Java Coding Test Cheatsheet

0. Frequently Used Libraries

```
import java.util.*;      // Data structures
import java.io.*;        // Fast I/O
import java.math.*;      // BigInteger, BigDecimal
```

1. Variable & Array Declaration

```
String[] arr1 = new String[5];
int[] arr2 = {1, 2, 3};
int N = 3;
int[] arr3 = new int[N];
```

2. Arrays Utility

```
Arrays.sort(arr);                                // Ascending
Arrays.sort(arr, Collections.reverseOrder());    // Descending
Arrays.sort(arr, 0, 4);                          // Partial sort
Arrays.binarySearch(arr, 2);                     // Binary search
List<String> list = Arrays.asList(arr1);         // Convert to List
int[] tmp = Arrays.copyOfRange(arr, 0, 3);       // Slice
```

3. length / length() / size()

```
arr.length       // Arrays
str.length()     // String
```

```
list.size()       // Collections
```

4. String Handling

```
str.split(" ");
str.substring(0, 5);
str.charAt(i);
str.toUpperCase();
str.toLowerCase();
String[] letters = str.split("");
String newStr = str.substring(0,4) + "X" + str.substring(5);
```

5. HashMap

```
Map<String, Integer> map = new HashMap<>();
map.put("key", 1);
map.get("key");
map.containsKey("key");
map.getOrDefault("key", 0);
for (String k : map.keySet()) map.get(k);
```

6. ArrayList

```
List<String> list = new ArrayList<>();
list.add("a");
list.set(0, "b");
list.remove("b");
list.contains("a");
list.indexOf("a");
```

7. Queue (LinkedList)

```
Queue<Integer> q = new LinkedList<>();
q.offer(1);
q.poll();
q.peek();
q.clear();
q.isEmpty();
```

8. PriorityQueue

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
PriorityQueue<Integer> maxPq = new PriorityQueue<>(Collections.reverseOrder());
```

9. Math

```
Math.max(a, b);
Math.min(a, b);
Math.abs(x);
Math.ceil(x);
Math.floor(x);
Math.round(x);
String.format("%.2f", d);  // Round to 2 decimal
Math.pow(a, b);
Math.sqrt(x);
```

10. HashSet

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

11. Stack

```
Stack<Integer> stack = new Stack<>();
stack.push(1);
stack.pop();
stack.peek();
```

12. Deque (ArrayDeque)

```
Deque<Integer> dq = new ArrayDeque<>();
dq.addFirst(1);
dq.addLast(2);
dq.pollFirst();
dq.pollLast();
```

13. TreeSet

```
TreeSet<Integer> ts = new TreeSet<>();
ts.add(5);
ts.first();
ts.last();
ts.lower(5);
ts.higher(5);
```

14. TreeMap

```
TreeMap<String, Integer> tm = new TreeMap<>();
tm.put("apple", 3);
tm.firstKey();
tm.lastKey();
```

15. Fast I/O

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
StringTokenizer st = new StringTokenizer(br.readLine());
int a = Integer.parseInt(st.nextToken());
String[] parts = br.readLine().split(" ");

BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
bw.write("Hello\n");
bw.flush();
bw.close();
```

---

16. DFS

```
void dfs(int node) {
    visited[node] = true;
    for (int next : graph.get(node)) {
        if (!visited[next]) dfs(next);
    }
}
```

17. BFS

```
Queue<Integer> q = new LinkedList<>();
q.offer(start);
visited[start] = true;
while (!q.isEmpty()) {
    int cur = q.poll();
    for (int next : graph.get(cur)) {
        if (!visited[next]) {
            q.offer(next);
            visited[next] = true;
        }
    }
}
```

18. Sorting

```
list.sort(Comparator.naturalOrder());
list.sort(Comparator.reverseOrder());
people.sort(Comparator.comparingInt(p -> p.age));
```

19. DP (Fibonacci)

```
int[] dp = new int[N+1];
dp[0] = 0; dp[1] = 1;
for (int i = 2; i <= N; i++) dp[i] = dp[i-1] + dp[i-2];
```

20. Union-Find

```
int find(int x) {
    if (x != parent[x]) parent[x] = find(parent[x]);
    return parent[x];
}
void union(int a, int b) {
    a = find(a); b = find(b);
    if (a != b) parent[b] = a;
}
```

21. Bitmask

```
int bit = 0;
bit |= (1 << 3);
bit &= ~(1 << 3);
boolean on = (bit & (1 << 3)) != 0;
Integer.bitCount(bit);
```

22. Greedy

```
int[] coins = {500, 100, 50, 10};
int count = 0;
for (int coin : coins) {
    count += target / coin;
    target %= coin;
}
```

23. Prefix Sum

```
int[] prefix = new int[N+1];
for (int i = 0; i < N; i++) prefix[i+1] = prefix[i] + arr[i];
```

24. Sliding Window

```
int sum = 0;
for (int i = 0; i < k; i++) sum += arr[i];
int max = sum;
for (int i = k; i < N; i++) {
    sum += arr[i] - arr[i-k];
    max = Math.max(max, sum);
}
```

25. Sieve of Eratosthenes

```
boolean[] isPrime = new boolean[N+1];
Arrays.fill(isPrime, true);
for (int i = 2; i*i <= N; i++) {
    if (isPrime[i]) {
        for (int j = i*i; j <= N; j += i) isPrime[j] = false;
    }
}
```

26. Dijkstra

```
PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a -> a[1]));
int[] dist = new int[N+1];
Arrays.fill(dist, INF);
dist[start] = 0;
pq.offer(new int[]{start, 0});
```

27. Tree / LCA

```
void dfs(int node, int par, int d) {
    parent[node] = par;
    depth[node] = d;
    for (int next : tree.get(node)) {
        if (next != par) dfs(next, node, d+1);
    }
}
```

28. Coding Test Patterns

- Use BufferedReader & BufferedWriter
- HashMap, HashSet, TreeMap, PriorityQueue
- Always validate inputs
- Use boolean[], Set, or visited[][] for visited states
- Practice common patterns (DFS/BFS, DP, Greedy, Two Pointers)

29. Backtracking (Permutation, Combination)

```
void permute(List<Integer> list, boolean[] used) {
    if (list.size() == N) {
        System.out.println(list);
        return;
    }
```

```
    for (int i = 0; i < N; i++) {
        if (!used[i]) {
            used[i] = true;
            list.add(arr[i]);
            permute(list, used);
            list.remove(list.size() - 1);
            used[i] = false;
        }
    }
}
```

30. Binary Search / Parametric Search

```
int left = 1, right = maxVal;
while (left <= right) {
    int mid = (left + right) / 2;
    if (condition(mid)) {
        answer = mid;
        right = mid - 1;
    } else {
        left = mid + 1;
    }
}
```

31. Recursion Optimization

- Tail recursion is not optimized in Java.
- Prefer loops for heavy stack recursion.
- Consider memoization or iterative conversion.

32. String Algorithms (KMP)

```
int[] makeTable(String pattern) {
    int[] table = new int[pattern.length()];
    int j = 0;
    for (int i = 1; i < pattern.length(); i++) {
```

```
        while (j > 0 && pattern.charAt(i) != pattern.charAt(j)) j = table[j-1];
        if (pattern.charAt(i) == pattern.charAt(j)) table[i] = ++j;
    }
    return table;
}
```

33. Permutations / Combinations / Bitmask

```
// Bitmask Combination
for (int mask = 0; mask < (1 << N); mask++) {
    for (int i = 0; i < N; i++) {
        if ((mask & (1 << i)) != 0) {
            // element i is selected
        }
    }
}
```

34. BigInteger / BigDecimal

```
BigInteger a = new BigInteger("12345678901234567890");
BigInteger b = new BigInteger("98765432109876543210");
BigInteger sum = a.add(b);
BigDecimal dec = new BigDecimal("1234.5678");
BigDecimal result = dec.setScale(2, RoundingMode.HALF_UP);
```

□□ : chatGPT

Revision #3
Created 22 May 2025 02:36:39 by Dain
Updated 22 May 2025 04:33:33 by Dain