面试题 2：实现单例模式

1. 饿汉式单例类

```
public class SingletonClass{
    private static final SingletonClass instance=new SingletonClass();
    //私有构造函数
    private SingletonClass(){}
    public static SingletonClass getInstance(){
      return instance;
}
}
```

2. 懒汉式单例模式

```
public class SingletonClass{
    private static SingletonClass instance=null;
    //私有构造函数
    private SingletonClass(){}
  public synchronized  static SingletonClass getInstance(){
     if(instance==null){
       instance=new SingletonClass();
}
return instance;
}
}
```

面试题 3：二维数组中的查找

题目描述：一个二维数组，每一行从左到右递增，每一列从上到下递增. 输入一个二维数组和一个整数，判断数组中是否含有整数。

```
public class Find{
   public static boolean find(int[][] array,int number){
       if(array==null){
          return false;
        }
    int column=array[0].length-1;
   int row=0;
   while (row<array.length && column>=0){
               if(array[row][column]==number){
```

```
                return true;
            }
            if(array[row][column]>number){
                column--;
            }else{
                row++;
            }
        }
        return false;
    }
    public static void main(String args[]){
         int[][] testarray=new int[4][4];
            testarray[0][0]=1;
            testarray[0][1]=2;
            testarray[0][2]=8;
            testarray[0][3]=9;
            testarray[1][0]=2;
            testarray[1][1]=4;
            testarray[1][2]=9;
            testarray[1][3]=12;
            testarray[2][0]=4;
            testarray[2][1]=7;
            testarray[2][2]=10;
            testarray[2][3]=13;
            testarray[3][0]=6;
            testarray[3][1]=8;
            testarray[3][2]=11;
            testarray[3][3]=15;
            System.out.println(find(testarray, 1));
        }
}
```

面试题 4：替换空格

题目：请实现一个函数，把字符串中的每个空格替换成“%20”。

```
   public class ReplaceBlank {
       public static void main(String args[]){
         String s="We are happy.";
       System.out.println(replaceBlank(s));
    }
    public String replaceBlank(String input){
       if(input==null)
          return null;
       StringBuffer outputBuffer=new StringBuffer();
```

```
        for(int i=0;i<input.length();i++){
            if(input.charAt(i)==' '){
                outputBuffer.append("%");
                outputBuffer.append("2");
                outputBuffer.append("0");
            }else {
                outputBuffer.append(String.valueOf(input.charAt(i)));
            }
        }
        return new String(outputBuffer);
    }
}
```

面试题 5：从尾到头打印链表

题目：输入一个链表的头结点，从尾到头反过来打印出每个结点的值。

方法一：非递归的方式

```
public class PrintListReverse{

public static void main (String args[])
    {
        ListNode node1=new ListNode();
        ListNode node2=new ListNode();
        ListNode node3=new ListNode();
        node1.data=1;
        node2.data=2;
        node3.data=3;
        node1.next=node2;
        node2.next=node3;
        printListReversversingly test=new printListReversversingly();
        test.printListReverse(node1);
    }
   public static void printListReverse(ListNode headNode){
      Stack<ListNode> stack=new Stack<ListNode>();
        while(headNode!=null){
            stack.push(headNode);
            headNode=headNode.next;
        }
        while(!stack.isEmpty()){
            System.out.println(stack.pop().data);
        }
```

```
}
}
```

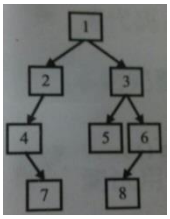
方法二:递归方式实现

```
public class PrintListReverse{

public static void main (String args[])
    {
        ListNode node1=new ListNode();
        ListNode node2=new ListNode();
        ListNode node3=new ListNode();
        node1.data=1;
        node2.data=2;
        node3.data=3;
        node1.next=node2;
        node2.next=node3;
        printListReversversingly test=new printListReversversingly();
        test.printListReverse(node1);
    }
   public static void printListReverse(ListNode headNode){
        if(headNode!=null){
            if(headNode.next!=null){
                printListReverse(headNode.next);
}
}
        System.out.println(headNode.data);
   }
}
```

面试题 6：重建二叉树

题目描述：输入二叉树的前序遍历和中序遍历的结果，重建出该二叉树。假设前序遍历和中序遍历结果中都不包含重复的数字，例如输入的前序遍历序列{1,2,4,7,3,5,6,8}和中序遍历序列{4,7,2,1,5,3,8,6}重建出如图所示的二叉树。




```
public class BinaryTreeNode {
```

```
    public static int value;
    public BinaryTreeNode leftNode;
    public BinaryTreeNode rightNode;
}

import java.util.Arrays;
public class Problem6{
    public static void main(String[] args) throws Exception {
        int[] preSort={1,2,4,7,3,5,6,8};
        int[] inSort={4,7,2,1,5,3,8,6};
        BinaryTreeNode root=constructCore(preSort,inSort);
    }
    public static BinaryTreeNode constructCore(int[]
preorder,int[] inorder) throws Exception{

        if(preorder==null||inorder==null){
            return null;
        }
        if(preorder.length!=inorder.length){
            throw new Exception("长度不一样，非法的输入");
        }
        BinaryTreeNode root=new BinaryTreeNode();
        for(int i=0;i<inorder.length;i++){
            if(inorder[i]==preorder[0]){
                root.value=inorder[i];
                System.out.println(root.value);
    root.leftNode=constructCore(Arrays.copyOfRange(preorder,1,
i+1), Arrays.copyOfRange(inorder, 0, i));
    root.rightNode=constructCore(Arrays.copyOfRange(preorder,i+
1, preorder.length),Arrays.copyOfRange(inorder, i+1,
inorder.length));
            }
        }
        return root;
    }
}
```

面试题 7：用两个栈实现队列

题目描述：用两个栈实现一个队列，实现对了的两个函数 appendTail 和 deleteHead，分别完成在队列尾插入结点和在队列头部删除结点的功能。

```
public class Problem7<T> {
    private Stack<T> stack1=new Stack<T>();
    private Stack<T> stack2=new Stack<T>();
```

```
    public void appendTail(T t){
        stack1.push(t);
    }
    public T deleteHead() throws Exception{
        if(stack2.isEmpty()){
            while(!stack1.isEmpty()){
                stack2.push(stack1.pop());
            }
        }
        if(stack2.isEmpty()){
            throw new Exception("队列为空，不能删除");
        }
        return stack2.pop();
    }
    public static void main(String args[]) throws Exception
    {
        Problem7<String> p7=new Problem7<>();
        p7.appendTail("1");
        p7.appendTail("2");
        p7.appendTail("3");
        p7.deleteHead();

    }
}
```

面试题 8：旋转数组的最小数字

题目描述：把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。输入一个递增排序的数组的一个旋转，输出旋转数组的最小元素。例如数组{3, 4, 5, 1, 2}为{1, 2, 3, 4, 5}的一个旋转，该数组的最小值为 1.

```
public class Problem8 {
    public static void main(String[] args) {
        Problem8 p8=new Problem8();
        //int[] array={1,1,1,2,0};
    // int[] array={3,4,5,1,2};
        int[] array={1,0,1,1,1};
        System.out.println(p8.findMinNum(array));
    }
    public Integer findMinNum(int[] array){
        if(array==null){
            return null;
        }
        int leftIndex=0;
        int rightIndex=array.length-1;
```

```
        int mid=0;
        while(array[leftIndex]>=array[rightIndex]){
            if(rightIndex-leftIndex<=1){
                mid=rightIndex;
                break;
            }
            mid=(leftIndex+rightIndex)/2;

    if(array[leftIndex]==array[rightIndex]&&array[leftIndex]==a
rray[mid]){
                if(array[leftIndex+1]!=array[rightIndex-1]){

    mid=array[leftIndex+1]<array[rightIndex-1]?(leftIndex+1):(r
ightIndex-1);
                    break;
                }else{
                    leftIndex++;
                    rightIndex--;
                }
            }else{
                if(array[mid]>=array[leftIndex])
                    leftIndex=mid;
                else {
                    if(array[mid]<=array[rightIndex])
                        rightIndex=mid;
            }
          }
        }
        return array[mid];

    }

}
```

面试题 9：斐波那契数列

题目一：写一个函数，输入 n，求斐波那契数列的第 n 项。

```
public class Fibonacci {
    public long fibonacci(int n)
    {
        long result=0;
        long preOne=0;
        long preTwo=1;
        if(n==0)
```

```
        {
            return preOne;
        }
        if(n==1)
        {
            return preTwo;
        }
        for(int i=2;i<=n;i++)
        {
            result=preOne+preTwo;
            preOne=preTwo;
            preTwo=result;

        }
        return result;
    }
}
```

面试题 10：二进制中 1 的个数

题目：请实现一个函数，输入一个整数，输出该数二进制表示中 1 的个数。例如把 9 表示成二进制是 1001，有 2 位是 1，因此如果输入 9，函数输出 2.

```
public class Problem10 {
    public static void main(String args[])
    {
        Problem10 test=new Problem10();
        System.out.println(test.numberOf1(3));

    }
    public int numberOf1(int n)
    {
        int count=0;
        while(n!=0)
        {
            count++;
            n=(n-1) & n;
        }
        return count;
    }
}
```

面试题 11：数值的整数次方

题目:实现函数 double Power(double base,int exponent),求 base 的 exponent 次方。不得使用库函数，同时不需要考虑大数问题。

```
import java.rmi.server.ExportException;
public class Problem11 {
    public static void main(String[] args) throws Exception {
        Problem11 p11=new Problem11();
        System.out.println(p11.power(2.0, 3));
    }
 public double power(double base,int exponent) throws Exception{
     double result=0.0;
     if(equal(base,0.0)&&exponent<0){
         throw new Exception("0的负数次幂没有意义");
     }
     if(exponent<0){
         result=1.0/powerWithExpoment(base,-exponent);
     }else{
         result=powerWithExpoment(base, exponent);
     }
    return result;

 }
private double powerWithExpoment(double base, int exponent) {
    if(exponent==0){
        return 1;
    }
    if(exponent==1){
        return base;
    }
    double result=1.0;
    for(int i=1;i<=exponent;i++){
        result=result*base;
    }
    return result;
}
private boolean equal(double num1, double num2) {
    if((num1-num2>-0.0000001)&&num1-num2<0.0000001)
    {
        return true;
    }else{
        return false;
    }
```

```
}
}
```

面试题 12：打印 1 到最大的 n 位数

题目：输入数字 n，按顺序打印出从 1 到最大的 n 位进制数。比如输入 3，则打印出 1、2、3 一直到 999.

```
public class Problem12 {
    public static void main(String[] args) {
        Problem12 p12=new Problem12();
        p12.printToMaxOfNDigits(2);
    }
    public void printToMaxOfNDigits(int n){
        int[] array=new int[n];
        if(n<=0)
            return;
        printArray(array,0);
    }
    private void printArray(int[] array,int n){
        for(int i=0;i<10;i++){
            if(n!=array.length){
                array[n]=i;
                printArray(array, n+1);
            }else{
                boolean isFirstNo0=false;
                for(int j=0;j<array.length;j++){
                    if(array[j]!=0){
                        System.out.print(array[j]);
                        if(!isFirstNo0)
                            isFirstNo0=true;
                    }else{
                        if(isFirstNo0)
                            System.out.print(array[j]);
                    }
                }
                System.out.println();
                return;
            }
        }
    }

}
```

面试题 13：在 O（1）时间删除链表结点

题目：给定单向链表的头指针和一个结点指针，定义一个函数在 O(1)时间删除该结点。

```
public class Problem13 {
public static void main(String[] args) {
    ListNode head=new ListNode();
    ListNode second=new ListNode();
    ListNode third=new ListNode();
    head.nextNode=second;
    second.nextNode=third;
    head.data=1;
    second.data=2;
    third.data=3;
    Problem13 p13=new Problem13();
    p13.deleteNode(head, second);
    System.out.println(head.nextNode.data);
}
public void deleteNode(ListNode head,ListNode deListNode){
    if(deListNode==null||head==null){
        return;
    }
    if(head==deListNode){//删除头结点
        head=null;
    }else{
        if(deListNode.nextNode==null){//删除尾结点
            ListNode pointListNode=head;
            while(pointListNode.nextNode.nextNode!=null){
                pointListNode=pointListNode.nextNode;
            }
            pointListNode.nextNode=null;
        }else{
            deListNode.data=deListNode.nextNode.data;
            deListNode.nextNode=deListNode.nextNode.nextNode;
        }
    }
}
}
class ListNode
{
    int data;
    ListNode nextNode;
}
```

面试题 14：调整数组顺序使奇数位于偶数前面

题目：输入一个整数数组，实现一个函数来调整该函数数组中数字的顺序，使得所有奇数位于数组的前半部分，所有的数组位于数组的后半部分。

```
public class Problem14 {
    public static void main(String args[])
    {
        int[] array={1,2,3,4,5,6,7};
        Problem14 test=new Problem14();
        test.order(array);
        for(int item:array)
            System.out.println(item);
    }
    public void order(int[] array)
    {
        if(array==null||array.length==0)
            return ;
        int start=0;
        int end=array.length-1;
        while(start<end){
            while(start<end&&!isEven(array[start])){
                start++;
            }
            while(start<end&&isEven(array[end])){
                end--;
            }
            if(start<end){
                int temp=array[start];
                array[start]=array[end];
                array[end]=temp;
            }
        }

    }
    private boolean isEven(int n)
    {
        return n%2==0;
    }
}
```

面试题 15：链表中倒数第 k 个结点

题目：输入一个链表，输出该链表中倒数第 k 个结点。为了符合大多数人的习惯，本题从 1 开始计数，即链表的尾结点是倒数第一个结点。例如一个有 6 个结点的链表，从头结点依次是 1, 2, 3, 4， 5, 6。倒数第三个结点就是值为 4 的结点。

```
public class Problem15 {
    public static void main(String[] args) {
        ListNode head=new ListNode();
        ListNode second=new ListNode();
        ListNode third=new ListNode();
        ListNode forth=new ListNode();
        head.nextNode=second;
        second.nextNode=third;
        third.nextNode=forth;
        head.data=1;
        second.data=2;
        third.data=3;
        forth.data=4;
        Problem15 test=new Problem15();
        ListNode resultListNode=test.findKToTail(head, 3);
        System.out.println(resultListNode.data);
    }
   public ListNode findKToTail(ListNode head,int k){
       if(head==null||k==0){
           return null;
       }
       ListNode resultNode=null;
       ListNode headListNode=head;
       for(int i=0;i<k;++i){
           if(headListNode.nextNode!=null){
               headListNode=headListNode.nextNode;
           }else{
               return null;
           }
       }
           resultNode=head;
           while(headListNode!=null){
               resultNode=resultNode.nextNode;
               headListNode=headListNode.nextNode;
           }
    return resultNode;
   }
}
```

```
public class ListNode
{
    int data;
    ListNode nextNode;
}
```

面试题 16：反转链表

题目：定义一个函数，输入一个链表的头结点，反转该链表并输出反转后链表的头结点。

```
public class Problem16 {
    public static void main(String[] args) {
        ListNode head=new ListNode();
        ListNode second=new ListNode();
        ListNode third=new ListNode();
        ListNode forth=new ListNode();
        head.nextNode=second;
        second.nextNode=third;
        third.nextNode=forth;
        head.data=1;
        second.data=2;
        third.data=3;
        forth.data=4;
        Problem16 test=new Problem16();
        ListNode resultListNode=test.reverseList(head);
        System.out.println(resultListNode.data);
    }
public ListNode reverseList(ListNode head){
    if(head==null){
        return null;
    }
    if(head.nextNode==null){
        return head;
    }
    ListNode preListNode=null;
    ListNode nowListNode=head;
    ListNode reversedHead=null;
    while(nowListNode.nextNode!=null){
        ListNode nextListNode=nowListNode.nextNode;
        if(nextListNode==null)
            reversedHead=nextListNode;
        nowListNode.nextNode=preListNode;
        preListNode=nowListNode;
        nowListNode=nextListNode;
```

```
    }
    return nowListNode;
}
}
```

面试题 17：合并两个排序的链表

题目：输入两个递增排序的链表，合并这两个链表并使新链表中的结点仍然是按照递增排序的。

```
public class Problem17 {
    public static void main(String[] args) {
        ListNode head1=new ListNode();
        ListNode second1=new ListNode();
        ListNode head2=new ListNode();
        ListNode second2=new ListNode();
        ListNode third2=new ListNode();
        head1.nextNode=second1;
        head2.nextNode=second2;
        second2.nextNode=third2;
        head1.data=1;
        second1.data=3;
        head2.data=2;
        second2.data=2;
        third2.data=2;
        Problem17 test=new Problem17();
        ListNode result=test.mergeList(head1, head2);
    System.out.println(result.nextNode.nextNode.nextNode.nextNo
de.data);
    }
public ListNode mergeList(ListNode head1,ListNode head2){
     if(head1==null){
        return head2;
    }else if(head2==null){
        return head1;
    }
    ListNode mergeHead=null;
    if(head1.data<head2.data){
        mergeHead=head1;
        mergeHead.nextNode=mergeList(head1.nextNode,head2);
    }else{
        mergeHead=head2;
        mergeHead.nextNode=mergeList(head1, head2.nextNode);
    }
    return mergeHead;
```

```
}
}
```

面试题 18：树的子结构

题目：输入两颗二叉树 A 和 B，判断 B 是不是 A 的子结构。

```
public class Problem18 {
    public static void main(String args[])
    {
        BinaryTreeNode root1=new BinaryTreeNode();
        BinaryTreeNode node1=new BinaryTreeNode();
        BinaryTreeNode node2=new BinaryTreeNode();
        BinaryTreeNode node3=new BinaryTreeNode();
        BinaryTreeNode node4=new BinaryTreeNode();
        BinaryTreeNode node5=new BinaryTreeNode();
        BinaryTreeNode node6=new BinaryTreeNode();
        root1.leftNode=node1;
        root1.rightNode=node2;
        node1.leftNode=node3;
        node1.rightNode=node4;
        node4.leftNode=node5;
        node4.rightNode=node6;
        root1.value=8;
        node1.value=8;
        node2.value=7;
        node3.value=9;
        node4.value=2;
        node5.value=4;
        node6.value=7;
        BinaryTreeNode root2=new BinaryTreeNode();
        BinaryTreeNode a=new BinaryTreeNode();
        BinaryTreeNode b=new BinaryTreeNode();
        root2.leftNode=a;
        root2.rightNode=b;
        root2.value=8;
        a.value=9;
        b.value=2;
        Problem18 test=new Problem18();
        System.out.println(test.hasSubTree(root1, root2));
    }
  public boolean hasSubTree(BinaryTreeNode root1,BinaryTreeNode
root2){
      boolean result=false;
      if(root1!=null&&root2!=null){
```

```
        if(root1.value==root2.value){
            result=doesTree1HavaTree2(root1,root2);
            if(!result)
                result=hasSubTree(root1.leftNode, root2);
             if(!result)
                result=hasSubTree(root1.rightNode, root2);
        }
       }
      return result;
   }
private boolean doesTree1HavaTree2(BinaryTreeNode root1,
BinaryTreeNode root2) {
    if(root2==null){
        return true;
    }else if(root1==null)
        return false;
    if(root1.value!=root2.value){
        return false;
    }
    return doesTree1HavaTree2(root1.leftNode, root2.leftNode)&&
           doesTree1HavaTree2(root1.rightNode, root2.rightNode);
}
}
```

面试题 19：二叉树的镜像

题目：请完成一个函数，输入一个二叉树，该函数输出它的镜像。

```
public class Problem19 {
public static void main(String[] args) {
    BinaryTreeNode root1=new BinaryTreeNode();
    BinaryTreeNode node1=new BinaryTreeNode();
    BinaryTreeNode node2=new BinaryTreeNode();
    BinaryTreeNode node3=new BinaryTreeNode();
    BinaryTreeNode node4=new BinaryTreeNode();
    BinaryTreeNode node5=new BinaryTreeNode();
    BinaryTreeNode node6=new BinaryTreeNode();
    root1.leftNode=node1;
    root1.rightNode=node2;
    node1.leftNode=node3;
    node1.rightNode=node4;
    node4.leftNode=node5;
    node4.rightNode=node6;
    root1.value=8;
    node1.value=8;
```

```
    node2.value=7;
    node3.value=9;
    node4.value=2;
    node5.value=4;
    node6.value=7;
    Problem19 test=new Problem19();
    BinaryTreeNode
rootBinaryTreeNode=test.mirrorBinaryTree(root1);
    System.out.println(root1.rightNode.value);
}
public BinaryTreeNode mirrorBinaryTree(BinaryTreeNode root){
        if(root==null){
            return null;
        }
        if(root.leftNode==null&&root.rightNode==null)
            return null;
        Stack<BinaryTreeNode> stack=new Stack<BinaryTreeNode>();
        while(root!=null||!stack.isEmpty()){
            while(root!=null){
                BinaryTreeNode temp=root.leftNode;
                root.leftNode=root.rightNode;
                root.rightNode=temp;
                stack.push(root);
                root=root.leftNode;
            }
            root=stack.pop();
            root=root.rightNode;
        }
        return root;
    }
}
```

面试题 20：顺时针打印矩阵

题目：输入一个矩阵，按照从外向里以顺时针的顺序依次打印出每一个数字。

```
public class Problem20 {
    public static void main(String[] args) {
        int[][] array={
                {1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
        Problem20 testCircle=new Problem20();
        testCircle.printMatixClockwisely(array);
    }
    public void printMatixClockwisely(int[][] array){
        if(array==null)
```

```
            return;
        int start=0;
        while(array[0].length>start*2&&array.length>start*2){
            printOneCircle(array,start);
            start++;
        }
    }
    private void printOneCircle(int[][] array, int start) {
        for(int i=start;i<array[0].length-start;i++){
            System.out.print(array[start][i]+" ");
        }
        if(array.length-1-start>start){
            for(int i=start+1;i<array.length-start-1;i++){

    System.out.print(array[i][array[0].length-1-start]+" ");
            }
        }
         if(array[0].length-start-1>start &&
array.length-start-1>start)
        {
            for(int i=array.length-start-1;i>start;i--)
            {
           System.out.print(array[array.length-start-1][i]+" ");
            }
        }
        if(array.length-1-start>start &&
array[0].length-1-start>start)
        {
            for(int i=array.length-start-1;i>start;i--)
            {
                System.out.print(array[i][start]+" ");
            }
        }

    }
}
```

面试题 21：包含 min 函数的栈

题目：定义栈的数据结构，请在该类型中实现一个能够得到栈的最小元素的min函数。在该栈中，调用min、push及pop德尔时间复杂度都是O(1)

```
public class Problem21 extends MyStack<Integer>{
    public static void main(String[] args) {
        Problem21 test=new Problem21();
```

```
        test.push(3);
        test.push(2);
        test.push(1);
        test.push(4);
        test.push(5);
        System.out.println(test.min());
    }
    private MyStack<Integer> minStack=new MyStack<Integer>();
    private MyStack<Integer> dataStack=new MyStack<Integer>();
    public void push(Integer item){
        dataStack.push(item);
        if(minStack.length==0||item<=minStack.head.data){
            minStack.push(item);
        }else{
            minStack.push(minStack.head.data);
        }
    }
    public Integer pop(){
        if(dataStack.length==0||minStack.length==0){
            return null;
        }
        minStack.pop();
        return dataStack.pop();
    }
    public Integer min(){
        if(minStack.length==0)
            return null;
        return minStack.head.data;
    }
}
```

面试题 22：栈的压入、弹出序列

题目：输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出序列。假设压入栈的所有数字均不相等。例如压栈序列为 1、2、3、4、5. 序列 4、5、3、2、1 是压栈序列对应的一个弹出序列，但 4、3、5、1、2 却不是。

```
public class Problem22 {
    public static void main(String[] args) {
        int[] array1={1,2,3,4,5};
        int[] array2={4,3,5,2,1};
        Problem22 test=new Problem22();
        System.out.println(test.isPopOrder(array1, array2));
    }
```

```
    public boolean isPopOrder(int[] line1,int[] line2){
        if(line1==null||line2==null){
            return false;
        }
        int point1=0;
        Stack<Integer> stack=new Stack<Integer>();
        for(int i=0;i<line2.length;i++){
            if(!stack.isEmpty()&&stack.peek()==line2[i]){
                stack.pop();
            }else{
                if(point1==line1.length){
                    return false;
                }else{
                    do{
                        stack.push(line1[point1++]);
    }while(stack.peek()!=line2[i]&&point1!=line1.length);
                if(stack.peek()==line2[i])
                    stack.pop();
                else
                    return false;
                }

            }
        }
        return true;
    }
}
```

面试题 23：从上往下打印二叉树

题目：从上往下打印二叉树的每个结点，同一层的结点按照从左到右的顺序打印。

```
import java.util.LinkedList;
import java.util.Queue;
import com.utils.BinaryTreeNode;
public class Problem23  {
    public static void main(String args[])
    {
        BinaryTreeNode root1=new BinaryTreeNode();
        BinaryTreeNode node1=new BinaryTreeNode();
        BinaryTreeNode node2=new BinaryTreeNode();
        BinaryTreeNode node3=new BinaryTreeNode();
        BinaryTreeNode node4=new BinaryTreeNode();
        BinaryTreeNode node5=new BinaryTreeNode();
        BinaryTreeNode node6=new BinaryTreeNode();
```

```
        root1.leftNode=node1;
        root1.rightNode=node2;
        node1.leftNode=node3;
        node1.rightNode=node4;
        node4.leftNode=node5;
        node4.rightNode=node6;
        root1.value=8;
        node1.value=8;
        node2.value=7;
        node3.value=9;
        node4.value=2;
        node5.value=4;
        node6.value=7;
        Problem23 test=new Problem23();
        test.printFromTopToBottom(root1);
    }
public void printFromTopToBottom(BinaryTreeNode root){
    if(root==null)
        return;
    Queue<BinaryTreeNode> queue=new
LinkedList<BinaryTreeNode>();
    queue.add(root);
    while(!queue.isEmpty()){
        BinaryTreeNode node=queue.poll();
        System.out.print(node.value);
        if(node.leftNode!=null){
            queue.add(node.leftNode);
        }
        if(node.rightNode!=null){
            queue.add(node.rightNode);
        }
    }
}
}
```

面试题 24：二叉搜索树的后序遍历序列

题目：输入一个整数数组，判断该数组是不是某二叉搜索树的后序遍历的结果。是则返回true，否则返回false。

```
public class Problem24 {
    public static void main(String[] args) {
        int[] array={5,7,6,9,11,10,8};
        //int[] array={7,4,6,5};
        //int[] array={6,7,8,5};
```

```
        Problem24 p24=new Problem24();
        System.out.println(p24.verfiySequenceOfBST(array));
    }
    public boolean verfiySequenceOfBST(int[] sequence){
        if(sequence==null||sequence.length==0)
            return false;
        int length=sequence.length;
        int root=sequence[length-1];
        int cut=0;
        for(int i=0;i<length-1;i++){
            if(sequence[i]>root)
                cut=i+1;
                break;
        }
        if(cut==0){
            verfiySequenceOfBST(Arrays.copyOfRange(sequence, 0,
length-1));
        }else{
        for(int j=cut;j<length-1;j++){
            if(sequence[j]<root)
                return false;
        }
        }
         boolean left=true;
         if(cut>0)
           left= verfiySequenceOfBST(Arrays.copyOfRange(sequence,
0, cut));
         boolean right=true;
         if(cut<length-1)
             right=
verfiySequenceOfBST(Arrays.copyOfRange(sequence,cut,length-1);
        return (right&&left);
    }
}
```

面试题 25：二叉树中和为某一值的路径

题目：输入一颗二叉树和一个整数，打印出二叉树中结点值的和为输入整数的所有路径。从树的根节点开始往下一直到叶结点所经过的结点形成一条路径。

```
public class Problem25 {
    public static void main(String args[])
    {
        BinaryTreeNode root1=new BinaryTreeNode();
        BinaryTreeNode node1=new BinaryTreeNode();
```

```
        BinaryTreeNode node2=new BinaryTreeNode();
        BinaryTreeNode node3=new BinaryTreeNode();
        BinaryTreeNode node4=new BinaryTreeNode();
        root1.leftNode=node1;
        root1.rightNode=node2;
        node1.leftNode=node3;
        node1.rightNode=node4;
        root1.value=10;
        node1.value=5;
        node2.value=12;
        node3.value=4;
        node4.value=7;
        Problem25 testFindPath=new Problem25();
        testFindPath.findPath(root1, 22);
    }
    public void findPath(BinaryTreeNode root,int sum){
        if(root==null)
            return;
        Stack<Integer> stack=new Stack<Integer>();
        int currentSum=0;
        findPath(root,sum,stack,currentSum);
    }
    private void findPath(BinaryTreeNode root, int sum, Stack<Integer>
stack,int currentSum) {
        currentSum+=root.value;
        stack.push(root.value);
        if(root.leftNode==null&&root.rightNode==null){
            if(currentSum==sum){
                System.out.println("找到一个路径");
                for(int path:stack){
                    System.out.print(path+" ");
                }
                System.out.println();
            }
        }
        if(root.leftNode!=null){
            findPath(root.leftNode, sum, stack, currentSum);
        }
        if(root.rightNode!=null){
            findPath(root.rightNode, sum, stack, currentSum);
        }
        stack.pop();
    }
}
```

面试题 26：复杂链表的复制

题目：实现函数复制一个复杂链表。在复杂链表中，每个结点除了有一个 next 指针指向下一个结点外，还有一个指向链表中任意结点或 null。

```
package com.example.offer26_40;
public class Problem26 {
    public static void main(String[] args) {
        Problem26 testClone=new Problem26();
        ComplexListNode root=new ComplexListNode();
        ComplexListNode node1=new ComplexListNode();
        ComplexListNode node2=new ComplexListNode();
        ComplexListNode node3=new ComplexListNode();
        ComplexListNode node4=new ComplexListNode();
        root.data=1;
        node1.next=node2;
        node2.next=node3;
        node3.next=node4;
        root.data=1;
        node1.data=2;
        node2.data=3;
        node3.data=4;
        node4.data=5;
        root.sibling=node1;
        node1.sibling=root;
        node3.sibling=node1;
        ComplexListNode result=testClone.clone(root);
        System.out.println(result.data);
    }
    public ComplexListNode clone(ComplexListNode head){
        cloneNodes(head);
        connectSiblingNodes(head);
        return reconnectNodes(head);
    }
    public void cloneNodes(ComplexListNode head){
        ComplexListNode node=head;
        while(node!=null){
            ComplexListNode cloneNode=new ComplexListNode();
            cloneNode.data=node.data;
            cloneNode.next=node.next;
            cloneNode.sibling=null;
            node.next=cloneNode;
            node=cloneNode.next;
        }
```

```
    }
    public void connectSiblingNodes(ComplexListNode head){
        ComplexListNode node=head;
        while(node!=null){
            ComplexListNode clonedNode=node.next;
            if(node.sibling!=null){
                clonedNode.sibling=node.sibling.next;
            }
            node=clonedNode.next;
        }
    }
    public ComplexListNode reconnectNodes(ComplexListNode head){
        ComplexListNode node=head;
        ComplexListNode clonedHead=null;
        ComplexListNode clonedNode=null;
        if(node!=null){
            clonedNode=node.next;
            clonedHead=clonedNode;
            node.next=clonedNode.next;
            node=node.next;
        }
        while(node!=null){
            clonedNode.next=node.next;
            clonedNode=clonedHead.next;
            node.next=clonedNode.next;
            node=node.next;
        }
        return clonedHead;
    }
}
class ComplexListNode
{
    int data;
    ComplexListNode next;
    ComplexListNode sibling;
}
```

面试题 27：二叉搜索树与双向链表

题目：输入一棵二叉搜索树，将该二叉搜索树转换成一个排序的双向链表。要求不能创建任何新的结点，只能调整树中结点指针的指向。

```
public class Problem27 {
    public BinaryTreeNode convert(BinaryTreeNode root){
        BinaryTreeNode lastNodeList=null;
```

```
        convertNode(root,lastNodeList);
        while(lastNodeList!=null&&lastNodeList.leftNode!=null){
            lastNodeList=lastNodeList.leftNode;
        }
        return lastNodeList;
    }

    private void convertNode(BinaryTreeNode root, BinaryTreeNode
lastNodeList) {
      if(root==null)
         return;
      BinaryTreeNode cuurent=root;
      if(cuurent.leftNode!=null){
         convertNode(cuurent.leftNode, lastNodeList);
      }
      cuurent.leftNode=lastNodeList;
      if(lastNodeList!=null)
         lastNodeList.rightNode=cuurent;
      lastNodeList=cuurent;
      if(cuurent.rightNode!=null){
         convertNode(cuurent.rightNode, lastNodeList);
      }
    }
}
```

面试题 28：字符串的排列

题目：输入一个字符串，打印出该字符串中字符的所有排列。

```
public class Problem28 {
    public static void main(String args[])
    {
        Problem28 testPermutation=new Problem28();
        testPermutation.permutation("abcd");
    }
    public void permutation(String str){
        int count=0;
        if(str==null)
            return;
        char[] chs=str.toCharArray();
        int point=0;
        System.out.print(chs);
        System.out.print(" ");
        count++;
        char temp1=chs[point];
```

```
            chs[point]=chs[++point];
            chs[point]=temp1;
            while(!String.valueOf(chs).equals(str)){
                System.out.print(chs);
                System.out.print(" ");
                count++;
                if(point==chs.length-1){
                    char temp=chs[point];
                    chs[point]=chs[0];
                    chs[0]=temp;
                    point=0;
                }else{
                    char temp=chs[point];
                    chs[point]=chs[++point];
                    chs[point]=temp;

                }
            }
            System.out.println(count);
        }
    }
```

面试题 29：数组中出现次数超过一半的数组

题目：数组中有一个数字出现次数超过数组长度的一半，请找出这个数字。例如输入一个长度为9的数组{1,2,3,2,2,2,5,4,2}。2出现的次数超过数组长度的一半，因此输出2.

```
public class Problem29 {
    public static void main(String[] args) {
        int[] array={1,2,3,2,2,2,5,4,2};
        Problem29 p=new Problem29();
        System.out.println(p.moreThanHalfNum(array));
    }
    public Integer moreThanHalfNum(int[] array){
        if(array==null)
            return null;
        int count=0;
        Integer resultInteger=null;
        for(int i=0;i<array.length;i++){
            if(count==0){
                resultInteger=array[i];
                count=1;
            }else if(array[i]==resultInteger)
                count++;
```

```
            else
                count--;
        }
        if(checkMoreThanHalf(array,resultInteger))
            return resultInteger;
        else
            return null;
    }

    private boolean checkMoreThanHalf(int[] array, Integer number)
{
        int times=0;
        for(int i=0;i<array.length;i++)
        {
            if(array[i]==number)
                times++;
        }
        if(times*2<=array.length)
            return false;
        else
            return true;
    }
}
```

面试题 30：最小的 k 个数。

题目：输入 n 个整数，找出其中最小的 k 个数。例如输入 4, 5, 1， 6, 2, 7, 3, 8 这 8 个数字，则最少的 4 个数字是 1, 2, 3, 4.

```
public class Problem30 {
    public static void main(String[] args) {
        Problem30 test=new Problem30();
        int[] array={4,5,1,6,2,7,3,8};
        test.getLeastNumbers(array,2);
    }
    public void getLeastNumbers(int[] array,int k){
        if(array==null||k<0||k>array.length)
            return;
        int[] kArray=Arrays.copyOfRange(array, 0, k);
        buildMaxHeap(kArray);
        for(int i=k;i<array.length;i++){
            if(array[i]<kArray[0]){
                kArray[0]=array[i];
                maxHeap(kArray,0);
            }
        }
        for(int i:kArray)
```

```
            System.out.print(i);
    }
    private void maxHeap(int[] kArray, int i) {
        int left=2*i;
        int right=left+1;
        int largest=0;
        if(right<kArray.length&&kArray[left]>kArray[i])
            largest=left;
        else
            largest=i;
        if(right<kArray.length&&kArray[right]>kArray[largest])
            largest=right;
        if(largest!=i){
            int temp=kArray[i];
            kArray[i]=kArray[largest];
            kArray[largest]=temp;
            maxHeap(kArray, largest);
        }
    }
    private void buildMaxHeap(int[] kArray) {
        for(int i=kArray.length/2;i>=0;i--)
            maxHeap(kArray, i);
    }

}
```

面试题 31：连续子数组的最大和

题目：输入一个整型数组，数组里有正数也有负数。数组中一个或连续的多个整数组成一个子数组。求所有子数组的和的最大值。要求时间复杂度为 O(n)。例如输入的数组为{1，-2, 3, 10，-4, 7, 2，-5}，和最大的子数组为{3, 10，-4, 7, 2}。

```
public class Problem31 {
    public static void main(String[] args) {
        Problem31 p=new Problem31();
        int[] array={1,-2,3,10,-4,7,2,-5};
        System.out.println(p.findGreatestSubArray(array));
    }
    public int findGreatestSubArray(int[] array){
        if(array==null)
            return 0;
        int currentSum=0;
        int greatestSum=0;
        for(int i=0;i<array.length;i++){
            if(currentSum<=0){
```

```
                currentSum=array[i];
            }else{
                currentSum+=array[i];
            }
            if(currentSum>greatestSum)
                greatestSum=currentSum;
        }
        return greatestSum;
    }
}
```

面试题 32：从 1 到 n 整数中 1 出现的次数

题目：输入一个整数 n，求从 1 到 n 这 n 个整数的十进制表示中 1 出现的次数。例如输入 12，这些整数中包含 1 的数字有 1, 10, 11, 12，1 一共出现了 5 次。

解题思路：解法二告诉我们 1~ N 中"1"的个数跟最高位有关，那我们换个角度思考，给定一个 N，我们分析 1~N 中的数在每一位上出现 1 的次数的和，看看每一位上"1"出现的个数的和由什么决定。

1 位数的情况：在解法二中已经分析过，大于等于 1 的时候，有 1 个，小于 1 就没有。

2 位数的情况：N=13,个位数出现的 1 的次数为 2，分别为 1 和 11，十位数出现 1 的次数为 4，分别为 10, 11, 12, 13，所以 f(N) = 2+4。N=23,个位数出现的 1 的次数为 3,分别为 1,11,21,十位数出现 1 的次数为 10,分别为 10~19,f(N)=3+10。由此我们发现，个位数出现 1 的次数不仅和个位数有关，和十位数也有关，如果个位数大于等于 1，则个位数出现 1 的次数为十位数的数字加 1；如果个位数为 0，个位数出现 1 的次数等于十位数数字。而十位数上出现 1 的次数也不仅和十位数相关，也和个位数相关：如果十位数字等于 1，则十位数上出现 1 的次数为个位数的数字加 1，假如十位数大于 1，则十位数上出现 1 的次数为 10。

3 位数的情况：

N=123，个位出现 1 的个数为 13:1, 11, 21，…，91, 101, 111, 121。十位出现 1 的个数为 20:10~19, 110~119。百位出现 1 的个数为 24:100~123。

我们可以继续分析 4 位数，5 位数，推导出下面一般情况： 假设 N，我们要计算百位上出现 1 的次数，将由三部分决定：百位上的数字，百位以上的数字，百位一下的数字。

如果百位上的数字为 0，则百位上出现 1 的次数仅由更高位决定，比如 12013，百位出现 1 的情况为 100~199, 1100~1199, 2100~2199，…，11100~11199，共 1200 个。等于更高位数字乘以当前位数，即 12 * 100。

如果百位上的数字大于 1,则百位上出现 1 的次数仅由更高位决定,比如 12213，百位出现 1 的情况为 100~199, 1100~1199, 2100~2199，…，11100~11199，12100~12199 共 1300 个。等于更高位数字加 1 乘以当前位数，即(12 + 1)*100。

如果百位上的数字为 1，则百位上出现 1 的次数不仅受更高位影响，还受低位影响。例如 12113，受高位影响出现 1 的情况：100~199, 1100~1199, 2100~2199，…，11100~11199，共 1200 个，但它还受低位影响，出现 1 的情况是 12100~12113，共 114 个，等于低位数字 113+1。

```
public class Problem32 {
    public static void main(String[] args) {
        Problem32 p=new Problem32();
        System.out.println(p.countOne(123));
    }
    public long countOne(long n)
    {
        long count = 0;
        long i = 1;
        long current = 0,after = 0,before = 0;
        while((n / i) != 0)
        {
            current = (n / i) % 10; //当前位数字
            before = n / (i * 10); //高位数字
            after = n - (n / i) * i; //低位数字
            if (current > 1)
                count = count + (before + 1) * i;
            else if (current == 0)
                count = count + before * i;
            else if(current == 1)
                count = count + before * i + after + 1;
            i = i * 10;
        }
        return count;
    }
}
```

面试题 33：把数组排成最小的数

题目：输入一个正整数数组，把数组里所有数字拼接起来排成一个数，打印能拼接出的所有数字的最小的一个。例如输入{3, 32, 321}，则打印最小的数字是321323.

```
public class Problem33 {
    public static void main(String[] args) {
        Problem33 test=new Problem33();
        int[] array={3,32,321};
        test.printMin(array);
    }
public void printMin(int[] array){
    int[] clone=array.clone();
    printMinNumber(clone,0,clone.length-1);
    for(int i:clone)
        System.out.print(i);
}
```

```
private void printMinNumber(int[] array, int start, int end) {
    if(start<end){
        int main_number=array[end];
        int small_cur=start;
        for(int j=start;j<end;j++){
    if(isSmall(String.valueOf(array[j]),String.valueOf(main_num
ber))){
                int temp=array[j];
                array[j]=array[small_cur];
                array[small_cur]=temp;
                small_cur++;
            }
        }
        array[end]=array[small_cur];
        array[small_cur]=main_number;
        printMinNumber(array, 0, small_cur-1);
        printMinNumber(array, small_cur+1, end);
    }
}
private boolean isSmall(String m, String n) {
    String left=m+n;
    String right=n+m;
    boolean result=false;
    for(int i=0;i<left.length();i++)
    {
        if(left.charAt(i)<right.charAt(i))
            return true;
        else
            if(left.charAt(i)>right.charAt(i))
                return false;
    }
    return result;
}
}
```

面试题 34：丑数

题目：我们把只包含因子 2, 3, 和 5 的称为丑数。求按从小到大的顺序的第 1500 个丑数。例如 6、8 都是丑数，但 14 不是，因为它包含因子 7. 习惯上我们把 1 当做第一个丑数。

```
public class Problem34 {
    public static void main(String[] args) {
        Problem34 p=new Problem34();
        System.out.println(p.getUglyNumber(1500));
```

```
    }
    public int getUglyNumber(int n){
        if(n<0)
            return 0;
        int[] uglyArray=new int[n];
        uglyArray[0]=1;
        int multiply2=1;
        int multiply3=1;
        int multiply5=1;
        for(int i=1;i<uglyArray.length;i++){
            int min=min(multiply2*2,multiply3*3,multiply5*5);
            uglyArray[i]=min;
            while(multiply2*2<=min)
                multiply2++;
            while(multiply3*3<=min)
                multiply3++;
            while(multiply5*5<=min)
                multiply5++;
        }
        return uglyArray[n-1];
    }
    private int min(int i, int j, int k) {
        int min=(i<j)?i:j;
        return (min<k)?min:k;
    }
}
```

面试题 35：第一个只出现一次的字符

题目：在字符串中找出第一个只出现一次的字符。如果输入“abaccdeff”，则输出‘b’。

```
public class Problem35 {
    public static void main(String[] args) {
        Problem35 p=new Problem35();
        System.out.println(p.firstNotRepeatChar("agbaccdeff"));
    }
    public Character firstNotRepeatChar(String str){
        if(str==null)
            return null;
        char[] strChar=str.toCharArray();
        LinkedHashMap<Character, Integer> hash=new
LinkedHashMap<Character, Integer>();
        for(char item:strChar){
            if(hash.containsKey(item))
```

```
                hash.put(item, hash.get(item)+1);
            else
                hash.put(item, 1);
        }
        for(char key:hash.keySet()){
            if(hash.get(key)==1)
                return key;
        }
        return null;
    }
}
```

面试题 36：数组中的逆序对

题目：在数组中的两个数字如果前一个数字大于后一个数字，则这两个数字组成一个逆序对。输入一个数组，求出这个数组的逆序对的总数。例如在数组{7, 5, 6, 4}中，一共存在 5 个逆序对，分别是（7, 6）、（7、5），（7、4），（6、4），（5、4）。

```
public class Problem36 {
    public static void main(String[] args) {
        Problem36 p=new Problem36();
        int[] array={7,5,6,4};
        System.out.println(p.inversePairs(array));
    }
    public int inversePairs(int[] array){
        if(array==null)
            return 0;
        int[] copy=array.clone();
        return mergeSort(array,copy,0,array.length-1);
    }
    private int mergeSort(int[] array, int[] result, int start, int
end) {
        if(start==end){
            result[start]=array[start];
            return 0;
        }
        int length=(end-start)/2;
        int left=mergeSort(result, array, start, start+length);
        int right=mergeSort(result, array, start+length+1, end);
        int leftIndex=start+length;
        int rightIndex=end;
        int count=0;
        int point=rightIndex;
        while(leftIndex>=start&&rightIndex>=start+length+1){
```

```
            if(array[leftIndex]>array[rightIndex]){
                result[point--]=array[leftIndex--];
                count+=rightIndex-start-length;
            }else{
                result[point--]=array[rightIndex--];
            }
        }
        for(int i=leftIndex;i>=start;i--)
            result[point--]=array[i];
        for(int j=rightIndex;j>=start+length+1;j--)
            result[point--]=array[j];
        return left+right+count;
    }
}
```

面试题 37：两个链表的第一个公共结点

题目：输入两个链表，找出它们的第一个公共结点。

```
import com.utils.ListNode;
public class Problem37 {
    public static void main(String[] args) {
        ListNode head1=new ListNode();
        ListNode second1=new ListNode();
        ListNode third1=new ListNode();
        ListNode forth1=new ListNode();
        ListNode fifth1=new ListNode();
        ListNode head2=new ListNode();
        ListNode second2=new ListNode();
        ListNode third2=new ListNode();
        ListNode forth2=new ListNode();
        head1.nextNode=second1;
        second1.nextNode=third1;
        third1.nextNode=forth1;
        forth1.nextNode=fifth1;
        head2.nextNode=second2;
        second2.nextNode=forth1;
        third2.nextNode=fifth1;
        head1.data=1;
        second1.data=2;
        third1.data=3;
        forth1.data=6;
        fifth1.data=7;
        head2.data=4;
        second2.data=5;
```

```
        third2.data=6;
        forth2.data=7;
        Problem37 test=new Problem37();
        System.out.println(test.findFirstCommonNode(head1,
head2).data);
    }
    public ListNode findFirstCommonNode(ListNode head1,ListNode
head2){
        int len1=getListLength(head1);
        int len2=getListLength(head2);
        ListNode longListNode=null;
        ListNode shortListNode=null;
        int dif=0;
        if(len1>len2){
            longListNode=head1;
            shortListNode=head2;
            dif=len1-len2;
        }else{
            longListNode=head2;
            shortListNode=head1;
            dif=len2-len1;
        }
        for(int i=0;i<dif;i++){
            longListNode=longListNode.nextNode;
        }
        while(longListNode!=null&&shortListNode!=null
                &&longListNode!=shortListNode){
            longListNode=longListNode.nextNode;
            shortListNode=shortListNode.nextNode;
        }
        return longListNode;
    }
    private int getListLength(ListNode head1) {
        int result=0;
        if(head1==null)
            return result;
        ListNode point=head1;
        while(point!=null){
            point=point.nextNode;
            result++;
        }
        return result;
    }
}
```

面试题 38：数字在排序数组中出现的次数

题目：统计一个数字在排序数组中出现的次数。例如输入排序数组{1, 2, 3, 3, 3, 3, 4, 5}和数字 3，由于 3 在这个数组中出现了 4 次，因此输出 4。

```
public class Problem38 {
    public static void main(String[] args) {
        Problem38 p=new Problem38();
        int[] array={1,2,3,3,3,3,4,5};
        System.out.println(p.getNumberOfK(array, 3));
    }
    private int getNumberOfK(int[] array, int k) {
        int number=0;
        if(array!=null){
        int first=getFirstK(array,k,0,array.length-1);
        int last=getLastK(array,k, 0, array.length-1);
        if(first>-1&&last>-1)
            number=last-first+1;
        }
        return number;
    }
     private int getFirstK(int[] array, int k,int start, int end)
{
        if(start>end)
            return -1;
        int middleIndex=(start+end)/2;
        int middleData=array[middleIndex];
        if(middleData==k){
if((middleIndex>0&&array[middleIndex-1]!=k)||middleIndex==0)
                return middleIndex;
            else
                end=middleIndex-1;
        }else if(middleData>k)
            end=middleIndex-1;
        else
            start=middleIndex+1;
        return getFirstK(array, k, start, end);
    }
    private int getLastK(int[] array,int k, int start, int end) {
        if(start>end)
            return -1;
        int middleIndex=(start+end)/2;
        int middleData=array[middleIndex];
        if(middleData==k){
```

```
    if((middleIndex<array.length-1&&array[middleIndex+1]!=k)||m
iddleIndex==array.length-1)
                return middleIndex;
            else
                start=middleIndex+1;
        }else if(middleData<k)
            start=middleIndex+1;
        else
            end=middleIndex-1;
        return getLastK(array, k, start, end);
    }
}
```

面试题 39：二叉树的深度

题目一：输入一棵二叉树的根结点，求该树的深度。从根结点到叶结点依次经过的结点（含根、叶结点）形成树的一条路径，最长路径的长度为树的深度。

```
public class Problem39 {
    public static void main(String[] args) {
        BinaryTreeNode root=new BinaryTreeNode();
        BinaryTreeNode node1=new BinaryTreeNode();
        BinaryTreeNode node2=new BinaryTreeNode();
        BinaryTreeNode node3=new BinaryTreeNode();
        BinaryTreeNode node4=new BinaryTreeNode();
        BinaryTreeNode node5=new BinaryTreeNode();
        BinaryTreeNode node6=new BinaryTreeNode();
        root.leftNode=node1;
        root.rightNode=node2;
        node1.leftNode=node3;
        node1.rightNode=node4;
        node2.rightNode=node5;
        node4.leftNode=node6;
        root.value=1;
        node1.value=2;
        node2.value=3;
        node3.value=4;
        node4.value=5;
        node5.value=6;
        node6.value=7;
        Problem39 p=new Problem39();
        System.out.println(p.treeDepth(root));
    }
    public int treeDepth(BinaryTreeNode root){
        if(root==null)
```

```
            return 0;
        int left=treeDepth(root.leftNode);
        int right=treeDepth(root.rightNode);
        return (left>right)?left+1:right+1;
    }
}
```

题目二：输入一棵二叉树的根结点，判断该树是不是平衡二叉树。如果某二叉树中任意结点的左右子树的深度相差不超过1，那么他就是一棵平衡二叉树。

```
public boolean isBalanced(BinaryTreeNode root){
        int depth=0;
        return isBalanced(root,depth);
    }
    private boolean isBalanced(BinaryTreeNode root, int depth) {
        if(root==null){
            depth=0;
            return true;
        }
        int left=0,right=0;
    if(isBalanced(root.leftNode,left)&&isBalanced(root.rightNod
e, right)){
            int diff=left-right;
            if(diff<=1&&diff>=-1){
                depth=1+(left>right?left:right);
                return true;
            }
        }
            return false;
    }
```

测试用例跟题目 1 相同。

面试题 40：数组中只出现一次的数字。

题目：一个整型数组里除了两个数字之外，其他的数字都出现了两次。请写程序找出这两个只出现一次的数字。要求时间复杂度是 O(n)，空间复杂度为 O(1)；

```
public class Problem40 {
    public static void main(String[] args) {
        int[] array={2,4,3,6,3,2,5,5};
        Problem40 p=new Problem40();
        p.findNumsAppearOnce(array);
    }
    public void findNumsAppearOnce(int[] array){
        if(array==null)
            return;
        int number=0;
```

```
        for(int i:array)
            number^=i;
        int index=findFirstBitIs1(number);
        int number1=0;
        int number2=0;
        for(int i:array){
            if(isBit1(i,index))
                number1^=i;
            else
                number2^=i;
        }
            System.out.println(number1);
        System.out.println(number2);
    }
private int findFirstBitIs1(int number) {
        int indexBit=0;
        while((number&1)==0){
            number=number>>1;
            ++indexBit;
        }
        return indexBit;
    }
    private boolean isBit1(int number, int index) {
        number=number>>index;
        return (number&1)==0;
    }
}
```

面试题 41：和为 s 的两个数字 VS 和为 s 的连续正数序列

题目一：输一个递增排序的数组和一个数字 s, 在数组中查找两个数使得它们的和正好是 s。如果有多对数字的和等于 s，输出任意一对即可。例如：输入数组 {1, 2, 4, 7, 11, 15} 和数字为 15. 输出 4 和 11.

```
public class Problem41 {
    public static void main(String[] args) {
        Problem41 p=new Problem41();
        int[] data={1,2,4,7,11,15};
        int sum=15;
        System.out.println(p.findNumberWithSum(data, sum));
    }
    public boolean findNumberWithSum(int[] data,int sum){
        boolean found=false;
        if(data==null)
            return found;
```

```
        int num1=0;
        int num2=0;
        int start=0;
        int end=data.length-1;
        while(start<end){
            int curSum=data[start]+data[end];
            if(curSum==sum){
                num1=data[start];
                num2=data[end];
                found=true;
                break;
            }else if(curSum>sum)
                end--;
            else
                start++;
        }
        System.out.println(num1);
        System.out.println(num2);
        return found;
    }
}
```

题目二：输入一个正数 s，打印出所有和为 s 的连续正数序列（至少含两个数）。例如输入 15，由于 1+2+3+4+5=4+5+6=7+8=15，所以结果打印出 3 个连续序列 1-5、4-6、和 7-8.

```
public void findContinuesSequence(int sum){
        if(sum<3)
            return;
        int small=1;
        int big=2;
        int middle=(1+sum)/2;
        int curSum=small+big;
        while(small<middle){
            if(curSum==sum){
                printContineNum(small,big);
            }
            while(curSum>sum&&small<middle){
                curSum-=small;
                small++;
                if(curSum==sum)
                    printContineNum(small, big);
            }
            big++;
            curSum+=big;
        }
```

```
    }
    private void printContineNum(int small, int big) {
        for(int i=small;i<=big;i++){
            System.out.print(i+" ");
        }
        System.out.println();
    }
```

面试题 42：翻转单词顺序 VS 左旋转字符串。

题目一:输入一个英文句子,翻转句子中单词的顺序,但单词内字符的顺序不变。为简单起见,标点符号和普通字母一样处理。例如输入字符串“I am a student.”,则输出“student. a am I”.

```
public class Problem42 {
    public static void main(String[] args) {
        Problem42 p=new Problem42();
        String string="I am a student.";
        p.reverseSentence(string);
    }
    public void reverseSentence(String sentence){
        if(sentence==null)
            return;
        String[] str=sentence.split(" ");
        StringBuffer sb=new StringBuffer();
        for(int i=str.length-1;i>=0;i--){
            sb.append(str[i]+" ");
        }
        System.out.println(sb);
    }
}
```

题目二:字符串的左旋转操作是把字符串前面的若干个字符转移到字符串的尾部。请定义一个函数实现字符串左旋转操作的功能。比如输入字符串“abcdefg”和数字 2.该函数左旋转 2 位得到的结果“cdefgab”.

```
public void leftRotateString(String sentence,int index){
        if(sentence==null||index>sentence.length()||index<0){
            return;
        }
        String[] splitString={sentence.substring(0,index),
                sentence.substring(index,sentence.length())};
        StringBuffer resultbBuffer=new StringBuffer();
        for(String s:splitString){
            resultbBuffer.append(reverse(s));
        }
        System.out.println(reverse(resultbBuffer.toString()));
```

```
    }
    public String reverse(String str) {
        char[] array=str.toCharArray();
        for(int i=0;i<(array.length+1)/2;i++)
        {
            char temp=array[i];
            array[i]=array[array.length-1-i];
            array[array.length-1-i]=temp;
        }
        return String.valueOf(array);
    }
```

面试题 43：n 个骰子的点数

题目：把 n 个骰子扔在地上，所有骰子朝上一面的点数之和为 s。输入 n，打印出 s 的所有可能的值出现的概率。

```
public class Problem43 {
    public static void main(String[] args) {
        Problem43 p=new Problem43();
        p.printProbability(2);
    }
    public void printProbability(int number){
        if(number<1)
            return;
        int gMaxValue=6;
        int[][] probabilities=new int[2][];
        probabilities[0]=new int[gMaxValue*number+1];
        probabilities[1]=new int[gMaxValue*number+1];
        int flag=0;
        for(int i=1;i<gMaxValue;i++){
            probabilities[flag][i]=1;
        }
        for(int k=2;k<=number;++k){
            for(int i=0;i<k;i++){
                probabilities[1-flag][i]=0;
            }
            for(int i=k;i<=gMaxValue*k;i++){
                probabilities[1-flag][i]=0;
                for(int j=1;j<=i&&j<=gMaxValue;j++)
    probabilities[1-flag][i]+=probabilities[flag][i-j];
            }
            flag=1-flag;
        }
        double total=Math.pow(gMaxValue, number);
```

```
        for(int i=number;i<gMaxValue*number;i++){
            double ratio=(double)probabilities[flag][i]/total;
            System.out.print(i+" ");
            System.out.println(ratio);
        }
    }
}
```

面试题 44：扑克牌的顺子

题目：从扑克牌中随机抽 5 张牌，判断是不是顺子，即这 5 张牌是不是连续的。2-10 为数字本身，A 为 1，J 为 11，Q 为 12，K 为 13，而大小王可以看成任意的数字。

```
public class Problem44 {
    public static void main(String[] args) {
        int[] array={0,4,6,8,0};
        Problem44 test=new Problem44();
        System.out.println(test.isContinuous(array));
    }
 public boolean isContinuous(int[] number){
     if(number==null){
         return false;
     }
     Arrays.sort(number);
     int numberZero=0;
     int numberGap=0;
     for(int i=0;i<number.length&&number[i]==0;i++){
         numberZero++;
     }
     int small=numberZero;
     int big=small+1;
     while(big<number.length){
         if(number[small]==number[big])
             return false;
         numberGap+=number[big]-number[small]-1;
         small=big;
         big++;
     }
     return (numberGap>numberZero)?false:true;
 }
}
```

面试题 45：圆圈中最后剩下的数字

题目：0,1，...,n-1 这 n 个数排成一个圆圈，从数字 0 开始每次从这个圆圈里删除第 m 个数字。求出这个圆圈里剩下的最后一个数字。

```
public class Problem45 {
    public static void main(String[] args) {
        Problem45 p=new Problem45();
        System.out.println(p.lastRemaining(6, 3));
    }
    public int lastRemaining(int n,int m){
    if(n<1||m<1)
        return -1;
    int last=0;
    for(int i=2;i<=n;i++){
        last=(last+m)%i;
    }
    return last;
}
}
```

面试题 46：求 1+2+...+n

题目：求 1+2+...+n,要求不能用除法、for、while、if、else、switch、case 等关键字及条件判断语句（A?B:C）。

面试题 47：不用加减乘除做加法

题目：写一个函数，求两个整数之和，要求在函数体内不得使用+、-、*、/四则运算符号。

```
public class Problem47 {
    public static void main(String[] args) {
        Problem47 p=new Problem47();
        System.out.println(p.add(8, 16));
    }
    public int add(int num1,int num2){
        int sum,carray;
        do{
            sum=num1^num2;
            carray=(num1&num2)<<1;
            num1=sum;
            num2=carray;
        }while(num2!=0);
```

```
        return num1;
    }
}
```