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
}
```