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)); }
}
```