ArrayToTwoPart

```java
/**
 * Created by lpf on 2021/5/9.
 */
public class ArrayToTwoPart {

    public static boolean isEven(int n) {
        return (n & 1) == 0;
    }

    public static void recorder(int[] arr, int low, int high) {

        if (arr == null) {
            return;
        }

        while (low < high) {
            while (low < high && !isEven(arr[low])) {
                low++;
            }

            while (low < high && isEven(arr[high])) {
                high--;
            }

            if (low < high) {
                int temp = arr[low];
                arr[low] = arr[high];
                arr[high] = temp;
            }
        }
    }


    public static void main(String[] args) {

        int[] arr = new int[]{2, 3, 1, 4, 5, 6};
        recorder(arr, 0, arr.length - 1);

        for (int i = 0; i < arr.length; i++) {
            System.out.println(arr[i]);
        }
    }
}
```

CombineArray

```java
/**
 * Created by lpf on 2021/5/8.
 */
public class CombineArray {

    public static void CombineArray(int[] big, int originBigLength, int[] small) {

        if (big == null || small == null) {
            return;
        }

        int newLength = originBigLength + small.length;

        int indexOfBig = originBigLength - 1;

        int indexOfSmall = small.length - 1;

        int indexOfNew = newLength - 1;

        while (indexOfBig >= 0 && indexOfNew >= indexOfBig) {

            int currentBig = big[indexOfBig];

            int currentSmall = small[indexOfSmall];

            if (currentBig == currentSmall) {
                big[indexOfNew--] = currentBig;
                big[indexOfNew--] = currentBig;
                indexOfBig--;
                indexOfSmall--;
            } else if (currentBig > currentSmall) {
                big[indexOfNew--] = currentBig;
                indexOfBig--;
            } else {
                big[indexOfNew--] = currentSmall;
                indexOfSmall--;
            }

            if (indexOfSmall < 0) {
                break;
            }
        }
    }

    public static void CombineArray2(int[] big, int originBigLength, int[] small) {

        if (big == null || small == null) {
            return;
        }

        int newLength = originBigLength + small.length;
        int indexOfNew = newLength - 1;

        int indexOfBig = originBigLength - 1;
        int indexOfSmall = small.length - 1;

        while (indexOfSmall >= 0 && indexOfNew > indexOfSmall) {

            int currentBig = big[indexOfBig];
            int currentSmall = small[indexOfSmall];

            if (currentBig == currentSmall) {
                big[indexOfNew--] = currentBig;
                big[indexOfNew--] = currentBig;
                indexOfSmall--;
                indexOfBig--;

            } else if (currentBig > currentSmall) {
                big[indexOfNew--] = currentBig;
                indexOfBig--;
            } else {
                big[indexOfNew--] = currentSmall;
                indexOfSmall--;
            }

            if (indexOfSmall < 0) {
                break;
            }
        }
    }


    public static void main(String[] args) {

        int[] big = new int[100];
        int[] temp = new int[]{1, 2, 3, 5, 7, 8};
        System.arraycopy(temp, 0, big, 0, temp.length);
        int[] small = new int[]{1, 2, 3, 4, 8, 9};


        CombineArray2(big, temp.length, small);

        for (int i = 0; i < big.length; i++) {
            System.out.println(big[i]);
        }
    }
}
```

ConstructTree

```java
/**
 * Created by lpf on 2021/5/9.
 *
 * 由前序和中序筹够二叉树
 */
public class ConstructTree {

    static class TreeNode {
        int value;
        TreeNode left;
        TreeNode right;
    }



    public static TreeNode construct(int[] preOrder, int[] inOrder , int length) {

        if(preOrder == null || inOrder == null || length <=0 ){
            return null;
        }

        try {
            return constructCore(preOrder, 0, length-1, inOrder, 0, length -1);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

//    public static TreeNode constructCore(int[] preOrder, int preOrderStart, int preOr
//
//
//        System.out.println("preOrderStart="+preOrderStart +" preOrderEnd = "+ preOrde
//
//
//
//        int rootValue = preOrder[preOrderStart];
//        TreeNode root = new TreeNode();
//        root.value = rootValue;
//        root.left = null;
//        root.right = null;
//
//        if(preOrderStart == preOrderEnd && inOrderStart == inOrderEnd && preOrder[pre
//            return root;
//        }
//
//        int indexInOrder = inOrderStart;
//        while(indexInOrder <= inOrderEnd && inOrder[indexInOrder]!= rootValue){
//            indexInOrder++;
//        }
//
```

```
//        int leftLength = indexInOrder - inOrderStart;
//        int preOrderSubEnd = preOrderStart + leftLength;
//        if(leftLength >0){
//            root.left = constructCore(preOrder, preOrderStart+1, preOrderSubEnd, inOr
//        }
//
//        if(leftLength < preOrderEnd - preOrderStart) {
//            System.out.println();
//            root.right = constructCore(preOrder, preOrderSubEnd+1, preOrderEnd, inOrd
//        }
//
//        return root;
//
//    }


    public static TreeNode constructCore(int[] preOrder, int preOrderStart, int preOrde
                                         int[] inOrder, int inOrderStart, int inOrderEn
        TreeNode root = new TreeNode();
        root.value = preOrder[preOrderStart];
        root.left = null;
        root.right = null;


        if(preOrderStart == preOrderEnd){
            if(inOrderStart == inOrderEnd && preOrder[preOrderStart] == inOrder[inOrderS
                return root;
            } else {
                throw new Exception("输入不合法");
            }
        }

        int indexOfRoot = inOrderStart;
        while(indexOfRoot <= inOrderEnd && inOrder[indexOfRoot] != root.value){
            indexOfRoot++;
        }

        int leftLength = indexOfRoot - inOrderStart;
        int preOrderSubEnd = preOrderStart + leftLength;

        if(leftLength > 0){
            root.left = constructCore(preOrder, preOrderStart+1, preOrderSubEnd, inOrde
        }

        if(leftLength < preOrderEnd - preOrderStart) {
            root.right = constructCore(preOrder, preOrderSubEnd +1, preOrderEnd, inOrde
        }

        return root;
    }
    public static void traverseTree(TreeNode root){
        if(root == null){
            return;
        }

        traverseTree(root.left);
        traverseTree(root.right);
        System.out.println(root.value);

    }

    public static void main(String[] args) {
        int length = 8;
        int preorder[] = {1, 2, 4, 7, 3, 5, 6, 8};
        int inorder[] = {4, 7, 2, 1, 5, 3, 8, 6};

        TreeNode root = construct(preorder, inorder ,length);

       traverseTree(root);
    }
}
```

DeleteDuplication

```java
/**
 * Created by lpf on 2021/5/9.
 *
 * 删除排序链表中重复的节点
 */
public class DeleteDuplication {

    static class ListNode {
        int value;
        ListNode next;
    }

    //    public static ListNode deleteDuplication(ListNode head) {
//
//        if (head == null) {
//            return null;
//        }
//
//        ListNode preNode = null;
//        ListNode pNode = head;
//
//        while (pNode != null) {
//            ListNode pNext = pNode.next;
//
//            boolean needDelete = false;
//
//            if (pNext != null && pNext.value == pNode.value) {
//                needDelete = true;
//            }
//
//            if (!needDelete) {
//                preNode = pNode;
//                pNode = pNode.next;
//            } else {
//
//                int value = pNode.value;
//                ListNode delete = pNode;
//                while (delete != null && delete.value == value) {
//                    pNext = delete.next;
//                    delete = null;
//                    delete = pNext;
//                }
//
//                if (preNode == null) {
//                    head = pNext;
//                } else {
//                    preNode.next = pNext;
//                }
//
//                pNode = pNext;
//
//              }
//          }
//
//          return head;
//
//      }
    public static ListNode deleteDuplication(ListNode head) {

      if(head == null){
          return null;
      }

        ListNode preNode = null;
        ListNode pNode = head;

        while(pNode != null){


            ListNode pNext = pNode.next;

            boolean needDelete = false;

            if(pNext != null && pNext.value == pNode.value ){
                needDelete = true;
            }

            if(!needDelete){

                preNode = pNode;
                pNode = pNode.next;

            } else {

                int value = pNode.value;
                ListNode delete = pNode;

                while(delete != null && delete.value == value){
                    pNext = delete.next;
                    delete = pNext;
                }

                if(preNode == null){
                    head = pNext;
                } else {
                    preNode.next = pNext;
                }

                pNode = pNext;
            }
        }
        return head;

    }


    public static void main(String[] args) {


        ListNode one = new ListNode();
        ListNode two = new ListNode();
        ListNode three = new ListNode();
        ListNode four = new ListNode();
        ListNode five = new ListNode();

        one.value = 1;
        one.next = two;

        two.value = 1;
        two.next = three;

        three.value = 2;
        three.next = four;

        four.value = 4;
        four.next = five;

        five.value = 3;
        five.next = null;

        ListNode head = deleteDuplication(one);

        while (head != null) {
            System.out.println(head.value);
            head = head.next;
        }
    }
}
```

DeleteListNode

```java
/**
 * Created by lpf on 2021/5/9.
 */
public class DeleteListNode {

    static class ListNode {
        int value;
        ListNode next;

    }

    public static ListNode deleteNode(ListNode head, ListNode delete){

        if(head == null || delete == null){
            return null;
        }

        // 不是尾结点
        if(delete.next != null){

            ListNode nextNode = delete.next;

            delete.value = nextNode.value;
            delete.next = nextNode.next;

            nextNode = null;

        } else if(head == delete){
            // 只有一个节点

            head = null;
            delete = null;

        } else {

            // 多个节点,还是尾结点
            ListNode pNode = head;
            while(pNode.next != delete){
                pNode = pNode.next;
            }
            pNode.next = null;
            delete = null;
        }

        return head;
    }


    public static void main(String[] args) {

        ListNode one = new ListNode();
        ListNode two = new ListNode();
        ListNode three = new ListNode();
        ListNode four = new ListNode();

        one.value = 1;
        one.next = two;

        two.value = 2;
        two.next = three;

        three.value = 3;
        three.next = four;

        four.value = 4;
        four.next = null;

        ListNode result = deleteNode(one, one);

        while(result!=null){
            System.out.println(result.value);
            result= result.next;
        }



    }
}
```

FindNextTreeNode

```
/**
 * Created by lpf on 2021/5/9.
 *
 * 一个二叉树,一个节点,中序遍历找下一个节点,每个节点有左右指针和父指针
 */
public class FindNextTreeNode {

    static class TreeNode {
        int value;
        TreeNode left;
        TreeNode right;
        TreeNode parent;
    }

    public static TreeNode createTreeNode(int val){
        TreeNode node = new TreeNode();
        node.value = val;
        node.left = null;
        node.right = null;
        node.parent = null;
        return node;
    }

    public static void connectTreeNodes(TreeNode parent, TreeNode left, TreeNode right)
        if(parent!=null){
            parent.left = left;
            parent.right = right;
        }

        if(left!=null){
            left.parent = parent;
        }

        if(right!=null){
            right.parent = parent;
        }

    }

//    public static TreeNode findNextNode(TreeNode target) {
//
//        if (target == null) {
//            return null;
//        }
//
//
//        TreeNode next = null;
//
//        if(target.right!=null){
//
//            TreeNode parent = target.right;
//              while(parent.left != null){
//                  parent = parent.left;
//              }
//
//              next = parent;
//          } else if(target.parent != null){
//
//              TreeNode parent = target.parent;
//              TreeNode current = target;
//
//              while(parent!=null && parent.left != current){
//                  current = parent;
//                  parent = parent.parent;
//              }
//
//              next = parent;
//
//          }
//
//          System.out.println("下一个节点是:"+next.value);
//          return next;
//      }

    public static TreeNode findNextNode(TreeNode target) {

        if(target == null) {
            return null;
        }

        TreeNode next = null;

        if(target.right!=null){

            TreeNode parent = target.right;
            while(parent.left!=null){
                parent =parent.left;
            }
            next = parent;

        } else {

            TreeNode parent = target.parent;
            TreeNode current = target;
            while(parent!=null && parent.left != current){
                current = parent;
                parent = parent.parent;
            }

            next = parent;

        }
        if(next!=null){
            System.out.println("下一个节点"+next.value);
        } else {
            System.out.println("没有下一个节点");
        }
        return next;
    }

    public static void main(String[] args) {
        TreeNode pNode8 = createTreeNode(8);
        TreeNode pNode6 = createTreeNode(6);
        TreeNode pNode10 = createTreeNode(10);
        TreeNode pNode5 = createTreeNode(5);
        TreeNode pNode7 = createTreeNode(7);
        TreeNode pNode9 = createTreeNode(9);
        TreeNode pNode11 = createTreeNode(11);

        connectTreeNodes(pNode8, pNode7, null);
        connectTreeNodes(pNode7, pNode6, null);
        connectTreeNodes(pNode6, pNode5, null);

        findNextNode(pNode8);
    }

}
```

FindSingleDisapperChar

```java
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.Map;

/**
 * Created by lpf on 2021/5/14.
 */
public class FindSingleDisapperChar {

    public static void main(String[] args) {

        String str = "antbaccdeff";

        findFirstDisapperChar(str);
    }

    private static void findFirstDisapperChar(String str) {

        if (str.isEmpty()) {
            return;
        }

        HashMap<String, Integer> map = new LinkedHashMap<>(256);

        char[] charArray = str.toCharArray();
        for (int i = 0; i < charArray.length; i++) {

            String current = String.valueOf(charArray[i]);
            int count = map.getOrDefault(current, 0);

            map.put(current, ++count);

        }

        for (Map.Entry<String, Integer> entry : map.entrySet()) {
            if (entry.getValue() == 1) {
                System.out.println(entry.getKey());
                break;
            }
        }

    }
}
```

Matrix

```java
/**
 * Created by lpf on 2021/5/8.
 */
public class Matrix {

    public static boolean find(int[][]matrix, int rows, int columns, int target) {
       boolean found = false;

        if(matrix == null || rows <=0 || columns<=0){
            return found;
        }

        int row = 0;
        int column = columns -1;
        while(row<rows && column >0){
            int current = matrix[row][column];
            if(current == target){
                found = true;
                System.out.println("当前row ="+row +" column = "+ column);
                break;
            }else if(current >target){
                --column;
            } else {
                ++row;
            }
        }

        return found;
    }

    public static void main(String[] args) {

        // int[][] matrix = new int[][] {{1,3,4},{2,4,6},{3,5,7}};

        // find(matrix, 3,3,5);
    }
}
```

MockQueueTest

```java
import java.util.Stack;

/**
 * Created by lpf on 2021/5/9.
 */
public class MockQueueTest {

    static class MockQueue<T> {

        private  Stack<T> inputStack = new Stack<>();
        private  Stack<T> outputStack = new Stack<>();


        public  T popQueue() {
            if (!outputStack.isEmpty()) {
                return outputStack.pop();
            }
            if (inputStack.isEmpty()) {
                System.out.println("没有元素了");
                return null;
            } else {
                while(inputStack.size() > 0){
                    outputStack.push(inputStack.pop());
                }
                return outputStack.pop();
            }
        }


        private  void pushToInputStack(T value){
            inputStack.push(value);
        }
    }




    public static void main(String[] args) {

        MockQueue<Integer> mockQueue = new MockQueue<Integer>();


        mockQueue.pushToInputStack(3);
        mockQueue.pushToInputStack(1);
        mockQueue.pushToInputStack(2);
        mockQueue.pushToInputStack(4);
        mockQueue.pushToInputStack(5);

        System.out.println(mockQueue.popQueue());

        mockQueue.pushToInputStack(6);
        System.out.println(mockQueue.popQueue());
    }
}
```

Power

```
/**
 * Created by lpf on 2021/5/9.
 */
public class Power {

    public static double powerWithN(double base , int n){

        if(base == 0){
            return 0;
        }

        if(n == 0){
            return 1;
        }

        if(n == 1){
            return base;
        }

        double result = powerWithN(base , n>>1);
        result = result* result;


        if((n & 1) == 1){
            result = result * base;
        }

        return result;

    }

    public static void main(String[] args) {

        System.out.println(powerWithN(2, 5));
    }
}
```

QuickSort

```java
/**
 * Created by lpf on 2021/5/8.
 */
public class QuickSort {

    public static int partition(int[] nums, int low, int high) {

        int p = nums[low];

        while (low < high) {
            while (low < high && nums[high] >= p) {
                high--;
            }
            nums[low] = nums[high];

            while (low < high && nums[low] <= p) {
                low++;
            }

            nums[high] = nums[low];
        }

        nums[low] = p;
        return low;
    }

    public static void quickSort(int[] nums, int low, int high) {

        if (low > high) {
            return;
        }

        int p = partition(nums, low, high);
        quickSort(nums, low, p - 1);
        quickSort(nums, p + 1, high);
    }

    public static void main(String[] args) {

        int[] nums = new int[]{2, 1, 4, 3, 6, 5, 4, 3};
        quickSort(nums, 0, nums.length - 1);

        for (int i = 0; i < nums.length; i++) {
            System.out.println(nums[i]);
        }

    }
}
```

ReplaceSpace

```java
/**
 * Created by lpf on 2021/5/8.
 */
public class ReplaceSpace {

    public static void replaceBlank(char[] string, int length){

        if(string == null || length <= 0){
            return;
        }

        int originLength = 0;
        int numberOfBlank = 0;
        int i = 0;
        while(string[i]!='\0'){

            ++originLength;

            if(string[i] == ' '){
                ++numberOfBlank;
            }

            i++;
        }

        int newNumber = originLength + numberOfBlank *2;

        if(newNumber > length){
            return;
        }

        int indexOfOrigin = originLength;
        int indexOfNew = newNumber;
        while(indexOfOrigin >=0 && indexOfNew >indexOfOrigin){
            if(string[indexOfOrigin] == ' '){
                string[indexOfNew--] = '0';
                string[indexOfNew--] = '2';
                string[indexOfNew--] = '%';
            }else{

                string[indexOfNew--] = string[indexOfOrigin];
            }

            --indexOfOrigin;
        }
    }

    public static void replaceBlank2(char[] string, int length){
        if(string == null || length <=0){
            return;
        }
        int blankCount = 0;
        int originLength = 0;
        int i = 0;
        while(string[i]!= '\0'){
            ++originLength;

            if(string[i] == ' '){
                ++blankCount;
            }
            ++i;
        }

        int newLength = originLength + blankCount *2;

        if(newLength > length) {
            return;
        }

        int indexOfNew = newLength-1;
        int indexOfOrigin = originLength-1;

        while(indexOfOrigin >=0 && indexOfNew > indexOfOrigin){
            if(string[indexOfOrigin] == ' '){
                string[indexOfNew--] = '0';
                string[indexOfNew--] = '2';
                string[indexOfNew--] = '%';
            }else {
                string[indexOfNew--] = string[indexOfOrigin];
            }

            indexOfOrigin--;
        }
    }

    public static void main(String[] args) {

        char[] string = new char[100];
        String str = "123 456 789";
        char[] temp = str.toCharArray();

        System.out.println(str.length());
        System.out.println(temp.length);

        System.arraycopy(temp, 0, string, 0, str.length());

        replaceBlank2(string, 100);

        System.out.println(string);

    }
```

```
}
```

Singleton

```java
/**
 * Created by lpf on 2021/5/8.
 */
public class Singleton {

    private Singleton(){}

    private static class Instance {
        private static Singleton singleton = new Singleton();
    }

    public static Singleton getInstance(){
        return Instance.singleton;
    }
}
```

SortAges

```java
/**
 * Created by lpf on 2021/5/9.
 */
public class SortAges {

    public static void sortAges(int[] ages, int length){

        if(ages == null || length <=0 ){
            return;
        }

        int oldestAge = 99;
        int[] timesOfAge = new int[100];

        for(int i = 0; i<timesOfAge.length; i++){
            timesOfAge[i] = 0;
        }

        for(int i = 0; i<length; i++){
            int age = ages[i];
            ++timesOfAge[age];
        }

        int index = 0;
        for(int i = 0; i<= oldestAge; i++){
            for(int j = 0; j<timesOfAge[i]; j++){
                ages[index] = i;
                ++index;
            }
        }

    }

    public static void main(String[] args) {
        int[] ages = new int[]{20,30,30,20,40,50,50,40,60,20,30};
        sortAges(ages,ages.length);

        for(int i = 0; i<ages.length; i++){
            System.out.println(ages[i]);
        }
    }
}
```