Java Interview Handbook — Batch 1

Generated: 2025-09-13 02:24:57Z (UTC)

Java Theory & Cheatsheet

```
JAVA THEORY & CHEATSHEET
------------------------
- Types: int, long, double, boolean, char, String
- Collections: List, Set, Map; ArrayList, LinkedList, HashSet, HashMap; PriorityQueue
- Streams/Lambdas; Comparator; Optional
- Concurrency: Thread, synchronized, ExecutorService
- DSA: arrays, linked lists, stacks/queues, heaps, trees, graphs
- Testing: JUnit 5 @Test, Assertions.*
```

001. ReverseString

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem001ReverseString {
    public static String reverseString(String s) {
        if (s == null) return null;
        return new StringBuilder(s).reverse().toString();
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem001ReverseString;


public class TestProblem001ReverseString {
    @Test void t() {
        assertEquals("cba", Problem001ReverseString.reverseString("abc"));
        assertEquals("", Problem001ReverseString.reverseString(""));
        assertNull(Problem001ReverseString.reverseString(null));
    }
}
```

002. IsPalindrome

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem002IsPalindrome {
    public static boolean isPalindrome(String s) {
        if (s == null) return false;
        int i=0,j=s.length()-1;
        while(i<j){ if(s.charAt(i++)!=s.charAt(j--)) return false; }
        return true;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem002IsPalindrome;


public class TestProblem002IsPalindrome {
    @Test void t() {
        assertTrue(Problem002IsPalindrome.isPalindrome("racecar"));
        assertFalse(Problem002IsPalindrome.isPalindrome("hello"));
    }
}
```

003. CharFrequency

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem003CharFrequency {
    public static Map<Character,Integer> charFrequency(String s) {
        Map<Character,Integer> m=new HashMap<>();
        if(s==null) return m;
        for(char c: s.toCharArray()) m.put(c, m.getOrDefault(c,0)+1);
        return m;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem003CharFrequency;


import java.util.*;
public class TestProblem003CharFrequency {
    @Test void t() {
        Map<Character,Integer> m=Problem003CharFrequency.charFrequency("aab");
        assertEquals(2, (int)m.get('a'));
        assertEquals(1, (int)m.get('b'));
    }
}
```

004. FirstNonRepeatedChar

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem004FirstNonRepeatedChar {
    public static Character firstNonRepeated(String s) {
        if(s==null) return null;
        Map<Character,Integer> c=new LinkedHashMap<>();
        for(char ch: s.toCharArray()) c.put(ch, c.getOrDefault(ch,0)+1);
        for(var e: c.entrySet()) if(e.getValue()==1) return e.getKey();
        return null;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem004FirstNonRepeatedChar;


public class TestProblem004FirstNonRepeatedChar {
    @Test void t() { assertEquals(Character.valueOf('c'), Problem004FirstNonRepeatedChar.firstNonRepeated("aabbc")); }
}
```

005. SecondLargest

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem005SecondLargest {
    public static Integer secondLargest(int[] a) {
        if(a==null||a.length<2) return null;
        Integer first=null, second=null;
        for(int n: a){
            if(first==null||n>first){second=first; first=n;}
            else if(n!=first && (second==null||n>second)) second=n;
        }
        return second;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem005SecondLargest;


public class TestProblem005SecondLargest {
    @Test void t() { assertEquals(2, Problem005SecondLargest.secondLargest(new int[]{1,3,2})); }
}
```

006. RemoveDuplicates

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem006RemoveDuplicates {
    public static int[] dedup(int[] a) {
        if(a==null) return null;
        LinkedHashSet<Integer> s=new LinkedHashSet<>(); for(int x:a) s.add(x);
        return s.stream().mapToInt(Integer::intValue).toArray();
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem006RemoveDuplicates;


public class TestProblem006RemoveDuplicates {
    @Test void t() { assertArrayEquals(new int[]{1,2}, Problem006RemoveDuplicates.dedup(new int[]{1,2,1})); }
}
```

007. RotateListK

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem007RotateListK {
    public static int[] rotateRight(int[] a,int k) {
        if(a==null||a.length==0) return a;
        int n=a.length; k%=n; if(k==0) return a.clone();
        int[] r=new int[n];
        for(int i=0;i<n;i++) r[(i+k)%n]=a[i];
        return r;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem007RotateListK;


public class TestProblem007RotateListK {
    @Test void t() { assertArrayEquals(new int[]{4,1,2,3}, Problem007RotateListK.rotateRight(new int[]{1,2,3,4},1)); }
}
```

008. MergeTwoSortedLists

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem008MergeTwoSortedLists {
    public static class ListNode{int val; ListNode next; ListNode(int v){val=v;}}
    public static ListNode merge(ListNode a,ListNode b){
        ListNode d=new ListNode(0), t=d;
        while(a!=null&&b!=null){
            if(a.val<b.val){t.next=a;a=a.next;} else {t.next=b;b=b.next;}
            t=t.next;
        } t.next=(a!=null?a:b); return d.next;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem008MergeTwoSortedLists;


public class TestProblem008MergeTwoSortedLists {
    @Test void t() {
        var A=new Problem008MergeTwoSortedLists.ListNode(1); A.next=new Problem008MergeTwoSortedLists.ListNode(3);
        var B=new Problem008MergeTwoSortedLists.ListNode(2); B.next=new Problem008MergeTwoSortedLists.ListNode(4);
        var R=Problem008MergeTwoSortedLists.merge(A,B);
        assertEquals(1,R.val); assertEquals(2,R.next.val);
    }
}
```

009. LinkedListCycle

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem009LinkedListCycle {
    static class Node{int v; Node next; Node(int v){this.v=v;}}
    public static boolean hasCycle(Node h){
        Node s=h,f=h; while(f!=null&&f.next!=null){ s=s.next; f=f.next.next; if(s==f) return true; }
        return false;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem009LinkedListCycle;


public class TestProblem009LinkedListCycle {
    @Test void t() {
        var a=new Problem009LinkedListCycle.Node(1); var b=new Problem009LinkedListCycle.Node(2); var c=new Problem009L
        a.next=b; b.next=c; c.next=b; assertTrue(Problem009LinkedListCycle.hasCycle(a));
    }
}
```

010. BinarySearch

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem010BinarySearch {
    public static int search(int[] a,int target){
        int lo=0,hi=a.length-1; while(lo<=hi){int mid=(lo+hi)/2; if(a[mid]==target)return mid; if(a[mid]<target)lo=mid+
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem010BinarySearch;


public class TestProblem010BinarySearch {
    @Test void t() { assertEquals(2, Problem010BinarySearch.search(new int[]{1,2,3,4},3)); }
}
```

Java Interview Handbook — Batch 2

Generated: 2025-09-13 02:24:57Z (UTC)

Java Theory & Cheatsheet

```
JAVA THEORY & CHEATSHEET
------------------------
- Types: int, long, double, boolean, char, String
- Collections: List, Set, Map; ArrayList, LinkedList, HashSet, HashMap; PriorityQueue
- Streams/Lambdas; Comparator; Optional
- Concurrency: Thread, synchronized, ExecutorService
- DSA: arrays, linked lists, stacks/queues, heaps, trees, graphs
- Testing: JUnit 5 @Test, Assertions.*
```

011. MergeSort

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem011MergeSort {
    public static int[] sort(int[] a){
        if(a.length<=1) return a.clone();
        int m=a.length/2; int[] L=new int[m], R=new int[a.length-m];
        System.arraycopy(a,0,L,0,m); System.arraycopy(a,m,R,0,a.length-m);
        L=sort(L); R=sort(R);
        int[] res=new int[a.length]; int i=0,j=0,k=0;
        while(i<L.length&&j<R.length) res[k++]= (L[i]<=R[j]?L[i++]:R[j++]);
        while(i<L.length) res[k++]=L[i++]; while(j<R.length) res[k++]=R[j++];
        return res;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem011MergeSort;


public class TestProblem011MergeSort {
    @Test void t() { assertArrayEquals(new int[]{1,2,3}, Problem011MergeSort.sort(new int[]{3,1,2})); }
}
```

012. QuickSort

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem012QuickSort {
    public static void sort(int[] a){qs(a,0,a.length-1);}
    static void qs(int[] a,int l,int r){ if(l>=r)return; int p=a[(l+r)/2],i=l,j=r; while(i<=j){ while(a[i]<p)i++; while
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem012QuickSort;


public class TestProblem012QuickSort {
    @Test void t() { int[] a={3,1,2}; Problem012QuickSort.sort(a); assertArrayEquals(new int[]{1,2,3}, a); }
}
```

013. KthLargest

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem013KthLargest {
    public static int kthLargest(int[] a,int k) {
        PriorityQueue<Integer> pq=new PriorityQueue<>();
        for(int x:a){ pq.offer(x); if(pq.size()>k) pq.poll(); }
        return pq.peek();
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem013KthLargest;


public class TestProblem013KthLargest {
    @Test void t() { assertEquals(3, Problem013KthLargest.kthLargest(new int[]{3,2,1,5,6,4}, 3)); }
}
```

014. KadaneMaxSubarray

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem014KadaneMaxSubarray {
    public static int maxSubarray(int[] a) {
        int best=a[0], cur=a[0];
        for(int i=1;i<a.length;i++){ cur=Math.max(a[i], cur+a[i]); best=Math.max(best,cur);}
        return best;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem014KadaneMaxSubarray;


public class TestProblem014KadaneMaxSubarray {
    @Test void t() { assertEquals(6, Problem014KadaneMaxSubarray.maxSubarray(new int[]{-2,1,-3,4,-1,2,1,-5,4})); }
}
```

015. Permutations

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem015Permutations {
    public static List<List<Integer>> permute(int[] nums){
        List<List<Integer>> res=new ArrayList<>(); backtrack(nums,new boolean[nums.length],new ArrayList<>(),res); retu
    }
    static void backtrack(int[] n, boolean[] used, List<Integer> cur, List<List<Integer>> res){
        if(cur.size()==n.length){res.add(new ArrayList<>(cur)); return;}
        for(int i=0;i<n.length;i++) if(!used[i]){ used[i]=true; cur.add(n[i]); backtrack(n,used,cur,res); cur.remove(cu
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem015Permutations;


import java.util.*;
public class TestProblem015Permutations {
    @Test void t() { assertEquals(6, Problem015Permutations.permute(new int[]{1,2,3}).size()); }
}
```

016. Combinations

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem016Combinations {
    public static List<List<Integer>> combine(int n,int k){
        List<List<Integer>> res=new ArrayList<>(); backtrack(1,n,k,new ArrayList<>(),res); return res;
    }
    static void backtrack(int start,int n,int k,List<Integer> cur,List<List<Integer>> res){
        if(cur.size()==k){res.add(new ArrayList<>(cur)); return;}
        for(int i=start;i<=n;i++){ cur.add(i); backtrack(i+1,n,k,cur,res); cur.remove(cur.size()-1);}
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem016Combinations;


public class TestProblem016Combinations {
    @Test void t() { assertEquals(6, Problem016Combinations.combine(4,2).size()); }
}
```

017. PowerSet

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem017PowerSet {
    public static List<List<Integer>> powerSet(int[] nums){
        List<List<Integer>> res=new ArrayList<>(); res.add(new ArrayList<>());
        for(int x: nums){
            int sz=res.size();
            for(int i=0;i<sz;i++){ List<Integer> n=new ArrayList<>(res.get(i)); n.add(x); res.add(n);}
        } return res;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem017PowerSet;


public class TestProblem017PowerSet {
    @Test void t() { assertEquals(8, Problem017PowerSet.powerSet(new int[]{1,2,3}).size()); }
}
```

018. FibMemo

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem018FibMemo {
    static Map<Integer,Long> memo=new HashMap<>();
    public static long fib(int n){ if(n<2) return n; if(memo.containsKey(n)) return memo.get(n); long v=fib(n-1)+fib(n-
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem018FibMemo;


public class TestProblem018FibMemo {
    @Test void t() { assertEquals(55L, Problem018FibMemo.fib(10)); }
}
```

019. NthFibonacciIterative

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem019NthFibonacciIterative {
    public static long fibN(int n){ if(n<2) return n; long a=0,b=1; for(int i=2;i<=n;i++){ long c=a+b; a=b; b=c; } retu
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem019NthFibonacciIterative;


public class TestProblem019NthFibonacciIterative {
    @Test void t() { assertEquals(55L, Problem019NthFibonacciIterative.fibN(10)); }
}
```

020. LruCacheSimple

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem020LruCacheSimple {
    public static class LRU<K,V> extends LinkedHashMap<K,V>{
        private final int cap;
        public LRU(int cap){ super(16,0.75f,true); this.cap=cap; }
        protected boolean removeEldestEntry(Map.Entry<K,V> e){ return size()>cap; }
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem020LruCacheSimple;


public class TestProblem020LruCacheSimple {
    @Test void t() { var l=new Problem020LruCacheSimple.LRU<Integer,Integer>(2); l.put(1,1); l.put(2,2); l.get(1); l.pu
}
```

Java Interview Handbook — Batch 3

Generated: 2025-09-13 02:24:57Z (UTC)

Java Theory & Cheatsheet

```
JAVA THEORY & CHEATSHEET
------------------------
- Types: int, long, double, boolean, char, String
- Collections: List, Set, Map; ArrayList, LinkedList, HashSet, HashMap; PriorityQueue
- Streams/Lambdas; Comparator; Optional
- Concurrency: Thread, synchronized, ExecutorService
- DSA: arrays, linked lists, stacks/queues, heaps, trees, graphs
- Testing: JUnit 5 @Test, Assertions.*
```

021. StackUsingList

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem021StackUsingList {
    public static class StackX<T>{LinkedList<T> s=new LinkedList<>(); public void push(T x){s.addLast(x);} public T pop
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem021StackUsingList;


public class TestProblem021StackUsingList {
    @Test void t() { var s=new Problem021StackUsingList.StackX<Integer>(); s.push(1); s.push(2); assertEquals(2,s.pop()
}
```

022. QueueUsingDeque

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem022QueueUsingDeque {
    public static class QueueX<T>{ArrayDeque<T> q=new ArrayDeque<>(); public void enqueue(T x){q.addLast(x);} public T
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem022QueueUsingDeque;


public class TestProblem022QueueUsingDeque {
    @Test void t() { var q=new Problem022QueueUsingDeque.QueueX<Integer>(); q.enqueue(1); q.enqueue(2); assertEquals(1,
}
```

023. MinHeapKSmallest

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem023MinHeapKSmallest {
    public static List<Integer> kSmallest(int[] a,int k){ PriorityQueue<Integer> pq=new PriorityQueue<>(); for(int x:a)
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem023MinHeapKSmallest;


import java.util.*;
public class TestProblem023MinHeapKSmallest {
    @Test void t() { assertEquals(java.util.Arrays.asList(1,2), Problem023MinHeapKSmallest.kSmallest(new int[]{3,1,2,4}
}
```

024. TopKFrequentWords

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem024TopKFrequentWords {
    public static List<String> topK(String[] words,int k){
        Map<String,Integer> c=new HashMap<>(); for(String w:words)c.put(w,c.getOrDefault(w,0)+1);
        PriorityQueue<String> pq=new PriorityQueue<>((a,b)->{int ca=c.get(a), cb=c.get(b); if(ca==cb) return b.compareT
        for(String w:c.keySet()){ pq.offer(w); if(pq.size()>k) pq.poll(); }
        List<String> res=new ArrayList<>(); while(!pq.isEmpty()) res.add(0,pq.poll()); return res;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem024TopKFrequentWords;


import java.util.*;
public class TestProblem024TopKFrequentWords {
    @Test void t() { assertEquals(java.util.List.of("a"), Problem024TopKFrequentWords.topK(new String[]{"a","b","a"},1)
}
```

025. AnagramGroups

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem025AnagramGroups {
    public static List<List<String>> group(String[] words){
        Map<String,List<String>> m=new HashMap<>();
        for(String w: words){ char[] cs=w.toCharArray(); java.util.Arrays.sort(cs); String k=new String(cs); m.computeI
        return new ArrayList<>(m.values());
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem025AnagramGroups;


public class TestProblem025AnagramGroups {
    @Test void t() { assertEquals(6, Problem025AnagramGroups.group(new String[]{"eat","tea","tan","ate","nat","bat"}).s
}
```

026. TwoSum

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem026TwoSum {
    public static int[] twoSum(int[] a,int target){
        Map<Integer,Integer> m=new HashMap<>();
        for(int i=0;i<a.length;i++){ int need=target-a[i]; if(m.containsKey(need)) return new int[]{m.get(need), i}; m.
        return null;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem026TwoSum;


public class TestProblem026TwoSum {
    @Test void t() { assertArrayEquals(new int[]{0,1}, Problem026TwoSum.twoSum(new int[]{2,7,11,15},9)); }
}
```

027. ThreeSum

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem027ThreeSum {
    public static List<List<Integer>> threeSum(int[] a){
        List<List<Integer>> res=new ArrayList<>(); java.util.Arrays.sort(a);
        for(int i=0;i<a.length;i++){ if(i>0&&a[i]==a[i-1]) continue; int l=i+1,r=a.length-1; while(l<r){ int s=a[i]+a[l
        return res;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem027ThreeSum;


public class TestProblem027ThreeSum {
    @Test void t() { assertFalse(Problem027ThreeSum.threeSum(new int[]{-1,0,1,2,-1,-4}).isEmpty()); }
}
```

028. LongestCommonPrefix

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem028LongestCommonPrefix {
    public static String lcp(String[] s){
        if(s==null||s.length==0) return "";
        String p=s[0];
        for(int i=1;i<s.length;i++){ while(!s[i].startsWith(p)){ p=p.substring(0, p.length()-1); if(p.isEmpty()) return
        return p;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem028LongestCommonPrefix;


public class TestProblem028LongestCommonPrefix {
    @Test void t() { assertEquals("fl", Problem028LongestCommonPrefix.lcp(new String[]{"flower","flow","flight"})); }
}
```

029. LongestIncreasingSubsequence

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem029LongestIncreasingSubsequence {
    public static int lis(int[] a){
        int[] tails=new int[a.length]; int size=0;
        for(int x: a){ int i=java.util.Arrays.binarySearch(tails,0,size,x); if(i<0)i=-(i+1); tails[i]=x; if(i==size) si
        return size;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem029LongestIncreasingSubsequence;


public class TestProblem029LongestIncreasingSubsequence {
    @Test void t() { assertEquals(4, Problem029LongestIncreasingSubsequence.lis(new int[]{10,9,2,5,3,7,101,18})); }
}
```

030. EditDistance

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem030EditDistance {
    public static int edit(String a,String b){
        int m=a.length(), n=b.length();
        int[][] dp=new int[m+1][n+1];
        for(int i=0;i<=m;i++) dp[i][0]=i;
        for(int j=0;j<=n;j++) dp[0][j]=j;
        for(int i=1;i<=m;i++) for(int j=1;j<=n;j++){
            if(a.charAt(i-1)==b.charAt(j-1)) dp[i][j]=dp[i-1][j-1];
            else dp[i][j]=1+Math.min(dp[i-1][j-1], Math.min(dp[i-1][j], dp[i][j-1]));
        }
        return dp[m][n];
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem030EditDistance;


public class TestProblem030EditDistance {
    @Test void t() { assertEquals(3, Problem030EditDistance.edit("kitten","sitting")); }
}
```

Java Interview Handbook — Batch 4

Generated: 2025-09-13 02:24:57Z (UTC)

Java Theory & Cheatsheet

```
JAVA THEORY & CHEATSHEET
------------------------
- Types: int, long, double, boolean, char, String
- Collections: List, Set, Map; ArrayList, LinkedList, HashSet, HashMap; PriorityQueue
- Streams/Lambdas; Comparator; Optional
- Concurrency: Thread, synchronized, ExecutorService
- DSA: arrays, linked lists, stacks/queues, heaps, trees, graphs
- Testing: JUnit 5 @Test, Assertions.*
```

031. UniquePathsGrid

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem031UniquePathsGrid {
    public static int uniquePaths(int m,int n){
        int[][] dp=new int[m][n];
        for(int i=0;i<m;i++) dp[i][0]=1;
        for(int j=0;j<n;j++) dp[0][j]=1;
        for(int i=1;i<m;i++) for(int j=1;j<n;j++) dp[i][j]=dp[i-1][j]+dp[i][j-1];
        return dp[m-1][n-1];
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem031UniquePathsGrid;


public class TestProblem031UniquePathsGrid {
    @Test void t() { assertEquals(6, Problem031UniquePathsGrid.uniquePaths(3,3)); }
}
```

032. CoinChangeMin

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem032CoinChangeMin {
    public static int coinChange(int[] coins,int amount){
        int INF=1_000_000, dp[]=new int[amount+1]; java.util.Arrays.fill(dp, INF); dp[0]=0;
        for(int a=1;a<=amount;a++) for(int c: coins) if(c<=a) dp[a]=Math.min(dp[a], dp[a-c]+1);
        return dp[amount]>=INF? -1: dp[amount];
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem032CoinChangeMin;


public class TestProblem032CoinChangeMin {
    @Test void t() { assertEquals(3, Problem032CoinChangeMin.coinChange(new int[]{1,2,5},11)); }
}
```

033. WordBreak

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem033WordBreak {
    public static boolean wordBreak(String s, java.util.Set<String> dict){
        boolean[] dp=new boolean[s.length()+1]; dp[0]=true;
        for(int i=1;i<=s.length();i++) for(int j=0;j<i;j++) if(dp[j] && dict.contains(s.substring(j,i))){ dp[i]=true; b
        return dp[s.length()];
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem033WordBreak;


import java.util.*;
public class TestProblem033WordBreak {
    @Test void t() { assertTrue(Problem033WordBreak.wordBreak("leetcode", new java.util.HashSet<>(java.util.List.of("le
}
```

034. LongestPalindromicSubstring

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem034LongestPalindromicSubstring {
    public static String lps(String s){
        if(s==null||s.isEmpty()) return "";
        int start=0,end=0;
        for(int i=0;i<s.length();i++){ int[] a=expand(s,i,i), b=expand(s,i,i+1); int[] best = (a[1]-a[0] > b[1]-b[0])?
        return s.substring(start, end+1);
    }
    static int[] expand(String s,int l,int r){ while(l>=0&&r<s.length()&&s.charAt(l)==s.charAt(r)){ l--; r++; } return
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem034LongestPalindromicSubstring;


public class TestProblem034LongestPalindromicSubstring {
    @Test void t() { String r=Problem034LongestPalindromicSubstring.lps("babad"); assertTrue(r.equals("bab")||r.equals(
}
```

035. SerializeDeserializeTree

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem035SerializeDeserializeTree {
    public static class Node{int val; Node left,right; Node(int v){val=v;}}
    public static String serialize(Node root){ StringBuilder sb=new StringBuilder(); ser(root,sb); return sb.toString()
    static void ser(Node n,StringBuilder sb){ if(n==null){sb.append("#,");return;} sb.append(n.val).append(','); ser(n.
    public static Node deserialize(String data){ String[] t=data.split(","); int[] i=new int[]{0}; return des(t,i);}
    static Node des(String[] t,int[] i){ if(i[0]>=t.length) return null; String v=t[i[0]++]; if(v.equals("#")||v.isEmpt
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem035SerializeDeserializeTree;


public class TestProblem035SerializeDeserializeTree {
    @Test void t(){
        var r=new Problem035SerializeDeserializeTree.Node(1); r.left=new Problem035SerializeDeserializeTree.Node(2); r.
        String s=Problem035SerializeDeserializeTree.serialize(r); assertNotNull(Problem035SerializeDeserializeTree.dese
    }
}
```

036. IsBalancedTree

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem036IsBalancedTree {
    static class Node{int v; Node l,r; Node(int v){this.v=v;}}
    public static boolean isBalanced(Node n){ return height(n)!=-1; }
    static int height(Node n){ if(n==null) return 0; int lh=height(n.l); if(lh==-1) return -1; int rh=height(n.r); if(r
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem036IsBalancedTree;


public class TestProblem036IsBalancedTree {
    @Test void t(){ var r=new Problem036IsBalancedTree.Node(1); r.l=new Problem036IsBalancedTree.Node(2); assertTrue(Pr
}
```

037. LowestCommonAncestor

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem037LowestCommonAncestor {
    static class Node{int v; Node l,r; Node(int v){this.v=v;}}
    public static Node lca(Node root, Node p, Node q){ if(root==null || root==p || root==q) return root; Node L=lca(roo
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem037LowestCommonAncestor;


public class TestProblem037LowestCommonAncestor {
    @Test void t(){ var r=new Problem037LowestCommonAncestor.Node(3); r.l=new Problem037LowestCommonAncestor.Node(5); r
}
```

038. TrieInsertSearch

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem038TrieInsertSearch {
    static class TrieNode{java.util.Map<Character,TrieNode> c=new java.util.HashMap<>(); boolean end=false;}
    public static class Trie{ TrieNode root=new TrieNode(); void insert(String w){ TrieNode n=root; for(char ch: w.toCh
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem038TrieInsertSearch;


public class TestProblem038TrieInsertSearch {
    @Test void t(){ var T=new Problem038TrieInsertSearch.Trie(); T.insert("hi"); assertTrue(T.search("hi")); assertFals
}
```

039. DijkstraSimple

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem039DijkstraSimple {
    public static Map<String,Integer> dijkstra(Map<String,List<String[]>> g,String src){
        Map<String,Integer> d=new HashMap<>(); d.put(src,0);
        PriorityQueue<String[]> pq=new PriorityQueue<>(Comparator.comparingInt(a->Integer.parseInt(a[0])));
        pq.offer(new String[]{"0",src});
        while(!pq.isEmpty()){ String[] cur=pq.poll(); int dist=Integer.parseInt(cur[0]); String u=cur[1]; if(dist> d.ge
            for(String[] e: g.getOrDefault(u, List.of())){ String v=e[0]; int w=Integer.parseInt(e[1]); int nd=dist+w;
        } return d;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem039DijkstraSimple;


import java.util.*;
public class TestProblem039DijkstraSimple {
    @Test void t(){ Map<String,List<String[]>> g=new HashMap<>(); g.put("A", List.of(new String[]{"B","1"})); g.put("B"
}
```

040. BfsGraph

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem040BfsGraph {
    public static List<Integer> bfs(Map<Integer,List<Integer>> g,int s){
        List<Integer> seen=new ArrayList<>(); ArrayDeque<Integer> q=new ArrayDeque<>(); q.add(s); seen.add(s);
        while(!q.isEmpty()){ int u=q.poll(); for(int v: g.getOrDefault(u, List.of())) if(!seen.contains(v)){ seen.add(v
        return seen;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem040BfsGraph;


import java.util.*;
public class TestProblem040BfsGraph {
    @Test void t(){ Map<Integer,List<Integer>> g=Map.of(1,List.of(2,3),2,List.of(4),3,List.of(),4,List.of()); assertEqu
}
```

Java Interview Handbook — Batch 5

Generated: 2025-09-13 02:24:57Z (UTC)

Java Theory & Cheatsheet

```
JAVA THEORY & CHEATSHEET
------------------------
- Types: int, long, double, boolean, char, String
- Collections: List, Set, Map; ArrayList, LinkedList, HashSet, HashMap; PriorityQueue
- Streams/Lambdas; Comparator; Optional
- Concurrency: Thread, synchronized, ExecutorService
- DSA: arrays, linked lists, stacks/queues, heaps, trees, graphs
- Testing: JUnit 5 @Test, Assertions.*
```

040. DfsGraphRecursive

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem040DfsGraphRecursive {
    public static List<Integer> dfs(Map<Integer, List<Integer>> g, int start){
        List<Integer> res = new ArrayList<>();
        Set<Integer> vis = new HashSet<>();
        dfsRec(g, start, vis, res);
        return res;
    }
    static void dfsRec(Map<Integer,List<Integer>> g, int u, Set<Integer> vis, List<Integer> res){
        if(vis.contains(u)) return;
        vis.add(u); res.add(u);
        for(int v: g.getOrDefault(u, List.of())) dfsRec(g, v, vis, res);
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem040DfsGraphRecursive;


import java.util.*;
public class TestProblem040DfsGraphRecursive {
    @Test void t(){
        Map<Integer,List<Integer>> g = Map.of(1,List.of(2,3), 2,List.of(4), 3,List.of(), 4,List.of());
        assertEquals(List.of(1,2,4,3), Problem040DfsGraphRecursive.dfs(g,1));
    }
}
```

041. SieveEratosthenes

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem041SieveEratosthenes {
    public static List<Integer> sieve(int n){
        boolean[] p=new boolean[n+1]; java.util.Arrays.fill(p,true); if(n>=0)p[0]=false; if(n>=1)p[1]=false;
        for(int i=2;i*i<=n;i++) if(p[i]) for(int j=i*i;j<=n;j+=i) p[j]=false;
        List<Integer> res=new ArrayList<>(); for(int i=2;i<=n;i++) if(p[i]) res.add(i); return res;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem041SieveEratosthenes;


import java.util.*;
public class TestProblem041SieveEratosthenes {
    @Test void t(){ assertEquals(List.of(2,3,5,7), Problem041SieveEratosthenes.sieve(10)); }
}
```

042. RotateMatrix90

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem042RotateMatrix90 {
    public static void rotate(int[][] m){
        int n=m.length;
        for(int i=0;i<n;i++) for(int j=i;j<n;j++){ int t=m[i][j]; m[i][j]=m[j][i]; m[j][i]=t; }
        for(int i=0;i<n;i++){ int l=0,r=n-1; while(l<r){ int t=m[i][l]; m[i][l]=m[i][r]; m[i][r]=t; l++; r--; } }
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem042RotateMatrix90;


public class TestProblem042RotateMatrix90 {
    @Test void t(){ int[][] m={ {1,2}, {3,4} }; Problem042RotateMatrix90.rotate(m); assertArrayEquals(new int[][]{ {3,1
}
```

043. FindMissingNumber

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem043FindMissingNumber {
    public static int missing(int[] a){
        int n=a.length; int expected=n*(n+1)/2, sum=0; for(int x:a) sum+=x; return expected-sum;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem043FindMissingNumber;


public class TestProblem043FindMissingNumber {
    @Test void t(){ assertEquals(2, Problem043FindMissingNumber.missing(new int[]{0,1,3})); }
}
```

044. FindDuplicate

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem044FindDuplicate {
    public static int findDuplicate(int[] a){
        int slow=a[0], fast=a[0];
        do{ slow=a[slow]; fast=a[a[fast]]; } while(slow!=fast);
        slow=a[0]; while(slow!=fast){ slow=a[slow]; fast=a[fast]; } return slow;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem044FindDuplicate;


public class TestProblem044FindDuplicate {
    @Test void t(){ assertEquals(2, Problem044FindDuplicate.findDuplicate(new int[]{1,3,4,2,2})); }
}
```

045. SearchInsertPosition

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem045SearchInsertPosition {
    public static int searchInsert(int[] a,int target){
        int lo=0,hi=a.length-1; while(lo<=hi){ int mid=(lo+hi)/2; if(a[mid]==target)return mid; if(a[mid]<target) lo=mi
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem045SearchInsertPosition;


public class TestProblem045SearchInsertPosition {
    @Test void t(){ assertEquals(2, Problem045SearchInsertPosition.searchInsert(new int[]{1,2,4,5},3)); }
}
```

046. IntervalMerge

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem046IntervalMerge {
    public static int[][] merge(int[][] intervals){
        java.util.Arrays.sort(intervals,(a,b)->Integer.compare(a[0],b[0]));
        java.util.List<int[]> res=new java.util.ArrayList<>();
        for(int[] in: intervals){
            if(res.isEmpty()|| res.get(res.size()-1)[1] < in[0]) res.add(in.clone());
            else res.get(res.size()-1)[1] = Math.max(res.get(res.size()-1)[1], in[1]);
        } return res.toArray(new int[0][]);
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem046IntervalMerge;


public class TestProblem046IntervalMerge {
    @Test void t(){ int[][] r=Problem046IntervalMerge.merge(new int[][]{ {1,3}, {2,6}, {8,10}, {15,18} }); assertEquals
}
```

047. BestTimeToBuySellStock

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem047BestTimeToBuySellStock {
    public static int maxProfit(int[] prices){
        int min=prices[0], best=0; for(int p: prices){ if(p<min) min=p; best=Math.max(best, p-min); } return best;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem047BestTimeToBuySellStock;


public class TestProblem047BestTimeToBuySellStock {
    @Test void t(){ assertEquals(5, Problem047BestTimeToBuySellStock.maxProfit(new int[]{7,1,5,3,6,4})); }
}
```

048. ProductOfArrayExceptSelf

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem048ProductOfArrayExceptSelf {
    public static int[] productExceptSelf(int[] a){
        int n=a.length; int[] res=new int[n]; int pref=1;
        for(int i=0;i<n;i++){ res[i]=pref; pref*=a[i]; }
        int suf=1; for(int i=n-1;i>=0;i--){ res[i]*=suf; suf*=a[i]; }
        return res;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem048ProductOfArrayExceptSelf;


public class TestProblem048ProductOfArrayExceptSelf {
    @Test void t(){ assertArrayEquals(new int[]{24,12,8,6}, Problem048ProductOfArrayExceptSelf.productExceptSelf(new in
}
```

049. UnionFind

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem049UnionFind {
    public static class UF{ int[] p, r; public UF(int n){ p=new int[n]; r=new int[n]; for(int i=0;i<n;i++)p[i]=i; } pub
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem049UnionFind;


public class TestProblem049UnionFind {
    @Test void t(){ var u=new Problem049UnionFind.UF(3); u.union(0,1); assertEquals(u.find(0), u.find(1)); }
}
```

Java Interview Handbook — Batch 6

Generated: 2025-09-13 02:24:57Z (UTC)

Java Theory & Cheatsheet

```
JAVA THEORY & CHEATSHEET
------------------------
- Types: int, long, double, boolean, char, String
- Collections: List, Set, Map; ArrayList, LinkedList, HashSet, HashMap; PriorityQueue
- Streams/Lambdas; Comparator; Optional
- Concurrency: Thread, synchronized, ExecutorService
- DSA: arrays, linked lists, stacks/queues, heaps, trees, graphs
- Testing: JUnit 5 @Test, Assertions.*
```

050. MatrixMultiply

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem050MatrixMultiply {
    public static int[][] multiply(int[][] A, int[][] B){
        int m=A.length, n=A[0].length, p=B[0].length;
        int[][] C=new int[m][p];
        for(int i=0;i<m;i++) for(int k=0;k<n;k++) if(A[i][k]!=0)
            for(int j=0;j<p;j++) C[i][j]+=A[i][k]*B[k][j];
        return C;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem050MatrixMultiply;


public class TestProblem050MatrixMultiply {
    @Test void t(){
        int[][] A={{1,2},{3,4}}, B={{5,6},{7,8}};
        int[][] R = Problem050MatrixMultiply.multiply(A,B);
        assertArrayEquals(new int[]{19,22}, R[0]);
        assertArrayEquals(new int[]{43,50}, R[1]);
    }
}
```

050. TopologicalSort

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem050TopologicalSort {
    public static java.util.List<Integer> topo(int n, int[][] edges){
        java.util.List<java.util.List<Integer>> g=new ArrayList<>(); for(int i=0;i<n;i++) g.add(new ArrayList<>());
        int[] indeg=new int[n]; for(int[] e: edges){ g.get(e[0]).add(e[1]); indeg[e[1]]++; }
        java.util.ArrayDeque<Integer> q=new java.util.ArrayDeque<>(); for(int i=0;i<n;i++) if(indeg[i]==0) q.add(i);
        java.util.List<Integer> res=new java.util.ArrayList<>(); while(!q.isEmpty()){ int u=q.poll(); res.add(u); for(i
        return res;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem050TopologicalSort;


public class TestProblem050TopologicalSort {
    @Test void t(){ assertEquals(4, Problem050TopologicalSort.topo(4, new int[][]{ {0,1}, {0,2}, {1,3}, {2,3} }).size()
}
```

051. SlidingWindowMax

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem051SlidingWindowMax {
    public static int[] maxSliding(int[] a,int k){
        int n=a.length; if(n==0||k==0) return new int[0];
        int[] res=new int[n-k+1]; java.util.Deque<Integer> dq=new java.util.ArrayDeque<>();
        for(int i=0;i<n;i++){ while(!dq.isEmpty() && dq.peekFirst()<=i-k) dq.pollFirst(); while(!dq.isEmpty() && a[dq.p
        return res;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem051SlidingWindowMax;


public class TestProblem051SlidingWindowMax {
    @Test void t(){ assertArrayEquals(new int[]{3,3,5,5,6,7}, Problem051SlidingWindowMax.maxSliding(new int[]{1,3,-1,-3
}
```

051. TransposeMatrix

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem051TransposeMatrix {
    public static int[][] transpose(int[][] M){
        int m=M.length, n=M[0].length;
        int[][] T=new int[n][m];
        for(int i=0;i<m;i++) for(int j=0;j<n;j++) T[j][i]=M[i][j];
        return T;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem051TransposeMatrix;


public class TestProblem051TransposeMatrix {
    @Test void t(){
        int[][] M={{1,2,3},{4,5,6}};
        int[][] T=Problem051TransposeMatrix.transpose(M);
        assertArrayEquals(new int[]{1,4}, T[0]);
        assertArrayEquals(new int[]{2,5}, T[1]);
        assertArrayEquals(new int[]{3,6}, T[2]);
    }
}
```

053. SpiralMatrix

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem053SpiralMatrix {
    public static java.util.List<Integer> spiral(int[][] m){
        List<Integer> res=new ArrayList<>();
        if(m.length==0) return res;
        int top=0, bot=m.length-1, left=0, right=m[0].length-1;
        while(top<=bot && left<=right){
            for(int j=left;j<=right;j++) res.add(m[top][j]); top++;
            for(int i=top;i<=bot;i++) res.add(m[i][right]); right--;
            if(top<=bot) for(int j=right;j>=left;j--) res.add(m[bot][j]); bot--;
            if(left<=right) for(int i=bot;i>=top;i--) res.add(m[i][left]); left++;
        }
        return res;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem053SpiralMatrix;


import java.util.*;
public class TestProblem053SpiralMatrix {
    @Test void t(){
        int[][] m={{1,2,3},{4,5,6},{7,8,9}};
        assertEquals(List.of(1,2,3,6,9,8,7,4,5), Problem053SpiralMatrix.spiral(m));
    }
}
```

055. MedianTwoSortedArraysSmall

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem055MedianTwoSortedArraysSmall {
    public static double median(int[] A, int[] B){
        int n=A.length+B.length;
        int i=0,j=0,prev=0,cur=0;
        for(int k=0;k<=n/2;k++){
            prev=cur;
            if(i<A.length && (j>=B.length || A[i]<=B[j])) cur=A[i++];
            else cur=B[j++];
        }
        if(n%2==1) return cur;
        return (prev+cur)/2.0;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem055MedianTwoSortedArraysSmall;


public class TestProblem055MedianTwoSortedArraysSmall {
    @Test void t(){
        assertEquals(2.0, Problem055MedianTwoSortedArraysSmall.median(new int[]{1,3}, new int[]{2}), 1e-9);
        assertEquals(2.5, Problem055MedianTwoSortedArraysSmall.median(new int[]{1,2}, new int[]{3,4}), 1e-9);
    }
}
```

060. MaxProfitKTransactions

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem060MaxProfitKTransactions {
    public static int maxProfit(int k, int[] prices){
        int n=prices.length; if(n==0||k==0) return 0;
        if(k>=n/2){ // unlimited
            int prof=0; for(int i=1;i<n;i++) if(prices[i]>prices[i-1]) prof+=prices[i]-prices[i-1]; return prof;
        }
        int[] buy=new int[k+1], sell=new int[k+1];
        for(int i=0;i<=k;i++){ buy[i]=Integer.MIN_VALUE/4; sell[i]=0; }
        for(int p: prices){
            for(int t=1;t<=k;t++){
                buy[t]=Math.max(buy[t], sell[t-1]-p);
                sell[t]=Math.max(sell[t], buy[t]+p);
            }
        }
        return sell[k];
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem060MaxProfitKTransactions;


public class TestProblem060MaxProfitKTransactions {
    @Test void t(){
        assertEquals(7, Problem060MaxProfitKTransactions.maxProfit(2, new int[]{3,2,6,5,0,3}));
    }
}
```

061. HeapSort

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem061HeapSort {
    public static void sort(int[] a){
        int n=a.length;
        for(int i=n/2-1;i>=0;i--) heapify(a,n,i);
        for(int i=n-1;i>=0;i--){
            int t=a[0]; a[0]=a[i]; a[i]=t;
            heapify(a,i,0);
        }
    }
    static void heapify(int[] a,int n,int i){
        int largest=i, l=2*i+1, r=2*i+2;
        if(l<n && a[l]>a[largest]) largest=l;
        if(r<n && a[r]>a[largest]) largest=r;
        if(largest!=i){ int t=a[i]; a[i]=a[largest]; a[largest]=t; heapify(a,n,largest); }
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem061HeapSort;


public class TestProblem061HeapSort {
    @Test void t(){
        int[] a={4,1,3,2}; Problem061HeapSort.sort(a);
        assertArrayEquals(new int[]{1,2,3,4}, a);
    }
}
```

063. CountingSortSimple

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem063CountingSortSimple {
    public static int[] sort(int[] a, int maxVal){
        int[] count=new int[maxVal+1];
        for(int x: a) count[x]++;
        int idx=0;
        for(int v=0; v<=maxVal; v++) while(count[v]-- > 0) a[idx++]=v;
        return a;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem063CountingSortSimple;


public class TestProblem063CountingSortSimple {
    @Test void t(){
        int[] a={3,1,2,1,0}; assertArrayEquals(new int[]{0,1,1,2,3}, Problem063CountingSortSimple.sort(a,3));
    }
}
```

064. RadixSortSimple

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem064RadixSortSimple {
    public static int[] sort(int[] a){
        int max=0; for(int x: a) if(x>max) max=x;
        int exp=1; int n=a.length;
        int[] out=new int[n];
        while(max/exp>0){
            int[] cnt=new int[10];
            for(int i=0;i<n;i++) cnt[(a[i]/exp)%10]++;
            for(int i=1;i<10;i++) cnt[i]+=cnt[i-1];
            for(int i=n-1;i>=0;i--) out[--cnt[(a[i]/exp)%10]] = a[i];
            System.arraycopy(out,0,a,0,n);
            exp*=10;
        }
        return a;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem064RadixSortSimple;


public class TestProblem064RadixSortSimple {
    @Test void t(){
        int[] a={170,45,75,90,802,24,2,66}; assertArrayEquals(new int[]{2,24,45,66,75,90,170,802}, Problem064RadixSortS
    }
}
```

Java Interview Handbook — Batch 7

Generated: 2025-09-13 02:24:57Z (UTC)

Java Theory & Cheatsheet

```
JAVA THEORY & CHEATSHEET
------------------------
- Types: int, long, double, boolean, char, String
- Collections: List, Set, Map; ArrayList, LinkedList, HashSet, HashMap; PriorityQueue
- Streams/Lambdas; Comparator; Optional
- Concurrency: Thread, synchronized, ExecutorService
- DSA: arrays, linked lists, stacks/queues, heaps, trees, graphs
- Testing: JUnit 5 @Test, Assertions.*
```

094. MinWindowSubstring

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem094MinWindowSubstring {
    public static String minWindow(String s, String t){
        if(t.length()==0) return "";
        int[] need=new int[128]; int required=0;
        for(char c: t.toCharArray()){ if(need[c]==0) required++; need[c]++; }
        int[] have=new int[128]; int formed=0;
        int l=0, bestLen=Integer.MAX_VALUE, bestL=0;
        for(int r=0;r<s.length();r++){
            char c=s.charAt(r); have[c]++;
            if(have[c]==need[c] && need[c]>0) formed++;
            while(formed==required){
                if(r-l+1<bestLen){bestLen=r-l+1; bestL=l;}
                char lc=s.charAt(l); have[lc]--; if(have[lc]<need[lc] && need[lc]>0) formed--; l++;
            }
        }
        return bestLen==Integer.MAX_VALUE? "": s.substring(bestL,bestL+bestLen);
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem094MinWindowSubstring;


public class TestProblem094MinWindowSubstring {
    @Test void t(){ assertEquals("BANC", Problem094MinWindowSubstring.minWindow("ADOBECODEBANC","ABC")); }
}
```

095. MaxRectangleHistogram

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem095MaxRectangleHistogram {
    public static int largestRectangleArea(int[] heights){
        int n=heights.length, max=0;
        java.util.Deque<Integer> st=new java.util.ArrayDeque<>();
        for(int i=0;i<=n;i++){
            int h = (i==n)? 0: heights[i];
            while(!st.isEmpty() && h < heights[st.peek()]){
                int height=heights[st.pop()];
                int left = st.isEmpty()? -1: st.peek();
                int width = i - left - 1;
                max = Math.max(max, height*width);
            }
            st.push(i);
        }
        return max;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem095MaxRectangleHistogram;


public class TestProblem095MaxRectangleHistogram {
    @Test void t(){ assertEquals(10, Problem095MaxRectangleHistogram.largestRectangleArea(new int[]{2,1,5,6,2,3})); }
}
```

098. UrlValidation

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.regex.*;
public class Problem098UrlValidation {
    private static final Pattern P = Pattern.compile("^(https?://)?([\w.-]+)(:[0-9]+)?(/.*)?$");
    public static boolean isValid(String url){ return url!=null && P.matcher(url).matches(); }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem098UrlValidation;


public class TestProblem098UrlValidation {
    @Test void t(){ assertTrue(Problem098UrlValidation.isValid("https://example.com")); assertFalse(Problem098UrlValida
}
```

099. EmailValidationRegex

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.regex.*;
public class Problem099EmailValidationRegex {
    private static final Pattern P = Pattern.compile("^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$");
    public static boolean isValid(String email){ return email!=null && P.matcher(email).matches(); }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem099EmailValidationRegex;


public class TestProblem099EmailValidationRegex {
    @Test void t(){ assertTrue(Problem099EmailValidationRegex.isValid("a@b.com")); assertFalse(Problem099EmailValidatio
}
```

101. FindKthSmallestBst

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem101FindKthSmallestBst {
    public static class Node{int v; Node l,r; Node(int v){this.v=v;}}
    public static Integer kthSmallest(Node root, int k){
        java.util.Deque<Node> st=new java.util.ArrayDeque<>();
        Node cur=root; int count=0;
        while(cur!=null || !st.isEmpty()){
            while(cur!=null){ st.push(cur); cur=cur.l; }
            cur=st.pop();
            if(++count==k) return cur.v;
            cur=cur.r;
        }
        return null;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem101FindKthSmallestBst;


public class TestProblem101FindKthSmallestBst {
    @Test void t(){
        var r=new Problem101FindKthSmallestBst.Node(2); r.l=new Problem101FindKthSmallestBst.Node(1); r.r=new Problem10
        assertEquals(2, Problem101FindKthSmallestBst.kthSmallest(r,2));
    }
}
```

102. CountSetBits

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem102CountSetBits {
    public static int countBits(int x){
        int c=0; while(x!=0){ x&=x-1; c++; } return c;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem102CountSetBits;


public class TestProblem102CountSetBits {
    @Test void t(){ assertEquals(3, Problem102CountSetBits.countBits(0b1011)); }
}
```

103. HammingDistance

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


public class Problem103HammingDistance {
    public static int hamming(int a, int b){
        int x=a^b, c=0; while(x!=0){ x&=x-1; c++; } return c;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem103HammingDistance;


public class TestProblem103HammingDistance {
    @Test void t(){ assertEquals(2, Problem103HammingDistance.hamming(1,4)); }
}
```

104. GrayCode

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem104GrayCode {
    public static java.util.List<Integer> grayCode(int n){
        List<Integer> res=new ArrayList<>(); res.add(0);
        for(int i=0;i<n;i++){
            int add = 1<<i;
            for(int j=res.size()-1;j>=0;j--) res.add(res.get(j)+add);
        }
        return res;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem104GrayCode;


import java.util.*;
public class TestProblem104GrayCode {
    @Test void t(){ assertEquals(List.of(0,1,3,2), Problem104GrayCode.grayCode(2)); }
}
```

105. PowerSetIterative

Statement: Mirror of the Python problem; Java implementation.
Explanation: Brute-force then optimized approach; include complexity.

```
package problems;


import java.util.*;
public class Problem105PowerSetIterative {
    public static java.util.List<java.util.List<Integer>> powerSet(int[] nums){
        List<List<Integer>> res=new ArrayList<>(); res.add(new ArrayList<>());
        for(int x: nums){
            int sz=res.size();
            for(int i=0;i<sz;i++){ List<Integer> n=new ArrayList<>(res.get(i)); n.add(x); res.add(n); }
        }
        return res;
    }
}
```

```
package tests;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import problems.Problem105PowerSetIterative;


import java.util.*;
public class TestProblem105PowerSetIterative {
    @Test void t(){ assertEquals(8, Problem105PowerSetIterative.powerSet(new int[]{1,2,3}).size()); }
}
```