ChatGPT

Top 30 Java Coding Problems for SDET Interviews

1. [ ] **Two Sum (Array, HashMap):** Given an array of integers and a target sum, find two elements whose values add up to the target. (Concepts: array traversal, HashMap lookup; Difficulty: Easy) *Why important:* This classic "two-sum" problem tests HashMap usage for constant-time lookups and is frequently asked to gauge problem-solving speed [1].

2. [ ] **Longest Substring Without Repeating Characters:** Given a string, find the length (or content) of the longest substring that contains no repeated characters. (Concepts: sliding-window, HashSet/HashMap; Difficulty: Medium) *Why important:* This is a standard sliding-window problem to test your ability to maintain and slide a window over a string in linear time. It's often cited as a key interview question on using two pointers with HashMaps [2].

3. [ ] **Reverse a String:** Write a method to reverse a given string (without using built-in reverse functions). (Concepts: string manipulation, loops; Difficulty: Easy) *Why important:* String reversal is a common sanity check of basic coding skills. Interviewers use it to verify understanding of character arrays or `StringBuilder` reversal [3].

4. [ ] **Reverse Words in a Sentence:** Given a sentence, reverse the order of the words (e.g. "Hello World" → "World Hello"). (Concepts: string splitting, array/list manipulation; Difficulty: Medium) *Why important:* This tests string parsing and manipulation (often by splitting on spaces). It's akin to problems like swapping adjacent words, a staple of QA coding rounds [4].

5. [ ] **String Palindrome Check:** Determine whether a given string reads the same forwards and backwards. (Concepts: string manipulation, two-pointer; Difficulty: Easy) *Why important:* Palindrome checks validate control-flow logic. It's a very common warm-up question (often done with two-pointer scanning) [5].

6. [ ] **Number Palindrome Check:** Check whether an integer is a palindrome (e.g. 12321 is a palindrome). (Concepts: arithmetic operations, no-string solution; Difficulty: Easy) *Why important:* Similar to string palindrome but tests numeric manipulation without converting to string. It ensures familiarity with arithmetic reversal or digit-extraction loops [5].

7. [ ] **Anagram Check:** Given two strings, determine if one is an anagram of the other. (Concepts: sorting or frequency counting; Difficulty: Easy) *Why important:* This classic problem tests string manipulation and sorting or HashMap usage. Grouping and checking anagrams is a well-known coding challenge [6] [7].

8. [ ] **Count Character Frequencies:** Count the number of occurrences of each character in a string (return as a map/HashMap). (Concepts: HashMap for frequency counting; Difficulty: Medium) *Why important:* Frequency counting is fundamental (e.g. for anagram checks or duplicate detection). Interviewers ask it to test HashMap skills [8].

9. [ ] **First Non-Repeating Character:** Given a string, find the first character that does not repeat. (Concepts: HashMap frequency, string traversal; Difficulty: Medium) *Why important:* This requires combining counting frequencies with maintaining order. It tests understanding of hash-based counting plus ordering.

10. [ ] **Remove Duplicates from Array:** Given an integer array, remove duplicate elements (in place or by returning unique list). (Concepts: arrays, HashSet or sorting; Difficulty: Medium) *Why important:* Removing duplicates is a common array manipulation problem. It checks use of Sets or two-pointer techniques to eliminate duplicates.

11. [ ] **Remove Duplicates from String:** Given a string, return the string with duplicate characters removed (preserving first occurrences). (Concepts: string, Set or index tracking; Difficulty: Medium) *Why important:* Similar to the array version, this tests use of a HashSet or boolean array to track seen characters. It's frequently asked (e.g. "remove duplicates from a string") [9].

12. [ ] **Find Missing Number in Array:** Given an array containing numbers 1..n with one number missing, find the missing number. (Concepts: arithmetic sum or boolean tracking; Difficulty: Medium) *Why important:* This tests array traversal and possibly arithmetic trick (sum formula). Interviewers use it to see if candidates think of O(n) solutions.

13. [ ] **Second Largest Number in Array:** Find the 2nd largest element in an array of integers. (Concepts: array traversal, variable tracking; Difficulty: Medium) *Why important:* Finding the second maximum checks understanding of multiple-pass or two-pointer logic. It's often asked (e.g. "second-highest number in an array" [10]) to test attention to edge cases.

14. [ ] **Maximum and Minimum in Array:** Find the largest and smallest numbers in an unsorted array. (Concepts: linear scan; Difficulty: Easy) *Why important:* Basic but essential. Helps verify simple loop logic. SoftwareTesting guides list this as a top Java interview program (max/min search) [11].

15. [ ] **Intersection of Two Arrays:** Given two arrays, find their intersection (elements common to both). (Concepts: HashSet, HashMap; Difficulty: Medium) *Why important:* This tests use of Set or Map to find shared elements efficiently. It's common in interviews to check set/hash operations and edge case handling.

16. [ ] **Merge Two Sorted Arrays:** Merge two sorted arrays into one sorted array. (Concepts: two-pointer technique; Difficulty: Medium) *Why important:* Merging sorted arrays (like in merge sort) checks the two-pointer approach and careful index handling. It's often used to evaluate understanding of array algorithms.

17. [ ] **Sort an Array (without built-in):** Sort an integer array (e.g. implement selection or bubble sort). (Concepts: sorting algorithms, complexity; Difficulty: Medium) *Why important:* Understanding sorting is fundamental. Interviewers may ask for simple sorts (selection/bubble) or expect use of `Arrays.sort()`. Known questions include writing sort without built-ins [12].

18. [ ] **Binary Search in a Sorted Array:** Given a sorted array and a target, implement binary search to find the target's index. (Concepts: divide-and-conquer, loops; Difficulty: Medium) *Why important:*

Classic algorithmic problem. Verifies understanding of logarithmic search. Interviewers expect efficient search strategies on sorted data.

19. [ ] **Maximum Subarray Sum (Kadane's Algorithm):** Find the contiguous subarray with the largest sum. (Concepts: dynamic programming/sliding window; Difficulty: Medium) *Why important:* A standard array problem (Kadane's algorithm) that tests handling of running totals. It's a common interview problem to test dynamic thinking in O(n) time.

20. [ ] **Minimum Size Subarray Sum (Sliding Window):** Given an array of positive numbers and a target sum, find the minimal length of a contiguous subarray whose sum $\geq$ target. (Concepts: sliding-window, two pointers; Difficulty: Upper-Medium) *Why important:* This variant of sliding window checks adjusting window size dynamically. It's a realistic interview question for subarray problems.

21. [ ] **Balanced Parentheses (Stack):** Check if an expression's parentheses/brackets are balanced (e.g. "[{()}]" is balanced). (Concepts: stack data structure; Difficulty: Medium) *Why important:* This tests the stack data structure and control flow. Verifying balanced brackets is a classic interview question [13].

22. [ ] **Group Anagrams:** Given an array of strings, group the anagrams together (e.g. output lists of anagrams). (Concepts: HashMap of sorted-key, lists; Difficulty: Medium) *Why important:* Grouping anagrams is a known coding challenge that tests HashMap usage with string keys [7]. It ensures you can manipulate strings (sort or count) and organize data by key.

23. [ ] **Generate All Permutations of a String:** Given a string, output all permutations of its characters. (Concepts: recursion, backtracking; Difficulty: Medium) *Why important:* This tests recursion and backtracking. Generating permutations is common in interviews to assess depth-first search logic.

24. [ ] **Generate All Subsets of an Array (Power Set):** Given an array of numbers, generate all possible subsets. (Concepts: recursion, backtracking; Difficulty: Medium) *Why important:* Similar to permutations, generating subsets checks recursion and bitmask or backtracking skills. It's a classic problem (power set) for algorithmic thinking.

25. [ ] **Fibonacci Sequence:** Compute the first N Fibonacci numbers (either iteratively or recursively). (Concepts: loops vs recursion, memoization; Difficulty: Easy) *Why important:* A basic problem often used to discuss recursion, iteration, and performance (e.g. memoization). Interviewers ask it to assess understanding of iterative vs recursive solutions.

26. [ ] **Factorial:** Compute the factorial of a number (iteratively or recursively). (Concepts: recursion, loops; Difficulty: Easy) *Why important:* A trivial check of recursion or loop use. It's a fundamental example problem in many interview preparation lists.

27. [ ] **Implement Queue Using Two Stacks:** Design a queue with enqueue/dequeue operations using two stacks. (Concepts: stack vs queue, data structure design; Difficulty: Medium) *Why important:* This classic design problem tests knowledge of stacks, queues, and amortized analysis. It checks if you can simulate one data structure using another.

28. [ ] **Longest Common Prefix:** Given an array of strings, find the longest common prefix. (Concepts: string comparison, array traversal; Difficulty: Easy) *Why important:* Simple problem that tests string traversal logic. It's a common coding question (e.g. on LeetCode) to practice string and array manipulation.

29. [ ] **Evaluate Reverse Polish Notation:** Evaluate an expression given in Reverse Polish Notation (postfix), e.g. `["2","1","+","3","*"]` = 9. (Concepts: stack, expression parsing; Difficulty: Medium) *Why important:* This tests stack usage and parsing tokens. It's a well-known problem to check understanding of expression evaluation.

30. [ ] **Majority Element:** Given an array where a majority element (> n/2 times) always exists, find that element. (Concepts: HashMap counting or Boyer–Moore; Difficulty: Medium) *Why important:* This problem appears frequently in interviews [14]. It checks ability to count with a map or use the Boyer–Moore voting algorithm for linear-time identification.

Each of these problems covers key Java fundamentals or common data-structure/algorithm patterns (hash maps, sliding windows, recursion, etc.), ensuring broad preparation for SDET/QA coding interviews. Mastery of these will help reason through many unseen variations in interviews.

Sources: Common coding interview problem lists and tutorials (e.g. SDET interview guides [1] [13], Java practice problems [3] [7]). Each problem above is drawn from frequently cited interview questions to maximize coverage and relevance.

---

[1] [6] [8] [9] [11] [12] Top Java Coding Interview Questions for SDET | by Beknazar | Medium
https://beknazarsuranchiyev.medium.com/top-17-java-coding-interview-questions-for-sdet-a978754eb078

[2] Top 10 Most Asked Sliding Window Interview Questions | by Kirti Arora | Medium
https://medium.com/@kirti07arora/top-10-most-asked-sliding-window-interview-questions-cf8dfdb3fe48

[3] [5] [10] Top 20 Java Interview Programs for Programming and Coding Interview
https://www.softwaretestinghelp.com/java-coding-interview-programs/

[4] Top Java Coding Interview Questions for Automation Tester | by Bhavin Thumar | Medium
https://medium.com/@bhavin_thumar/top-java-coding-interview-questions-for-automation-tester-85550ab4d73c

[7] Solving the Group Anagrams Problem in Java and Go | by Mehar Chand | Medium
https://medium.com/@mehar.chand.cloud/solving-the-group-anagrams-problem-in-java-and-go-82e8d39299ee

[13] Valid Parentheses in an Expression - GeeksforGeeks
https://www.geeksforgeeks.org/dsa/check-for-balanced-parentheses-in-an-expression/

[14] Top Interview 5/150 (169. Majority Element) | by Aleksandr Gladkikh | Medium
https://medium.com/@alsgladkikh/top-interview-5-150-169-majority-element-469f9732b16e