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")); }
}
```