# 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.

```java
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;
    }
}
```

```java
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)); }
}
```