -> remove min and add current
4. Extract keys from heap as result.

Java Code:
```
Map<Integer, Integer> freqMap = new HashMap<>();
for (int num : nums) {
    freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);
}

PriorityQueue<Map.Entry<Integer, Integer>> minHeap =
    new PriorityQueue<>((a, b) -> a.getValue() - b.getValue());
for (Map.Entry<Integer, Integer> entry : freqMap.entrySet()) {
    minHeap.offer(entry);
    if (minHeap.size() > k) {
        minHeap.poll();
    }
}

int[] result = new int[k]; int i = 0;
while (!minHeap.isEmpty()) {
    result[i++] = minHeap.poll().getKey();
}
```

Time Complexity: O(n log k)
Space Complexity: O(n)

Additional Java Structures & Utilities for Amazon OA

■ StringBuilder (Efficient String Manipulation)

```
StringBuilder sb = new StringBuilder();
sb.append("hello");
sb.append(" world");
String result = sb.toString();
sb.setCharAt(0, 'H');
sb.insert(5, ",");
sb.deleteCharAt(5);
sb.reverse();
```

Why use it? Much faster than using '+' for many string operations.

■ Stack (LIFO Structure)

```
Stack<Character> stack = new Stack<>();
stack.push('(');
char top = stack.peek();
stack.pop();
boolean empty = stack.isEmpty();
```

Use in: Valid Parentheses, Expression Evaluation

■ Deque as Stack/Queue

```
Deque<Integer> deque = new ArrayDeque<>();
deque.push(1);      // Stack push
deque.pop();        // Stack pop
deque.offer(1);     // Queue add
deque.poll();       // Queue remove
```

■ Useful String Tricks

```
String sorted = new String(charArray); // after Arrays.sort(charArray)
String[] parts = str.split(",");
int val = Integer.parseInt("123");
```

■ Common Patterns

- Sliding Window: use two pointers
- HashMap Counting: map.put(x, map.getOrDefault(x, 0) + 1)
- PriorityQueue for Top K: minHeap of size k

These structures appear frequently in OA and LeetCode-style questions.

# ■ Java Queue – Cheat Sheet Addition

■ Queue (FIFO Structure)

```
Queue<Integer> queue = new LinkedList<>();
queue.offer(5);            // Add to the queue
int first = queue.peek();  // Get front element without removing
int removed = queue.poll(); // Remove and return front element
boolean empty = queue.isEmpty();
int size = queue.size();
```

■ Common Uses:
- BFS (Breadth-First Search) on graphs and grids
- Maintaining order of events or tasks

■ Example:

```
Queue<int[]> q = new LinkedList<>();
q.offer(new int[]{0, 0});

while (!q.isEmpty()) {
    int[] pos = q.poll();
    int i = pos[0], j = pos[1];
    // Process i, j
}
```

■ Time Complexity:
- Each operation (offer, poll, peek) is O(1)