Java Codathon — 30 Programs (12 Easy, 12 Medium, 6 Hard)

This PDF contains 30 Java programs chosen to match your Codathon pattern (12 Easy, 12 Medium, 6 Hard). Each question includes: detailed explanation, algorithm, time complexity, edge-cases, sample input/output and a full runnable Java program (single-file style with Scanner input). Input formats are noted in each problem's explanation. Practice by running each program and dry-running on the sample I/O provided. Good luck — revise the easy ones first, then medium, then the hard ones.

Table of Contents

Q1. Prime Number Check

Explanation: Input: single integer n. Algorithm: Check edge cases (n < 2 -> not prime). Check divisibility from 2 to sqrt(n). If any divisor found -> not prime; otherwise prime. Time Complexity: O(√n). Space: O(1). Sample I/O: Input: 17 -> Output: Prime

```
import java.util.*;
class Q1_PrimeCheck {
    public static boolean isPrime(int n) {
        if (n < 2) return false;
        if (n == 2) return true;
        if (n % 2 == 0) return false;
        int r = (int)Math.sqrt(n);
        for (int i = 3; i <= r; i += 2) {
            if (n % i == 0) return false;
        }
        return true;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.println(isPrime(n) ? "Prime" : "Not Prime");
    }
}
```

Q2. Armstrong Number Check

Explanation: Input: single integer n. Algorithm: Count digits (d), then compute sum of each digit^d. Compare with original number. For large counts, use loop with Math.pow (convert to int). Time Complexity: O(digits). Sample I/O: 153 -> Armstrong (1^3+5^3+3^3 = 153). Edge cases: 0 and 1 are Armstrong.

```
import java.util.*;
class Q2_Armstrong {
    public static boolean isArmstrong(int n) {
        int temp = n;
        int digits = String.valueOf(n).length();
        int sum = 0;
        while (temp > 0) {
            int d = temp % 10;
            sum += (int)Math.pow(d, digits);
            temp /= 10;
        }
        return sum == n;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.println(isArmstrong(n) ? "Armstrong" : "Not Armstrong");
    }
}
```

Q3. Factorial of a Number

Explanation: Input: single non-negative integer n. Compute n! iteratively (for large n, it may overflow int — use long if needed or BigInteger for very large n). Algorithm: Multiply 1..n. Time Complexity: O(n). Sample I/O: 5 -> 120. Edge cases: 0! = 1.

```
import java.util.*;
import java.math.BigInteger;
class Q3_Factorial {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        // Using BigInteger to avoid overflow for larger n
        BigInteger fact = BigInteger.ONE;
        for (int i = 1; i <= n; i++) fact = fact.multiply(BigInteger.valueOf(i));
        System.out.println(fact.toString());
    }
}
```

Q4. Fibonacci Series (first n terms)

Explanation: Input: integer n (n >= 1). Output first n Fibonacci numbers (0-based: 0,1,1,2,...). Using iterative approach is efficient (O(n)). Recursive approach is exponential without memoization. Sample I/O: n=6 -> 0 1 1 2 3 5

```
import java.util.*;
class Q4_Fibonacci {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        if (n <= 0) return;
        int a = 0, b = 1;
        if (n >= 1) System.out.print(a);
        if (n >= 2) System.out.print(" " + b);
        for (int i = 3; i <= n; i++) {
            int c = a + b;
            System.out.print(" " + c);
            a = b; b = c;
        }
        System.out.println();
    }
}
```

Q5. Reverse a Number

Explanation: Input: integer n (non-negative). Extract last digit with n%10, build reversed number rev = rev*10 + d, and divide n/=10. Handle trailing zeros: reversing 120 -> 21. Time Complexity: O(digits). Sample I/O: 1234 -> 4321.

```
import java.util.*;
class Q5_ReverseNumber {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int rev = 0;
        while (n > 0) {
            int d = n % 10;
            rev = rev * 10 + d;
            n /= 10;
        }
        System.out.println(rev);
    }
}
```

Q6. Sum of Digits of a Number

Explanation: Input: integer n. Repeatedly extract digits with %10 and add them. Handles negative numbers by taking absolute value. Time Complexity: O(digits). Sample I/O: 1234 -> 10.

```
import java.util.*;
class Q6_SumOfDigits {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = Math.abs(sc.nextInt());
        int sum = 0;
        while (n > 0) {
            sum += n % 10;
            n /= 10;
        }
        System.out.println(sum);
    }
}
```

Q7. Palindrome Number

Explanation: Input: integer n. Reverse the number and compare with original. Works for positive integers; if negative, you can define behavior (here negative numbers are not palindromes unless specified). Sample I/O: 121 -> Palindrome, 123 -> Not Palindrome.

```
import java.util.*;
class Q7_PalindromeNumber {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int original = n, rev = 0;
        while (n > 0) {
            rev = rev * 10 + (n % 10);
            n /= 10;
        }
        System.out.println(original == rev ? "Palindrome" : "Not Palindrome");
    }
}
```

Q8. GCD (Greatest Common Divisor) of two numbers

Explanation: Input: two integers a and b. Use Euclidean algorithm: while (b != 0) { t = b; b = a % b; a = t; } result is a. Time Complexity: O(log(min(a,b))). Sample I/O: 12 18 -> GCD = 6.

```
import java.util.*;
class Q8_GCD {
    public static int gcd(int a, int b) {
        a = Math.abs(a); b = Math.abs(b);
        while (b != 0) {
            int t = b;
            b = a % b;
            a = t;
        }
        return a;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt(), b = sc.nextInt();
        System.out.println(gcd(a,b));
    }
}
```

Q9. LCM (Least Common Multiple) of two numbers

Explanation: Input: two integers a and b. Compute LCM via (a / gcd(a,b)) * b to avoid overflow. If a or b is 0, LCM is 0 by common convention. Sample I/O: 12 18 -> LCM = 36.

```
import java.util.*;
class Q9_LCM {
    public static long gcd(long a, long b) {
        a = Math.abs(a); b = Math.abs(b);
        while (b != 0) {
            long t = b; b = a % b; a = t;
        }
        return a;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        long a = sc.nextLong(), b = sc.nextLong();
        if (a == 0 || b == 0) { System.out.println(0); return; }
        long g = gcd(a,b);
        long l = (a / g) * b;
        System.out.println(l);
    }
}
```

Q10. Strong Number (sum of factorial of digits equals number)

Explanation: Input: integer n. For each digit compute factorial and add. Precompute factorials for 0..9 to be efficient. Compare to original. Sample I/O: 145 -> Strong (1! + 4! + 5! = 145).

```
import java.util.*;
class Q10_StrongNumber {
    public static int fact(int n) {
        int f = 1;
        for (int i = 2; i <= n; i++) f *= i;
        return f;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt(), temp = n, sum = 0;
        int[] facts = new int[10];
        for (int i = 0; i < 10; i++) facts[i] = fact(i);
        while (temp > 0) {
            sum += facts[temp % 10];
            temp /= 10;
        }
        System.out.println(sum == n ? "Strong" : "Not Strong");
    }
}
```

Q11. Neon Number (sum of digits of square equals number)

Explanation: Input: integer n. Compute square = n*n. Sum digits of square; if equal to n -> Neon. Example: 9 -> 9*9 = 81 -> 8+1 = 9 -> Neon. Time complexity: O(digits(square)).

```
import java.util.*;
class Q11_NeonNumber {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int sq = n * n;
        int sum = 0;
        while (sq > 0) {
            sum += sq % 10;
            sq /= 10;
        }
        System.out.println(sum == n ? "Neon" : "Not Neon");
    }
}
```

Q12. Pattern Printing - Right Triangle and Inverted

Explanation: Input: integer n (number of rows). Print a right-angled triangle of '*' of height n and an inverted triangle. Uses nested loops. Sample (n=3): * * * * * * * * * * * *

```
import java.util.*;
class Q12_Patterns {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        // Right triangle (increasing)
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) System.out.print("* ");
            System.out.println();
        }
        System.out.println();
        // Inverted triangle (decreasing)
        for (int i = n; i >= 1; i--) {
            for (int j = 1; j <= i; j++) System.out.print("* ");
            System.out.println();
        }
    }
}
```

Q13. Find Max and Min in Array

Explanation: Input: first integer n (size), followed by n integers (array elements). Algorithm: Initialize max and min to arr[0], traverse array updating them. Time: O(n). Sample I/O: n=5 arr=2 9 1 5 3 -> Max=9 Min=1

```
import java.util.*;
class Q13_MaxMin {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
        int max = arr[0], min = arr[0];
        for (int x : arr) {
            if (x > max) max = x;
            if (x < min) min = x;
        }
        System.out.println("Max=" + max + " Min=" + min);
    }
}
```

Q14. Sum and Average of Array Elements

Explanation: Input: n then n elements. Sum while traversing and compute average as (double)sum/n. Edge: n=0 avoid division-by-zero (here we assume n>=1). Sample: n=3 arr=2 3 4 -> Sum=9 Avg=3.0

```
import java.util.*;
class Q14_SumAverage {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        long sum = 0;
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) { arr[i] = sc.nextInt(); sum += arr[i]; }
        double avg = (double) sum / n;
        System.out.println("Sum=" + sum + " Average=" + avg);
    }
}
```

Q15. Count Even and Odd Numbers in Array

Explanation: Input: n followed by n integers. Maintain counters for even and odd using % 2. Sample: n=5 arr=1 2 3 4 5 -> Even=2 Odd=3

```
import java.util.*;
class Q15_EvenOddCount {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int even = 0, odd = 0;
        for (int i = 0; i < n; i++) {
            int x = sc.nextInt();
            if (x % 2 == 0) even++; else odd++;
        }
        System.out.println("Even=" + even + " Odd=" + odd);
    }
}
```

Q16. Reverse an Array (in-place)

Explanation: Input: n followed by n elements. Swap arr[i] and arr[n-1-i] up to middle. Time: O(n). Sample: [1,2,3,4] -> [4,3,2,1]

```
import java.util.*;
class Q16_ReverseArray {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
        int l = 0, r = n - 1;
```

```
        while (l < r) {
            int t = arr[l]; arr[l] = arr[r]; arr[r] = t;
            l++; r--;
        }
        System.out.println(Arrays.toString(arr));
    }
}
```

Q17. Linear Search in Array

Explanation: Input: n, n elements, and a target value to search. Traverse and print index of first occurrence or -1 if not found. Time: O(n).

```
import java.util.*;
class Q17_LinearSearch {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
        int target = sc.nextInt();
        int idx = -1;
        for (int i = 0; i < n; i++) if (arr[i] == target) { idx = i; break; }
        System.out.println(idx);
    }
}
```

Q18. Binary Search (array may be unsorted — we sort first)

Explanation: Input: n followed by n elements (unsorted allowed) and the target. We sort the array first, then apply binary search. If required to preserve original indexes, alternative approaches are needed. Time: O(n log n) due to sorting, binary step O(log n).

```
import java.util.*;
class Q18_BinarySearch {
    public static int binarySearch(int[] arr, int target) {
        int l=0, r=arr.length-1;
        while (l <= r) {
            int mid = l + (r - l) / 2;
            if (arr[mid] == target) return mid;
            else if (arr[mid] < target) l = mid + 1;
            else r = mid - 1;
        }
        return -1;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
        int target = sc.nextInt();
        Arrays.sort(arr);
        int idx = binarySearch(arr, target);
        System.out.println(idx);
    }
}
```

Q19. Bubble Sort

Explanation: Input: n then n elements. Repeatedly bubble up the largest element to the end by adjacent swaps. Time O(n^2). Sample: [3,1,2] -> [1,2,3]

```
import java.util.*;
class Q19_BubbleSort {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
```

```
        for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
        for (int i = 0; i < n-1; i++) {
            for (int j = 0; j < n-i-1; j++) {
                if (arr[j] > arr[j+1]) {
                    int t = arr[j]; arr[j] = arr[j+1]; arr[j+1] = t;
                }
            }
        }
        System.out.println(Arrays.toString(arr));
    }
}
```

Q20. Selection Sort

Explanation: Input: n then elements. For each position i, find index of minimum in i..n-1 and swap with i. Time O(n^2).

```
import java.util.*;
class Q20_SelectionSort {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
        for (int i = 0; i < n-1; i++) {
            int minIdx = i;
            for (int j = i+1; j < n; j++) if (arr[j] < arr[minIdx]) minIdx = j;
            int t = arr[i]; arr[i] = arr[minIdx]; arr[minIdx] = t;
        }
        System.out.println(Arrays.toString(arr));
    }
}
```

Q21. Insertion Sort

Explanation: Input: n then n elements. Iterate from index 1..n-1, insert current element into correct place among previous sorted portion. Time O(n^2) worst-case.

```
import java.util.*;
class Q21_InsertionSort {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
        for (int i = 1; i < n; i++) {
            int key = arr[i]; int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j+1] = arr[j];
                j--;
            }
            arr[j+1] = key;
        }
        System.out.println(Arrays.toString(arr));
    }
}
```

Q22. Second Largest Element in Array

Explanation: Input: n then n elements. Traverse once maintaining firstMax and secondMax. Handle duplicates correctly (if all equal, secondMax may remain minimal sentinel — handle with check).

```
import java.util.*;
class Q22_SecondLargest {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
```

```
        Integer first = null, second = null;
        for (int x : arr) {
            if (first == null || x > first) {
                second = first; first = x;
            } else if ((second == null || x > second) && x != first) {
                second = x;
            }
        }
        if (second == null) System.out.println("No second largest");
        else System.out.println(second);
    }
}
```

Q23. Frequency of Each Element in Array (using HashMap)

Explanation: Input: n then n elements. Use HashMap to count frequencies. Print pairs element:frequency. Time O(n).

```
import java.util.*;
class Q23_Frequency {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        Map<Integer,Integer> freq = new LinkedHashMap<>();
        for (int i = 0; i < n; i++) {
            int x = sc.nextInt();
            freq.put(x, freq.getOrDefault(x, 0) + 1);
        }
        for (Map.Entry<Integer,Integer> e : freq.entrySet()) {
            System.out.println(e.getKey() + " : " + e.getValue());
        }
    }
}
```

Q24. Count Prime Numbers in an Array

Explanation: Input: n then n integers. For each element check primality (reuse sqrt method) and count how many primes are present. Time: O(n * sqrt(max_element)).

```
import java.util.*;
class Q24_CountPrimesInArray {
    public static boolean isPrime(int n) {
        if (n < 2) return false;
        if (n == 2) return true;
        if (n % 2 == 0) return false;
        int r = (int)Math.sqrt(n);
        for (int i = 3; i <= r; i += 2) if (n % i == 0) return false;
        return true;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int count = 0;
        for (int i = 0; i < n; i++) {
            int x = sc.nextInt();
            if (isPrime(x)) count++;
        }
        System.out.println(count);
    }
}
```

Q25. Left Rotate Array by k Positions (Reversal Method)

Explanation: Input: n followed by n elements and integer k. Algorithm: k = k % n. reverse(arr,0,k-1), reverse(arr,k,n-1), reverse(arr,0,n-1). This performs left rotation in O(n) time and O(1) extra space. Sample: [1,2,3,4,5], k=2 -> [3,4,5,1,2].

```
import java.util.*;
```

```
class Q25_LeftRotate {
    public static void reverse(int[] arr,int l,int r) {
        while (l < r) {
            int t = arr[l]; arr[l] = arr[r]; arr[r] = t;
            l++; r--;
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
        int k = sc.nextInt();
        if (n == 0) { System.out.println("[]"); return; }
        k = k % n;
        if (k < 0) k += n;
        if (k != 0) {
            reverse(arr, 0, k-1);
            reverse(arr, k, n-1);
            reverse(arr, 0, n-1);
        }
        System.out.println(Arrays.toString(arr));
    }
}
```

Q26. Right Rotate Array by k Positions (Reversal Method)

Explanation: Input: n, array, k. Right rotation by k is equivalent to left rotation by (n-k). Algorithm using reversal: reverse whole array, reverse(0,k-1), reverse(k,n-1). Sample: [1,2,3,4,5], k=2 -> [4,5,1,2,3].

```
import java.util.*;
class Q26_RightRotate {
    public static void reverse(int[] arr,int l,int r) {
        while (l < r) { int t = arr[l]; arr[l] = arr[r]; arr[r] = t; l++; r--; }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
        int k = sc.nextInt();
        if (n == 0) { System.out.println("[]"); return; }
        k = k % n;
        if (k < 0) k += n;
        if (k != 0) {
            // reverse whole
            reverse(arr, 0, n-1);
            // reverse first k elements
            reverse(arr, 0, k-1);
            // reverse remaining
            reverse(arr, k, n-1);
        }
        System.out.println(Arrays.toString(arr));
    }
}
```

Q27. Rotate Array by 1 Step k Times (Brute Force)

Explanation: Input: n, array, k. Brute-force method: repeat k times -> save first element, shift left all elements, put saved element at end. Time complexity: O(n*k). Use only if k is small or for conceptual clarity.

```
import java.util.*;
class Q27_RotateByOneKTimes {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
        int k = sc.nextInt();
        if (n == 0) { System.out.println("[]"); return; }
        k = k % n;
```

```
        for (int r = 0; r < k; r++) {
            int first = arr[0];
            for (int i = 0; i < n-1; i++) arr[i] = arr[i+1];
            arr[n-1] = first;
        }
        System.out.println(Arrays.toString(arr));
    }
}
```

Q28. Find All Armstrong Numbers in a Given Range

Explanation: Input: two integers low and high. For each number in range [low, high], check Armstrong property (sum of digits^count == number). Print matches. Time: O((high-low)*digits) and careful with ranges (keep range moderate in Codathon).

```
import java.util.*;
class Q28_ArmstrongInRange {
    public static boolean isArmstrong(int n) {
        int temp = n, sum = 0;
        int digits = String.valueOf(n).length();
        while (temp > 0) {
            int d = temp % 10;
            sum += (int)Math.pow(d, digits);
            temp /= 10;
        }
        return sum == n;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int low = sc.nextInt(), high = sc.nextInt();
        ArrayList<Integer> list = new ArrayList<>();
        for (int i = low; i <= high; i++) if (isArmstrong(i)) list.add(i);
        System.out.println(list.toString());
    }
}
```

Q29. Optimized Prime Check (with small skips)

Explanation: Input: integer n. Optimized variant: handle n<2, check 2 and 3, skip multiples of 2/3 and iterate i from 5 to sqrt(n) with i+=6 checks (i and i+2). This is faster in practice and commonly used in competitive programming.

```
import java.util.*;
class Q29_OptimizedPrime {
    public static boolean isPrime(long n) {
        if (n < 2) return false;
        if (n <= 3) return true;
        if (n % 2 == 0 || n % 3 == 0) return false;
        long r = (long)Math.sqrt(n);
        for (long i = 5; i <= r; i += 6) {
            if (n % i == 0 || n % (i + 2) == 0) return false;
        }
        return true;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        long n = sc.nextLong();
        System.out.println(isPrime(n) ? "Prime" : "Not Prime");
    }
}
```

Q30. Largest Subarray Sum (Kadane's Algorithm)

Explanation: Input: n then n integers (can be negative). Kadane's Algorithm keeps maxEndingHere and maxSoFar. Initialize both with first element and iterate. Time: O(n). Sample: [-2,1,-3,4,-1,2,1,-5,4] -> max sum = 6 (subarray [4,-1,2,1]).

```
import java.util.*;
class Q30_Kadane {
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
        int maxEnding = arr[0], maxSoFar = arr[0];
        for (int i = 1; i < n; i++) {
            maxEnding = Math.max(arr[i], maxEnding + arr[i]);
            maxSoFar = Math.max(maxSoFar, maxEnding);
        }
        System.out.println(maxSoFar);
    }
}
```