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
}
```