一、数组排序问题

1. 冒泡排序算法

冒泡排序算法的一趟原理如下：

1. 相邻元素进行两两比较，如果前者大于后者，则交换，否则不变。
2. 每一趟依次向后进行第 1 步，直到最后一个元素。
3. 经过一趟的依次比较后的结果是，最后一个数是本趟比较中最大的数。所谓大数 ”沉入水底“ ，轻的气泡 “向上冒泡”。
4. 继续重复 1、2、3 步骤。每趟只比较到上一步剩下的部分即可（除去最后的已经“沉到底部”的元素）。

详细步骤如下：

```
冒泡排序：比较次数（内层循环）和趟数（外层循环）

[7 3 2 1 9]：

每一趟 找出最大值放到最后

* 总结规律：i<length-1  j<length-i-1
* 第一趟：i=0 j<=3
* 比较第一次：（j和j+1）
* j=0:3 7 2 1 9
* j=1:3 2 7 1 9
* j=2:3 2 1 7 9
* j=3:3 2 1 7 [9]
* 第二趟：i=1 j<=2
* j=0:2 3 1 7 9
* j=1:2 1 3 7 9
* j=2:2 1 3 [7 9]
* 第三趟：i=2 j<=1
* j=0:1 2 3 7 9
* j=1:1 2 [3 7 9]
* 第四趟：i=3 j<=0
* j=0:1 [2 3 7 9]
* 总结规律：i<length-1  j<length-i-1
* 交换（j和j+1）
```

具体代码如下：

```
    public static void maopao(){
        int[] arr = {7,3,12,1,0};
        for (int i = 0; i < arr.length-1; i++) {
            for (int j = 0; j < arr.length-i-1; j++) {
                if(arr[j]>arr[j+1]){
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }
        System.out.println(Arrays.toString(arr));
    }
```

2. 选择排序算法

思路：选择排序每趟选择一个固定位置上的元素，向后逐一和此位置上的数进行比较，如果小于此位置上的数，则交换。

一趟结束后，此位置上的数就是当前排序序列中最小的元素了。

重复上面的步骤即可排序成功。

步骤举例：

```
* 选择排序:每次拿i上的元素和后面的数一一比较
   * [3,6,9,2,7] i 和 j 比较+
    * 第一趟：i=0:[3] 6 9 2 7  | 1<=j<=4
    * j=1:[3] 6 9 2 7 --arr[i]=3和arr[j]=6比较
    * j=2:[3] 6 9 2 7 --arr[i]=3和arr[j]=9比较
    * j=3:[2] 6 9 3 7 --arr[i]=3和arr[j]=2比较
    * j=4:[2] 6 9 3 7 --arr[i]=2和arr[j]=7比较
    * 第二趟：i=1: 2[6]9 3    | 2<=j<=4
    * j=2:2[6]9 3 7  ---arr[i]=6和arr[j]=9比较
    * j=3:2[3]9 6 7  ---arr[i]=6和arr[j]=3比较---
    * j=4:2[3]9 6 7  ---arr[i]=3和arr[j]=7比较---
    * 第3趟：i=2: 2 3[9] 6 7 | 3<=j<=4
    * j=3:2 3 [6] 9 7  ---arr[i]=9和arr[j]=6比较---
    * j=4:2 3 [6] 9 7  ---arr[i]=6和arr[j]=7比较---
    * 第4趟：i=3: 2 3 6[9]7  | 4<=j<=4
    * j=4: 2 3 6[7]9   ---arr[i]=9和arr[j]=7比较---
/**
     * 总结:
     * i--0~length-1
     * j--i+1~length-1
     * 比较:
     * i和j
     */
    public static void selorder(int[] arr){
//        int[] arr = {3,6,9,2,7};
```

```
        for (int i = 0; i < arr.length-1; i++) {
            for (int j = i+1; j <= arr.length-1; j++) {
                if(arr[i]>arr[j]){
                    int temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                }
            }
        }
        //System.out.println(Arrays.toString(arr));
    }
```

3. 插入排序算法

思路：从第二个位置上开始，每次选择一个位置上元素，和前面的数逐一比较，如果小于前面的数，则前面数后移，直到大于前面的数，则停止移动，插入位置。这样每趟走完之后，前面的已经插入的序列一定是有序。

就如生活中的起扑克牌一样，每起一张都会插入手中，但是插入后，手中的牌一直是有序的。

步骤举例：

```
* 插入排序
   * 第一趟：假设第一个元素是有序的，从第二个开始和前面有序的序列中插入到合适位置
    * 第一趟：[9] (2) 1 7 4 3 6
    * 第一次：[2 9] 1 7 4 3 6
    *
    * 第二趟：[2 9] （1） 7 4 3 6
    * 第一次：2 1 9 7 4 3 6
    * 第二次：1 2 9 7 4 3 6
    *
    * 第三趟：[1 2 9] （7） 4 3 6
    * 第一次：....
    * 插入排序每一趟的结果是保证前面的是有序的序列。
    public static void insertOrder() {
        int[] arr = {9,8,7,6,5,4,3,2};
        for (int i = 1; i < arr.length; i++) {
            int insert = arr[i];
            for (int j = i; j > 0; j--) {
                //前面的数大于已经选择插入的数，则后移
                if(arr[j-1]>insert) {
                    arr[j] = arr[j-1];
                    arr[j-1] = insert;
                }else {
                    arr[j] = insert;
                    break;
                }
            }
        }
        System.out.println("排序后："+Arrays.toString(arr));
```

```
    }
```

4. 快速排序算法

思路：需要使用到递归。

每次选择一个元素，通过一趟的过程，前后比较，从而找到自己的合适位置，即此位置前的都比次数小，此位置之后的，都比次数大。

然后再对已经分割的两部分，重复递归使用相同的方法来找每个数的位置。递归嵌套即可完成排序。

```
package com;

import java.util.Arrays;

public class QuickSort {
    public static void quickSort(int[] arr,int low,int high){
        int i,j,temp,t;
        if(low>high){
            return;
        }
        i=low;
        j=high;
        //temp就是基准位
        temp = arr[low];

        while (i<j) {
            //先看右边，依次往左递减
            while (temp<=arr[j]&&i<j) {
                j--;
            }
            //再看左边，依次往右递增
            while (temp>=arr[i]&&i<j) {
                i++;
            }
            //如果满足条件则交换
            if (i<j) {
                t = arr[j];
                arr[j] = arr[i];
                arr[i] = t;
            }

        }
        //最后将基准为与i和j相等位置的数字交换
         arr[low] = arr[i];
         arr[i] = temp;
        //递归调用左半数组
        quickSort(arr, low, j-1);
        //递归调用右半数组
        quickSort(arr, j+1, high);
```

```
    }


    public static void main(String[] args){
        int[] arr = {10,7,2,4,7,62,3,4,2,1,8,9,19};
        quickSort(arr, 0, arr.length-1);
        System.out.println(Arrays.toString(arr));
    }

}
```

二、数组相关其他问题

5. 如何把数组中的元素逆序?

思路：逆序就是把数组的元素前后倒置。

我们可以把前后两个数 i 和 arr.length-1-i 位置上的数进行交换，但是一定注意交换次数是长度的一半。

举例如下：

```
     *  0-3  1-2
     * {3,8,6,2}---i----
     * {2,6,8,3}
     *
     * 0-4 1-3 2-2 3-1 4-0  ---i-length-1-i
     * {1,2,3,4,5}
    public static void reverse(){
        int[] arr = {3,6,8,9,1};
        System.out.println(Arrays.toString(arr));
        for (int i = 0; i < arr.length/2; i++) {
            int temp = arr[i];
            arr[i] = arr[arr.length-1-i];
            arr[arr.length-1-i] = temp;
        }

        System.out.println(Arrays.toString(arr));
    }
```

6. 把一个数插入有序数组，保持数组依然有序

思路：把这个数依次和数组的第一个开始的每个数进行比较，找到第一个比自己大的数的位置即是要插入的位置。

但是从后向前找位置，其实效率更高，因为在查找过程中就可以完成后移动的操作。

```
/**
     * 数组插入
     * {3,5,6,2,9,8,0}  最后一位保留位置
     * index = 3;插入数7
```

```
     * {3,5,6,7,2,9,8}
     *
     */
    public static void insert(){
        int[] arr = {3,5,6,2,9,8,0};
        int index = 3;
        int num = 7;
        System.out.println(">>>=="+Arrays.toString(arr));
        int i = arr.length-1;
        for (; i>index; i--) {
            arr[i] = arr[i-1];
        }
        arr[i] = num;
        System.out.println(">>>=="+Arrays.toString(arr));
    }
```

7. 实现双色球彩票随机出号，六个红球，一个篮球，并顺序输出

思路：

- 第一步：按数组元素个数，随机生成放入
- 第二步：修改放入的规则，根据条件放入
- 第三步：解决条件判断获取问题

双色球：六个红球（1~33）同一注中不能重复，一个蓝球（1~16）。

写出出球算法：

```
         [6, 27, 10, 10, 14, 5]
         i=0
         [6,0,0,0,0,0]
         i=1 ---27
         [6,27,0,0,0,0]
         i=2 ---10
         [6,27,10,0,0,0]
         i=3 ---10 i=4
         [6,27,10,0,23,0]
    public static void caipiao(){
        Random random = new Random();
        int[] red = new int[6];
        for (int i = 0; i < red.length; ) {
            int rand = random.nextInt(33);//产生一个随机数
            //放入之前 要把rand和数组中的元素 逐一比较 看是否存在，存在则重新随机

            boolean isExist = false;
            //给定rand数  判断rand是否在数组red中存在
            for (int j = 0; j < i+1; j++) {
                if(rand==red[j]){//判断是否数组中存在此数
                    isExist = true;//表示 数是存在于数组中的
                    break;
```

```
                }
            }
            //放入
            if(isExist==false){
                red[i] = rand;
                i++; //注意：只有放入成功后，才能修改下次放入的下标位置
            }
        }
      //生成一个篮球
        int blue = random.nextInt(16);
        Arrays.sort(red);//排序
        System.out.println(Arrays.toString(red)+"+"+blue);
    }
```

8. 使用二维数组，实现杨辉三角

思路：杨辉三角，是二项式系数在三角形中的一种几何排列。在二维数组中的第一列和中间对等列为1，其余元素的值为上一行前两个元素之和： arr[i][j] = arr[i-1][j]+arr[i-1][j-1]。

```
public static void yanghui2(){
        int[][] arr = new int[6][];

        for (int i = 0; i < arr.length; i++) {
            arr[i] = new int[i+1];
            for (int j = 0; j < arr[i].length; j++) {
                if(i==j||j==0){
                    arr[i][j] = 1;
                }else{
                    arr[i][j] = arr[i-1][j]+arr[i-1][j-1];
                }
            }
        }

        print(arr);
    }

    /**
     * 打印二维数组
     * @param arr
     */
    public static void print(int[][] arr){
        for (int i = 0; i < arr.length; i++) {
            System.out.println(Arrays.toString(arr[i]));
        }
    }
```

三、查找问题

9. 顺序查找法

给定数字，在数组中查找所在的下标，没有找到则返回 -1。

思路：顺序查找是按照序列原有顺序对数组进行遍历比较查询的基本查找算法，只需要按数组下标索引一次对比即可。

根据题目要求找不到返回-1,所以可以假设找不到，设置变量标志位 int index = -1。然后依次查找，找到则替换标志为位置索引。index = i。

```
/**
     * 在指定的数组中查找指定值num
     * 如果找到了，则返回次数在数组中的下标
     * 如果找不到，则返回-1
     *
     */
    public static void findNum(int num){
        int[] arr = {6,9,3,8,20,11};
        int index = -1;

        for (int i = 0; i < arr.length; i++) {
          //如果找到 则跳出 且记录下标位置
            if(arr[i]==num){
                index = i;
                break;
            }
        }

        System.out.println("下标为："+index);
    }
```

10. 查找最大数

思路：题目要求实现，在一个数组序列中找出其中最大那个数。我们可以使用一个变量存放第一个数，然后后面取出每个数进行比较，大于此变量的数则替换掉此变量。直到查找结束，留在变量中的值即为最大数。

```
    public static void findMax(){
        int[] arr = {7,3,9,11,1};
          //用于保存最大数
        int max = 0;
        for (int i = 0; i < arr.length; i++) {
            //和max比较 大的数留下
            if(arr[i]>max){
                max = arr[i];
            }
        }
        System.out.println(">>>>----"+max);
    }
```

11. 二分查找法，前提是所查找的数组必须是有序的

思路：在有顺序的序列中，每次取出查找范围内的中间数进行比较，如果大于中间数，则说明要找的数在后面，否则在前面。依次调整开始范围和结束范围即可。

```
private static int binarySearch0(int[] a, int fromIndex, int toIndex, int key)
{
          //记录查找的开始位置
        int low = fromIndex;
          //记录查找的结束位置
        int high = toIndex;
        //如果结束位置不大于开始位置 则可以继续查找
        while (low <= high) {
          //除以2 找出中间位置的数
            int mid = (low + high) >>> 1;

            if (a[mid] < key)
                low = mid + 1;//如果要找的数大于了中间数，则下次查找的开始位置调整为中间数的下一个数
            else if (a[mid] > key)
                high = mid - 1;//如果找的数小于中间数，则下次查找的结束位置为中间数的上一个位置
            else
                return mid; // key found
        }
        return -(low + 1); // key not found.
    }
```

四、进制转换问题

12. 把一个十进制转换为八进制

思路：十进制转为其他数的方式就是整除取余数的方法。每次循环整除 8 取余数，直到除到 0 为止。

```
    public static String toOctString(int num) {
        //保存余数
        String str = "0";

        do {
            int mod = num%8;//取出余数
            str = mod + str;//拼接余数
            //修改原值  整除
            num = num/8;
        }while(num!=0);

        return str;
    }
```

13. 把一个十进制数转为二进制

思路：二进制即为逢 2 进 1。只有 0 和 1。所以可以每次整除 2，取出余数，逆序拼接成字符串。直到被整除到 0 为止。最后字符串中结果即为转换后的结果。

```
    public static String toOctString(int num) {
        //保存余数
        String str = "";

        do {
            int mod = num%8;//取出余数
            str = mod + str;//拼接余数
            //修改原值  整除
            num = num/8;
        }while(num!=0);

        return "0"+str;
    }
```

14. 把一个十进制数转为十六进制

思路：基本和上面转换思想相同，但是十六进制和其他二进制八进制不同之处就是多出了 ABCDEF 六个字符。定义一个数组序列，使用下标和字符做出对应映射。这样即可巧妙的解决对应的判断问题。否则，写很多个 if 判断就太繁琐了。

```
public static String toHexString(int num) {
        char[] ch =
{'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
        //保存余数
        String str = "";

        do {
            int mod = num%16;//取出余数
            str = ch[mod] + str;//拼接余数
            //修改原值  整除
            num = num/16;
        }while(num!=0);

        return "0x"+str;
    }
```

五、日期问题

15. 输入年份、月份、日期，打印出当前月份的日历表

思路关键点：本题的关键点就是需要控制换行和每月天数问题。

1. 先设置当月 1 日，然后计算出 1 日是星期几，这样可以确定 1号在周几的开始位置；
2. 控制每周日换行或每7天换行，但是注意第一周需要调整开始数。

```java
public class HomeWork {
    private static Scanner sc = new Scanner(System.in);
    public static void main(String[] args) {
        System.out.println("请输入年份：");
        int year = sc.nextInt();
        System.out.println("请输入月份：");
        int month = sc.nextInt();
        System.out.println("请输入日期：");
        int date = sc.nextInt();


        Calendar cal = Calendar.getInstance();

        //设置年份
        cal.set(Calendar.YEAR, year);
        //设置月份
        cal.set(Calendar.MONTH, month-1);  //注意***********
        //设置日期  注意：先设置日期  为1日。因为获取月第一天为周几?
        cal.set(Calendar.DATE, 1);

        //获取1日 周几
        int week = cal.get(Calendar.DAY_OF_WEEK);

        //打印星期
        String header = "星期日\t星期一\t星期二\t星期三\t星期四\t星期五\t星期六\t";
        System.out.println(header);
        //空格数
        for (int i = 0; i < week-1; i++) {
            System.out.print("\t");
        }

        //当月总天数
        int days = cal.getActualMaximum(Calendar.DAY_OF_MONTH);

        //开始打印日期
        for (int i = 1; i <= days; i++) {
            if(i==date) {
                System.out.print("["+i+"]\t");
            }else {
                System.out.print(i+"\t");
            }
            //判断是否需要换行
            if((i+week-1) % 7==0) { //不能直接使用i ,需要使用空格数 修正
                System.out.println();
            }
        }
    }
}
```

16. switch

分别输入年月日，计算出当天是年中第几天?

思路：本题主要限制的实现方式只能使用 switch。考点主要是如何能巧妙的使用 break，这里主要是在 case 月份，要反着来写即可。

```
/**
     * switch:分别输入年月日，计算出当天是年中第几天?
     */
    public static void daysOfYear() {
        int year = 2019;
        int month = 3;
        int date = 12;

        int days = 0;
        switch (month) {
        case 12:
            days += 30; //11月份的天数
        case 11:
            days += 31; //上个月10月份的天数
        case 10:
            days += 30; //上个月9月份的天数
        case 9:
            days += 31;
        case 8:
            days += 31;
        case 7:
            days += 30;
        case 6:
            days += 31;
        case 5:
            days += 30;
        case 4:
            days += 31;
        case 3:
            days += 28;
        case 2:
            days += 31;
        case 1:
            days += date;//1月份不需要累加其他月份天数
        default:
            break;
        }
        //判断闰年    能被4整除但不能被100整除   或者  能被400整除
        if((year%4==0&&year%100!=0)||year%400==0) {
            days++;
        }
        System.out.println(year+"年"+month+"月"+date+"日是当年中的第"+days+"天");
```

```
    }
```

六、乘法表问题

17. 嵌套循环实现打印九九乘法表

思路：嵌套循环。主要控制内层循环条件。每行的列数正好为当前的行号。所以可以 j<=i 控制列数。

- 第一行：一列
- 第二行：两列
- ……

所以列数正好为行号。

```
    public static void cheng() {
        for (int i = 1; i <= 9; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print(i+"*"+j+"="+(i*j)+"\t");
            }
            System.out.println();
        }
    }
```

18. 使用单层循环实现九九乘法表

关键思路：在一层循环中使用了两个条件。当没达到 j==i 时阻止了i++，从而实现了列数控制的效果。

```
public static void main(String[] args) {
        // 循环控制实现输出九行
        for (int i = 1, j = 1; i < 10; j++) {
            System.out.print(i + "*" + j + "=" + (i * j) + "\t");
            //当行数等于列数时换行，但是列数重0开始，行数加一
            if (j == i) {
                System.out.println();
                j = 0;
                i++;
            }
        }
    }
```

七、应用题

19. 小明投入股市10000元，股市涨跌，第一天涨10%，第二天跌10%，第三天涨10%.....如此交替，几天后小明的钱少于1000?

思路：本题就是普通的循环题。即需要判断奇数天时减去 10%。偶数天时，加上 10% 即可。

```
public static void testWhile() {
        double money = 10000;  //110  110-11=99  99+9=108   108-10 98
```

```
        int days = 0;
        while(money > 1000) {
            days++;
            //修改条件变量
            if(days%2==0) {
                //跌
                money -= money*0.1;
            }else {
                //涨
                money += money*0.1;
            }
        }

        //打印
        System.out.println(money+"=days=:"+days);
    }
```

20. 写一个方法实现判断一个数是否为素数?

思路：只能被 1 和本身整除的数，除以从 2 到本数 -1，只有一个数能被整除，则此数一定不是素数（不用循环）。如果所有的都不能被整除才是素数。

```
    public static boolean isShu(int num) {
        //标志位
        boolean is = true; //假设就是素数

        for (int i = 2; i <= num/2; i++) {
            if(num%i==0) {//数除以i 被整除了  则可以确定不是素数
//                System.out.println("====不是素数=");
                is = false;
                break;
            }
        }
        return is;
    }
```

21. 一篮子鸡蛋，三三余二，四四余三，五五数四，篮子有至少几个鸡蛋?

思路：三三余二，其实就是三个三个数数，剩下两个。所以就是整除余数为二。所以我们可以依次取余数看是否满足条件。

```
    public static void main(String[] args) {
        for (int i = 1; i < Integer.MAX_VALUE; i++) {
            if(i%3==2 && i%4==3 && i%5==4) {
                System.out.println("次数是："+i);
                break;//找到后跳出循环
            }
        }
    }
```

22. 有 n 个人围成一圈，顺序排号。从第一个人开始报数（从 1 到 3 报数），凡报到 3 的人退出圈子，问最后留下的是原来第几号的哪位?

思路：本题只需要循环取出第三个数设为 false，表示退出。依次结束后，剩下为 true 的即为留下的。

```
public static void main(String[] args) {
        System.out.print("请输入一个整数：");
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        scan.close();
        // 定义数组变量标识某人是否还在圈内
        boolean[] isIn = new boolean[n];
        for (int i = 0; i < isIn.length; i++)
            isIn[i] = true;
        // 定义圈内人数、报数、索引
        int inCount = n;
        int countNum = 0;
        int index = 0;
        while (inCount > 1) {
            if (isIn[index]) {
                countNum++;
                if (countNum == 3) {
                    countNum = 0;
                    isIn[index] = false;
                    inCount--;
                }
            }
            index++;
            if (index == n)
                index = 0;
        }
        for (int i = 0; i < n; i++)
            if (isIn[i])
                System.out.println("留下的是：" + (i + 1));
    }
```

23. 求 1+2!+3!+...+20!

思路：单独一个方法实现求一个数的阶乘。然后循环计算 1~20 即可实现题目要求。

```
public static void main(String[] args) {
        long sum = 0;
        for (int i = 0; i < 20; i++)
            sum += factorial(i + 1);
        System.out.println(sum);
    }

    // 阶乘
    private static long factorial(int n) {
        int mult = 1;
          //从1到本身累计相乘
        for (int i = 1; i < n + 1; i++)
            mult *= i;
        return mult;
    }
```

24. 分数序列求和

2/1,3/2,5/3,8/5,13/8,21/13...求出这个数列的前 20 项之和。

分析：请抓住分子与分母的变化规律。

思路：本题就是一个非常有规律的变化数序列。分母为 1、2、3、5、8.....即从第三个开始为前两个数之和。分子为 2、3、5、8.....即从第三个开始也为前两个相邻数之和。

所以在循环题中使用两个变量分别存储前两个数，从而计算出下一次数的值。

```
public static void main(String[] args) {
        double n1 = 1;
        double n2 = 1;
        double fraction = n1 / n2;
        double Sn = 0;
        for (int i = 0; i < 20; i++) {
            double t1 = n1;
            double t2 = n2;
            n1 = t1 + t2;
            n2 = t1;
            fraction = n1 / n2;
            Sn += fraction;
        }
        System.out.print(Sn);
    }
```

25. 猴子吃桃问题

猴子第一天摘下若干个桃子，当即吃了一半，还不瘾，又多吃了一个第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第 10 天早上想再吃时，见只剩下一个桃子了。求第一天共摘了多少。

分析：采取逆向思维的方法，从后往前推断。即从第 10 天开始往前推，每天两倍数加 2，推出第一个的结果。

```
public static void main(String[] args){
    int m = 1;
     for(int i=10;i>0;i--){
         m = 2*m + 2;
     }
     System.out.println("小猴子共摘了"+m+"桃子");
}
```

26. 有 1、2、3、4 个数字，能组成多少个互不相同且无重复数字的三位数？都是多少?

思路：可填在百位、十位、个位的数字都是 1、2、3、4。组成所有的排列后再去掉不满足条件的排列。

```
public static void main(String[] args) {
        int count = 0;
        int n = 0;
        for (int i = 1; i < 5; i++) {
            for (int j = 1; j < 5; j++) {
                if (j == i)
                    continue;
                for (int k = 1; k < 5; k++) {
                    if (k != i && k != j) {
                        n = i * 100 + j * 10 + k;
                        System.out.print(n + " ");
                        if ((++count) % 5 == 0)
                            System.out.println();
                    }
                }
            }
        }
        System.out.println();
        System.out.println("符合条件的数共：" + count + "个");
    }
```

27. 自由落体反弹

一球从 100 米高度自由落下，每次落地后反跳回原高度的一半；再落下，求它在第 10 次落地时，共经过多少米？第 10 次反弹多高?

思路：使用两个变量分别记录初始的高度 h。每次高度折半。但是总路程需要记录下落到地面和反弹高度。所以需要记录累计每次的高度路程。从而算出结果。

```
public static void main(String[] args) {
        fun();
```

```
    }

    private static void fun() {
        double h = 100;
        double d = 100;
        for (int i = 1; i <= 10; i++) {
            d = d + h;
            h = h / 2;
        }

        System.out.println("路程: " + d);
        System.out.println("高度: " + h / 2);

    }
```

28. 完数

一个数如果恰好等于它的因子之和，这个数就称为“完数”，例如 6=1+2+3。编程找出 1000 以内的所有完数。

思路：就是循环分解，逐一循环尝试判断。

```
// 求完数
    private static void compNumber(int n) {
        System.out.println(n + "以内的完数: ");
        for (int i = 1; i < n + 1; i++) {
            int sum = 0;
            for (int j = 1; j < i / 2 + 1; j++) {
                if ((i % j) == 0) {
                    sum += j;
                    if (sum == i) {
                        System.out.print(i + " ");
                    }
                }
            }
        }
    }
```

29. 求最大公约数和最小公倍数

```
// 求最大公约数和最小公倍数
    private static void max_min(int m, int n) {
        int temp = 1;
        int yshu = 1;
        int bshu = m * n;
        if (n < m) {
            temp = n;
            n = m;
            m = temp;
```

```
        }
        while (m != 0) {
            temp = n % m;
            n = m;
            m = temp;
        }
        yshu = n;
        bshu /= n;
        System.out.println(m + "和" + n + "的最大公约数为" + yshu);
        System.out.println(m + "和" + n + "的最小公倍数为" + bshu);
    }
```

30. 将一个正整数分解质因数

例如：输入 90，打印出 90=233*5。

思路：从 2 开始依次整除，找出所有不能被整除的因子数。

```
private static void decompose(int n) {
        System.out.print(n + "=");
        for (int i = 2; i < n + 1; i++) {
            while (n % i == 0 && n != i) {
                n /= i;
                System.out.print(i + "*");
            }
            if (n == i) {
                System.out.println(i);
                break;
            }
        }
    }
```

31. 打印出所有的“水仙花数”

“水仙花数”：一个三位数，其各位数字立方和等于该数本身。例如：153 是一个“水仙花数”，因为 153 等于 1 的三次方加上 5 的三次方加上 3 的三次方。

思路分析：利用 for 循环控制 100~999 个数，每个数分解出个位、十位、百位。

```
// 判断水仙花数
    private static boolean isLotus(int lotus) {
        int m = 0;
        int n = lotus;
        int sum = 0;
        m = n / 100;
        n -= m * 100;
        sum = m * m * m;
        m = n / 10;
        n -= m * 10;
```

```
            sum += m * m * m + n * n * n;
            if (sum == lotus)
                return true;
            else
                return false;
        }
```

八、递归问题

32. 古典问题：一对兔子

有一对兔子，从出生后第 3 个月起每个月都生一对兔子，小兔子长到第三个月后每个月又生一对兔子，假如兔子都不死，问每个月的兔子对数为多少?

思路：斐波那契数列（Fibonacci sequence）

程序分析： 兔子的规律为数列 1,1,2,3,5,8,13,21...，当 n=1 或 n=2时，分别为 1。

从 n=3 开始 f(n) = f(n-1) + f(n-2)。所以转化为程序则如下：

```
public static void main(String[] args) {
        int n = 10;
        System.out.println("第" + n + "个月兔子总数为" + fun(n));
    }

    private static int fun(int n) {
        if (n == 1 || n == 2)
            return 1;
        else
            return fun(n - 1) + fun(n - 2);
    }
```

33. 使用递归实现 n! 的阶乘

思路：求一个数的阶乘，既可以使用循环，也可以使用递归。本地要求使用递归。

把公式分解后 n! = n*(n-1)! ；但是 1的阶乘为 1。所以我们可以定义一个方法 jie(int n)，假定方法就是求阶乘的方法，则每次 n! = n * jie(n-1)。这样就实现了方法逻辑了。

```
public static void main(String[] args) {
        int n = 10;
        System.out.println(n+"阶乘为: " + jie(n));
    }

    private static int jie(int n) {
        if(n==1) {
            return 1;
        }else {
            return n*jie(n-1);
        }
    }
```

34. 使用递归遍历一个给定的目录下的所有文件

思路：本题要求遍历出目录下所有的子文件及子目录下的所有文件及子孙后代文件。这就只能通过递归来实现。

也是声明一个查找下一级文件的方法，然后根据判断是目录则递归调用方法即可找出所有文件。

```
public static void main(String[] args) {
        find(new File("D:\\201806shixi"));
    }
    public static void find(File file) {
        if(file.isDirectory()) {
            File[] files = file.listFiles();
            for (File file2 : files) {
                find(file2); ///递归
            }
        }else {
            System.out.println(file.getAbsolutePath());
        }
    }
```

35. 用递归实现字符串倒转

思路：字符串倒序可以有多种实现方法，但是本地要求递归。所以我们需要找出相同的可以重复的方法来递归完成。

例如：“hello”

- 第一次：截取第一个字符“h”，截取剩下“ello”，使用“ello”+“h”；
- 第二次：那么“ello”字符串，可以使用相同的方法，“llo”+“e”；
- 第三次：“lo”+“l”；
- 第四次：“o”+“l”；
- 第五次：“o”长度已经是一个了，所以依次返回给上一步“olleh”。

```
public class StringReverse {
 public static String reverse(String originStr) {
     if(originStr == null || originStr.length()== 1) {
         return originStr;
     }
     return reverse(originStr.substring(1))+ originStr.charAt(0);
 }

 public static void main(String[] args) {
     System.out.println(reverse("hello"));
 }
}
```

九、循环打印问题

36. 打印直角三角形

思路：直角三角形规律很明显。

- 当第一行 i=0 时，1个星号
- 当第二行 i=1 时，2个星号
- 当第三行 i=2 时，3个星号
- ……

依次类推，规律为列数 j < i+1。

另外，注意每一行后，需要换行。

```
/**
     * 直角三角形------i----j
     * *  ---------i=0  1个
     * **----------i=1  2个
     * ***---------i=2  3个
     * ****--------i=3  4个
     * *****-------i=4  5个
     */
    public static void test1(){
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < i+1; j++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
```

37. 打印右直角三角形

思路：控制空格和星号个数，直角三角形规律很明显。

- 当第一行 i=0 时，1个星号

- 当第二行 i=1 时，2个星号
- 当第三行 i=2 时，3个星号
- ……

依次类推，规律为列数 j < i+1。

而空格数量正好相反，分别为 4个、3 个、2 个、1 个、0个。

使用规律的公式：行数-i-1。另外，注意每一行后，需要换行。

```
/**
     * 直角三角形------i----j---------空格规律
     *     *  ------i=0  1个   ------4   5-i-1
     *    **--------i=1  2个   ------3
     *   ***--------i=2  3个  -------2
     *  ****--------i=3  4个 -------1
     * *****--------i=4  5个 -------0
     */
    public static void test2(){
        for (int i = 0; i < 5; i++) {

            //打印空格的循环
            for (int j = 0; j < 5-i-1; j++) { ///总结关系  j=5-i-1
                System.out.print(" ");// 打印空格
            }


            //打印星号
            for (int j = 0; j < i+1; j++) { //j===i+1
                System.out.print("*");
            }
            System.out.println();
        }
    }
```

38. 等腰三角形

思路：分析本题的规律，也是空格数和星号个数来完成的。

- 空格数：分别为 4、3、2、1、0 。公式结果：行数-i-1
- 星号数：分别为 1、3、5、7、9。 公式结果：2*i+1

```
/**
     * 等腰三角形
     *      *
     *     ***
     *    *****
     *   *******
     *  *********
     */
```

```
    public static void rect2() {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4-i-1; j++) {
                System.out.print(" ");
            }
            for (int j = 0; j < 2*i+1; j++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
```

39. 打印菱形

思路：本题就是需要分为两部分打印就可以了。上半部分就是上一题中的等腰三角形。而下一部分也是倒三角形。

规律：

- 空格数：1、2、3
- 星号数：5、3、1

公式：4-i-1

```
/**2*i+1
     *                    i   空格    星号
     *    *     --------- 0 -- 3 --1
     *   ***    --------- 1 -- 2 --3
     *  *****   --------- 2 -- 1 --5
     * *******  --------- 3 -- 0 --7
     *  *****   --------- 2 -- 1 --5 ===========另一个循环
     *   ***    --------- 1 -- 2 --3
     *    *     --------- 0 -- 3 --1
     */

    public static void test5(){
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4-i-1; j++) {
                System.out.print(" ");
            }
            for (int j = 0; j < 2*i+1; j++) {
                System.out.print("*");
            }
            System.out.println();
        }

        for (int i = 2; i >=0; i--) {
            for (int j = 0; j < 3-i; j++) {
                System.out.print(" ");
            }
```

```
            for (int j = 0; j < 2*i+1; j++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
```

十、设计模式问题

40. 工厂模式

某软件公司开发的项目可以用于不同行业的客户，在客户中有的使用 SQL Server 数据库，有的是 MySQL，有的使用的是 Oracle，如何设计底层连接数据库类 DAO，可以方便地切换。请选用相应的设计模式设计该系统。

思路：由于连接数据库可能是多样的，为了让开发者方便的使用相同的接口来开发。我们可以使用工厂模式。让工厂类来提供不同数据库 DAO 类的创建，使用者只需要知道类型即可。

先设计一个接口，各个类型的数据库 DAO 实现此接口。工厂类负责创建此接口的实现。

```
package com.wdzl.demo;

public interface IDao {
    public void insert();
}
package com.wdzl.demo;

public class DaoFactory {
    public static final int DAO_MYSQL = 1;
    public static final int DAO_SQLSERVER = 2;
    public static final int DAO_ORACLE = 3;

    public static IDao newDao(int type){
        if(type==DAO_MYSQL){
            return new MysqlDAO();
        }else if(type==DAO_ORACLE){
            return new OracleDAO();
        }else if(type==DAO_SQLSERVER){
            return new SqlServerDAO();
        }else{
            throw new RuntimeException("没有此类型的数据库！");
        }
    }
    /**
     * mysql的数据库访问层
     * @author Administrator
     *
     */
    public static class MysqlDAO implements IDao{
```

```
        @Override
        public void insert() {
            System.out.println("插入到MySql中");
        }
    }
    public static class SqlServerDAO implements IDao{
        @Override
        public void insert() {
            System.out.println("插入到 SqlServer 中");
        }
    }
    public static class OracleDAO implements IDao{
        @Override
        public void insert() {
            System.out.println("插入到 Oracle 中");
        }
    }
}
```

41. 生成器模式

网易游戏软件公司决定开发一款基于角色扮演的多人在线网络游戏，玩家可以在游戏中扮演虚拟世界中的一个特定角色，角色根据不同的游戏情节和统计数据（例如力量、魔法、技能等）具有不同的能力，角色也会随着不断升级而拥有更加强大的能力。作为该游戏的一个重要组成部分，需要对游戏角色进行设计，而且随着该游戏的升级将不断增加新的角色。通过分析发现，游戏角色是一个复杂对象，它包含性别、面容等多个组成部分，不同类型的游戏角色，其性别、面容、服装、发型等外部特性都有所差异，例如“天使”拥有美丽的面容和披肩的长发，并身穿一袭白裙；而“恶魔”极其丑陋，留着光头并穿一件刺眼的黑衣。

思路：无论是何种造型的游戏角色，它的创建步骤都大同小异，都需要逐步创建其组成部分，再将各组成部分装配成一个完整的游戏角色。生成器模式设计模式来实现游戏角色的创建。

```
//生成器模式
package com.product;
//产品人物类
public class Actor {
    private String Type;//脸色类型
    private String sex;//性别
    private String face;//脸型
    private String costume;//服装
    private String hairstyle;//发型
添加get.set方法
}


package com.build;
import com.product.Actor;
public abstract class ActorBuilder {
    //抽象建造者
```

```
    protected Actor actor=new Actor();
    public abstract void buildType();
    public abstract void buildSex();
    public abstract void buildFace();
    public abstract void buildCostume();
    public abstract void buildHairstyle();
    public Actor createActor(){
        return actor;
    }
}
```

```
package com.build.Impl;
import com.build.ActorBuilder;
//天使角色建造器:具体建造者
public class AngelBuilder extends ActorBuilder {
    public void buildType() {actor.setType("天使"); }
    public void buildSex() {actor.setSex("神"); }
    public void buildFace() {actor.setFace("美丽"); }
    public void buildCostume() { actor.setCostume("白裙"); }
    public void buildHairstyle() {actor.setHairstyle("披肩的长发"); }
}
```

```
package com.build.Impl;
import com.build.ActorBuilder;
//魔鬼角色建造器:具体建造者
public class GhostBuilder extends ActorBuilder {
    public void buildType() { actor.setType("魔鬼"); }
    public void buildSex() { actor.setSex("妖"); }
    public void buildFace() { actor.setFace("丑陋"); }
    public void buildCostume() { actor.setCostume("刺眼的黑衣"); }
    public void buildHairstyle() { actor.setHairstyle("光头"); }
    }
```

```
package com.director;
import com.build.ActorBuilder;
import com.product.Actor;
//构建器
public class DirectorController {
    public Actor construct(ActorBuilder actorBuilder){
        actorBuilder.buildType();
        actorBuilder.buildSex();
        actorBuilder.buildFace();
        actorBuilder.buildCostume();
        actorBuilder.buildHairstyle();
        return actorBuilder.createActor();
    }
```

```
}


//测试类
package wtf;
import com.build.ActorBuilder;
import com.build.Impl.AngelBuilder;
import com.build.Impl.GhostBuilder;
import com.director.DirectorController;
import com.product.Actor;
public class Driver {
    public static void main(String[] args) {
        System.out.println("    天使    ");
        DirectorController directorController=new DirectorController();
        ActorBuilder actorBuilder=new AngelBuilder();
        Actor angle=directorController.construct(actorBuilder);
        String type=angle.getType();
        System.out.println(type+"的外观:\t");
        System.out.println("性别:\t"+angle.getSex());
        System.out.println("面容:\t"+angle.getFace());
        System.out.println("服装:\t"+angle.getCostume());
        System.out.println("发型:\t"+angle.getHairstyle());

        System.out.println("    恶魔    ");
        ActorBuilder actorBuilder1=new GhostBuilder();
        Actor ghost = directorController.construct(actorBuilder1);
        String type2=ghost.getType();
        System.out.println(type2+"的外观:\t");
        System.out.println("性别:\t"+ghost.getSex());
        System.out.println("面容:\t"+ghost.getFace());
        System.out.println("服装:\t"+ghost.getCostume());
        System.out.println("发型:\t"+ghost.getHairstyle());
    }
}
```

42. 单例设计模式

某公司开发了一个服务器负载均衡软件，该软件运行在一台负载均衡服务器上，可以将并发访问和数据流量分发到服务器集群中的多台设备上进行并发处理。需要确保负载均衡器的唯一性，只能有一个负载均衡器来负责服务器的管理和请求的分发，否则将会带来服务器状态的不一致以及请求分配冲突等问题。

思路：要确保负载均衡器的唯一性，可以使用单例设计模式来解决。

```
//服务器
public class CustomServer {
    public String name;
    public int Size;
    public String getName() {return name;}
```

```
    public void setName(String name) {this.name = name;}
    public int getSize() {return size;}
    public void setSize(int size) {this.size = size; }
public CustomServer(String name) { this.name = name; }
}


//服务器负载均衡(Load Balance)软件
package com.wtf;
import java.util.Random;
public class LoadBalancer
{
    // 私有静态变量，存储唯一实例
    private static LoadBalancer instance = null;
    // 服务器集合
    private ArrayList<CustomServer> serverList = null;
    // 私有构造函数
    private LoadBalancer()
    {
        serverList = new ArrayList<CustomServer>();
    }
    // 公共静态成员方法，返回唯一实例
    public  synchronized static LoadBalancer getLoadBalancer() {
        if (instance == null) {
            instance = new LoadBalancer();
        }
        return instance;
    }
    // 添加一台Server
    public void addServer(CustomServer server)
    {
        serverList.add(server);
    }
    // 移除一台Server
    public void removeServer(String serverName)
    {
         for(CustomServer server:serverList){
            if (server.name.equals(serverName))
            {
                serverList.remove(server);
                break;
            }
        }
    }
    // 获得一台Server - 使用随机数获取
    private Random rand = new Random();
    public CustomServer getServer()
    {
        int index=rand.nextInt(serverList.size());
```

```
        return serverList.get(index);
    }
}


//测试类
package wtf;
import com.wtf.CustomServer;
import com.wtf.LoadBalancer;
public class Test {
    public static void main(String[] args)
    {
        LoadBalancer balancer, balancer2, balancer3;
        balancer = LoadBalancer.getLoadBalancer();
        balancer2 = LoadBalancer.getLoadBalancer();
        balancer3 = LoadBalancer.getLoadBalancer();
        // 判断负载均衡器是否相同
        if (balancer == balancer2 && balancer == balancer3 && balancer2 ==
balancer3)
        {
            System.out.println("服务器负载均衡器是唯一的！");
        }
        // 增加服务器
        balancer.addServer(new CustomServer("Server 1") );
        balancer.addServer(new CustomServer("Server 2"));
        balancer.addServer(new CustomServer("Server 3"));
        balancer.addServer(new CustomServer("Server 4"));
        // 模拟客户端请求的分发
        for (int i = 0; i < 10; i++)
        {
            CustomServer server = balancer.GetServer();
            System.out.println("该请求已分配至 : " + server.Name);
        }
    }
}
```

43. 单例设计模式

注意：实现一个单例有两点注意事项，①将构造器私有，不允许外界通过构造器创建对象；②通过公开的静态方法向外界返回类的唯一实例。

```
//饿汉式单例
public class Singleton {
 private Singleton(){}
 private static Singleton instance = new Singleton();
 public static Singleton getInstance(){
 return instance;
 }
}
```

```
// 懒汉式单例
public class Singleton {
 private static Singleton instance = null;
 private Singleton() {}
 public static synchronized Singleton getInstance(){
 if (instance == null) instance = new Singleton();
 return instance;
 }
}
```

十一、多线程

44. 使用线程解决生产者消费者问题

思路：本题主要考查线程通信问题。wait() 和 notify() 的用法。

本例子使用 Staff 作为生产者和消费者共享的临界资源。Staff 容器中定义了放入馒头和取出馒头的控制。

如果生产者生产出馒头放入箩筐时，如果发现已经满了 则 wait() 进入等待阻塞状态。直到被其他线程（消费者）通过 notify() 来唤醒。当然，生产者在生产一个馒头并成功放入箩筐时，就可以通知消费者来消费了。

而消费者则和生产者正好相反的步骤，消费时，如果成功消费则通知生产者，如果发现没有了，则等待阻塞。直到被唤醒。

```
package com.demo3;
/**
 *      线程通信：
 *      生产者和消费者
 *
 * wait()和notify()必须放入synchronized代码段中
 * synchronized 修饰方法或代码段
 */
public class ProductorAndCustomer {

    public static void main(String[] args) {
        //实例化容器
        Staff staff = new Staff();

        //实例化 消费者
        Customer c = new Customer(staff);
        //实例化生产者
        Producer p = new Producer(staff);

        //启动线程
        p.start();
        c.start();
    }
```

```
}
/**
 * 消费者
 */
class Customer extends Thread{
    private Staff staff;
    public Customer(Staff staff){
        this.staff = staff;
    }
    @Override
    public void run() {
        for (int i = 0; i < 20; i++) {
            //取出馒头
            Matou matou = staff.pop();
            System.out.println("***消费**"+matou);
            try {
                Thread.sleep((int)(Math.random()*1000));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
/**
 * 生产者 线程类
 */
class Producer extends Thread{
    private Staff staff;

    public Producer(Staff staff) {
        this.staff = staff;
    }

    @Override
    public void run() {
        for (int i = 0; i < 20; i++) {
            //生产馒头
            Matou matou = new Matou(i+1);
            System.out.println("---生产--"+matou);
            //放入容器
            staff.push(matou);
            try {
                Thread.sleep((int)(Math.random()*800));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
/**
馒头类
*/
class Matou{
    private int id;
    public Matou(int id){
        this.id = id;
    }
    @Override
    public String toString() {
        return "Matou [id=" + id + "]";
    }
}
/**
 * 容器类
 */
class Staff{
    //下标 个数
    private int index;
    //存放馒头的数组容器
    private Matou[] staff = new Matou[5];

    /**
     * 放入馒头到容器
     */
    synchronized public void push(Matou matou){
        //先判断容器是否已满
        if(index==staff.length){
            //已满 则等待
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        //未满放入
        staff[index] = matou;
        index++;

        //通知消费者
        notify();
    }

    /**
     * 取出馒头方法    出栈
     */
    synchronized public Matou pop(){
        if(index == 0){
```

```
            //为空
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        //取出馒头
        index--;
        Matou matou = staff[index];

        //消费一个馒头之后，唤醒生产者
        notify();
        return matou;
    }
}
```

45. 模拟一个线程死锁

思路：出现死锁的四个必要条件：

1. 互斥条件：资源不能互相共享，各用各的。
2. 请求与保持条件：已经得到了资源的线程，还想要申请新的资源。
3. 非剥夺条件：已经分配的资源不能在相应的线程中强行的剥夺。
4. 循环等待条件：系统中若干个线程组成了环路，该环路中每一个线程都在等待着相邻的线程占据的资源。

所以需要满足以上条件，则需要我们在使用过程中使用 synchronized 关键字来加锁同步。并造成相互等待的状态。

```
package com.demo3;

/**
 * 死锁
 *
 */
public class TestLock {
    public static void main(String[] args) {
        Fork fork = new Fork();
        Knife nife = new Knife();

        Person person1 = new Person(fork, nife);
        Person person2 = new Person(nife, fork);
        person1.start();
        person2.start();
    }
}
/**
 * 顾客线程
```

```
 */
class Person extends Thread{
    private Tools tool1;
    private Tools tool2;
    public Person(Tools tool1,Tools tool2){
        this.tool1 = tool1;
        this.tool2 = tool2;
    }
    /**
     * 模拟使用刀子和叉子吃饭动作
     */
    @Override
    public void run() {
        synchronized (tool1) {
            tool1.use(this);
            try {
                Thread.sleep(3000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            synchronized (tool2) {
                tool2.use(this);
                try {
                    Thread.sleep(3000);//使用3秒后释放资源
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }

        }
        System.out.println(getName()+"===吃饭结束=");
    }
}

/**
 * 餐具接口
 */
interface Tools{
    public void use(Person p);
}
/**
 * 刀子
 */
class Knife implements Tools{
    public void use(Person cu){
        System.out.println("顾客"+cu.getName()+"使用刀子");
    }
}
/**
```

```
 * 叉子
 */
class Fork implements Tools{
    public void use(Person cu){
        System.out.println("顾客"+cu.getName()+"使用叉子");
    }
}
```

46. 多线程售票系统

模拟一个多线程的售票系统，多个售票员可以同时售票，并能保证出票数据的一致性

思路：在本题中要注意售票系统是只有一套，一个对象，而售票员是多个，可以同享售票系统。所以售票系统就是共享的临界资源。那么我们只要对售票这一个过程进行互斥操作，synchronized 同步售票方法就可以了。

```
package com.demo3;
/**
 * synchronized 同步关键字
 *     共享临界资源
 *
 */
public class TestSyn {
    public static void main(String[] args) {
        TicketSys ticketSys = new TicketSys();

        Seller seller1 = new Seller(ticketSys);
        Seller seller2 = new Seller(ticketSys);

        seller1.start();
        seller2.start();

    }
}
/**
 * 售票员
 *
 */
class Seller extends Thread{
    private TicketSys ticketSys;
    public Seller(TicketSys ticketSys){
        this.ticketSys = ticketSys;
    }
    @Override
    public void run() {
        for (int i = 0; i < 50; i++) {
            int ticketId = ticketSys.sale();//出售票
            if(ticketId==-1){
                System.out.println(getName()+"---票卖完了----");
```

```
            }else{
                System.out.println(getName()+"==========="+ticketId);
            }
            try {
//                Thread.sleep((int)(Math.random()*800));
                Thread.sleep(1);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
/**
 * 售票系统
 */
class TicketSys{
    private int count = 50;//总票数

    //售票
    synchronized public int sale(){
        if(count>0){//T2
            count--;//票数减去一张
            return 50-count;
        }
        return -1;//票已经售完
    }

    public int sale2(){
        synchronized ("") {
            if(count>0){//T2
                count--;//票数减去一张
                return 50-count;
            }
            return -1;//票已经售完
        }

    }
}
```

十二、人民币转换

47. 人民币金额转换

阿拉伯数字的金额转换成中国传统的形式如：￥1011 -> 一千零一拾一元整。

思路：人民币从阿拉伯数字转为大写中文，这就需要首先解决 0~9 到壹~玖之间的映射。这种映射，可以通过数组来解决即可。而还有要考虑的问题是，元拾佰仟的转换问题。这个可以结合数组及位数来实现。每多一位则从数组中依次往后取出单位拼接到结果字符串中。

```
package test;

public class RenMingBi {

    private static final char[] data = new char[]{
 '零','壹','贰','叁','肆','伍','陆','柒','捌','玖' };
    private static final char[] units = new char[]{
'元','拾','佰','仟','万','拾','佰','仟','亿'};
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println(
                convert(100000003));
    }

    public static String convert(int money)
    {
        StringBuffer sbf = new StringBuffer();
        int unit = 0;
        while(money!=0)
        {
            sbf.insert(0,units[unit++]);
            int number = money%10;
            sbf.insert(0, data[number]);
            money /= 10;
        }

        return sbf.toString().replaceAll("零[拾佰仟]","零").replaceAll("零
+万","万").replaceAll("零+元","元").replaceAll("零+","零");
    }
}
```

十三、字符串、正则表达式等其他问题

48. 判断身份证

要么是 15 位，要么是 18 位，最后一位可以为字母，并写程序提出其中的年月日。

思路：首先需要了解正则表达式，\d 表示数字。{n,m} 表示字符个数。可以使用正则表达式从第 6 位之后开始截取 8 位即可取出年月日。

```
package com.demo2;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexTest {

    public static void main(String[] args) {
```

```
        // 测试是否为合法的身份证号码
        String[] strs = { "130681198712092019", "13068119871209201x",
"13068119871209201", "123456789012345",
                "12345678901234x", "1234567890123" };
        Pattern p1 = Pattern.compile("(\\d{17}[0-9a-zA-Z]|\\d{14}[0-9a-zA-
Z])");
        for (int i = 0; i < strs.length; i++) {
            Matcher matcher = p1.matcher(strs[i]);
            System.out.println(strs[i] + ":" + matcher.matches());
        }

        Pattern p2 = Pattern.compile("\\d{6}(\\d{8}).*"); // 用于提取出生日字符串
        Pattern p3 = Pattern.compile("(\\d{4})(\\d{2})(\\d{2})");// 用于将生日字
符串进行分解为年月日
        for (int i = 0; i < strs.length; i++) {
            Matcher matcher = p2.matcher(strs[i]);
            boolean b = matcher.find();
            if (b) {
                String s = matcher.group(1);
                Matcher matcher2 = p3.matcher(s);
                if (matcher2.find()) {
                    System.out.println(
                            "生日为" + matcher2.group(1) + "年" +
matcher2.group(2) + "月" + matcher2.group(3) + "日");
                }
            }

        }

    }

}
```

49. 按字节截取字符串

编写一个截取字符串的函数，输入为一个字符串和字节数，输出为按字节截取的字符串，但要保证汉字不被截取半个，如“我 ABC”，4，应该截取“我 AB”。输入“我 ABC 汉DEF”，6，应该输出“我 ABC”，而不是“我 ABC + 汉的半个”。

思路：本题首先要注意一个字符中，汉字是张勇两个字节的。字母和数字为一个字节。而且如果我们把字符串转为 byte 数组时，会发现中文其实是两个负数来表示的。所以在做截取时，只要注意碰到负数则表示到汉字了。对于负数也必须得偶数个负数才能为一个完整的汉字。

```
public static void main(String[] args) throws Exception{
        String str = "我a爱中华abc我爱我的祖国def';
        String str = "我ABC汉";
        int num = trimGBK(str.getBytes("GBK"),5);
        System.out.println(str.substring(0,num) );
    }
```

```
    public static int  trimGBK(byte[] buf,int n){
        int num = 0;
        boolean bChineseFirstHalf = false;
        for(int i=0;i<n;i++)
        {
            if(buf[i]<0 && !bChineseFirstHalf){
                bChineseFirstHalf = true;
            }else{
                num++;
                bChineseFirstHalf = false;
            }
        }
        return num;
    }
```

50. 填字题目

给定一个四句古诗，程序实现从每一句中随机挖去一个字，并顺序打乱生成四个选项。从而随机生成一个填字题目。

思路：本题主要考查对字符串的掌握。可以根据每句古诗的长度来生成一个随机数，作为要挖去的字符。并保存到变量数组中。以便作为题目选项。然后把数组的保存的字符随机打乱，作为选项。

```
package com;
import java.util.Random;
import java.util.Scanner;

public class 填字游戏2 {

    public static void main(String[] args) {
        String content = "白日依山尽,黄河入海流,欲穷千里目,更上一层楼";
        String[] poems = content.split(",");//题目
        String[] options = new String[4];

        char[] answerChar = new char[4];;

        Random random = new Random();
        //挖去一个字
        for (int i = 0; i < poems.length; i++) {
            char ch = poems[i].charAt(random.nextInt(4));//
            poems[i] = poems[i].replace(ch, '?');//挖去的字替换成问号
            answerChar[i] =  ch; //挖去的为答案 保留起来
        }

        //生成正确选项 放入选项数组随机位置
        int rightPos = random.nextInt(4);
        options[rightPos] = new String(answerChar);
```

```
        for (int i = 0; i < options.length; i++) {
            if(i==rightPos){
                continue;
            }

            int s = random.nextInt(4);
            int e = random.nextInt(4);
            char temp = answerChar[s];
            answerChar[s] = answerChar[e];
            answerChar[e] = temp;

            boolean isRepeat = false;
            String opStr = new String(answerChar);
            //是否和已有的选项重复（已有中包含了正确的选项）
            for (int j = 0; j <options.length; j++) {
//                System.out.println(options[j]+"="+i+"="+opStr);
                if(options[j]!=null && options[j].equals(opStr)){
                    isRepeat = true;
                    break;
                }
            }
            if(isRepeat){
                i--;//条件变量减一 这样下次循环的i++等于没变化
//                System.out.println("===重复了：==="+Arrays.toString(options));
                continue;
            }
            options[i] = opStr; //如果没有重复，则次
        }


        System.out.println("题目如下：\n");
        //打印题目
        for (int i = 0; i < poems.length; i++) {
            System.out.println(poems[i]);
        }
        System.out.println("\n请从下面选项中选出正确答案：");
        for (int i = 0; i < options.length; i++) {
            System.out.print((char)('A'+i)+":");
            System.out.print(options[i].charAt(0)+",");
            System.out.print(options[i].charAt(1)+",");
            System.out.print(options[i].charAt(2)+",");
            System.out.println(options[i].charAt(3));
        }
        System.out.println("您选择的答案是：");
        Scanner sc = new Scanner(System.in);
        char input = sc.next().charAt(0);
        if ((input-'A')==rightPos) {
            System.out.println("恭喜您答对了！");
        } else {
```

```
            System.out.println("对不起，答错了！");
            System.out.println("正确答案是"  + (char)('A' + rightPos));
        }

        sc.close();

    }
}
```

51. 统计一篇英文文章中单词个数

思路：本题主要是用到 IO 的字符流来读取字符文件，并根据空格，换行，制表符等进行判断单词存在的数量。读取字符逐一分析间隔符号，然后统计结果。

```
import java.io.FileReader;
public class WordCounting {
 public static void main(String[] args) {
 try(FileReader fr = new FileReader("a.txt")) {
 int counter = 0;
 boolean state = false;
 int currentChar;
 while((currentChar= fr.read()) != -1) {
 if(currentChar== ' ' || currentChar == '\n'
 || currentChar == '\t' || currentChar == '\r') {
 state = false;
 }
 else if(!state) {
 state = true;
 counter++;
 }
 }
 System.out.println(counter);
 }
 catch(Exceptione) {
 e.printStackTrace();
 }
 }
}
```

53. 去重排序

给定任意长度的字符串，字符串中可以有任意顺序和任意个数的任意字符。要求去除重复只留下唯一的字符并升序排序。比如：aabbxxcaaccyyxxxyydddd 输出结果为：abcdxy。

思路：本题关键点是两个词——唯一和排序，自然要想到 TreeSet。

所以我们把所有字符放入TreeSet 则可以自动去除重复及排序了。然后再取出来组合起来就好。

```
package com.wdzl.demo;
```

```
import java.util.TreeSet;

public class Test {

    public static void main(String[] args) {
        String str = "aabbxxcaaccyyxxxyydddd";
        TreeSet<Character> ts = new TreeSet<>();
        for (int i = 0; i < str.length(); i++) {
            ts.add(str.charAt(i));
        }
        str = "";
        for (Character character : ts) {
            str += character;
        }
        System.out.println(str);
    }

}
```

54. 全排列

给出五个数字 12345 的所有排列。

思路：本题主要使用分区，分别递归处理。

```
public class FullPermutation {
 public static void perm(int[] list) {
 perm(list,0);
 }
 private static void perm(int[] list, int k) {
 if (k == list.length) {
 for (int i = 0; i < list.length; i++) {
 System.out.print(list[i]);
 }
 System.out.println();
 }else{
 for (int i = k; i < list.length; i++) {
 swap(list, k, i);
 perm(list, k + 1);
 swap(list, k, i);
 }
 }
 }
 private static void swap(int[] list, int pos1, int pos2) {
 int temp = list[pos1];
 list[pos1] = list[pos2];
 list[pos2] = temp;
 }
```

```
 public static void main(String[] args) {
 int[] x = {1, 2, 3, 4, 5};
 perm(x);
 }
}
```

55. 回文素数

所谓回文数就是顺着读和倒着读一样的数（例如：11，121，1991……），回文素数就是既是回文数又是素数（只能被 1 和自身整除的数）的数。编程找出 11~9999 之间的回文素数。

思路：本题主要有两个关键点：判断是否为素数。判断是否为回文。素数判断可以单独方法来完成，整除从 2~本身-1， 如果全部除不尽，则为素数。回文数也是定义一个方法，分别取出首尾两个数，比较是否相等即可。

```
public class PalindromicPrimeNumber {
 public static void main(String[] args) {
 for(int i = 11; i <= 9999; i++) {
 if(isPrime(i) && isPalindromic(i)) {
 System.out.println(i);
 }
 }
 }

 public static boolean isPrime(int n) {
 for(int i = 2; i <= Math.sqrt(n); i++) {
 if(n % i == 0) {
 return false;
 }
 }
 return true;
 }
 public static boolean isPalindromic(int n) {
 int temp = n;
 int sum = 0;
 while(temp > 0) {
 sum= sum * 10 + temp % 10;
 temp/= 10;
 }
 return sum == n;
 }
}
```

56. 给一个任意的字符串，如果统计出每个字符出现的次数

例如 aabbxxcaaccyyxxxyydddd，统计出结果为：{a=4, b=2, c=3, d=4, x=5, y=4}。

思路：本题只需要逐一取出每个字符，放入 Map，Map 是键值对结构。在放入时，我们先判断是否已经放过了，如果没有放过则初始为 1。如果存在，则取出加一，再放回。

```
package com.wdzl.demo;

import java.util.HashMap;
import java.util.TreeSet;

public class Test {

    public static void main(String[] args) {
        String str = "aabbxxcaaccyyxxxyydddd";
        HashMap<Character, Integer> s = new HashMap<>();
        for (int i = 0; i < str.length(); i++) {
            Character c = str.charAt(i);
            Integer count = s.get(c);
            if(count==null){
                count = 0;
            }
            count++;
            s.put(c, count);
        }
        System.out.println(s);
    }

}
```

57. 有一串无序的任意序列字符串，包含了一些特殊字符，现在需要去除特殊字符，只留下数字字母下划线。

例如 aabb23xx&caaccy#yxx@xyyd5_d4dd”处理后需要留下结果
为：“aabb23xxcaaccyyxxxyyd5_d4dd”。

思路：如果使用循环判断字符类型也可以处理，但是太麻烦，所以使用正则表达式会很方便。\W 在正则表达式中代表的是非数字字母下划线的字符。这样可以使用此表达式替换成空字符。

```
public class Test {

    public static void main(String[] args) {
        String str = "aabb23xx&caaccy#yxx@xyyd5_d4dd";
        str = str.replaceAll("[\\W]", "");
        System.out.println(str);
    }

}
```

58. 写一个算法判断一个英文单词的所有字母是否全都不同（不区分大小写）

思路：循环逐一取出每个字母，并放入数组中，如果后面取出的字符在数组中存在则直接为假。

如果整个循环结束后，还是没有重复的在数组中。则认为是没有重复的。

```
public class AllNotTheSame {
 public static boolean judge(String str) {
 String temp = str.toLowerCase();
 int[] letterCounter = new int[26];
 for(int i = 0; i <temp.length(); i++) {
 int index = temp.charAt(i)- 'a';
 letterCounter[index]++;
 if(letterCounter[index] > 1) {
 return false;
 }
 }
 return true;
 }
 public static void main(String[] args) {
 System.out.println(judge("hello"));
 System.out.print(judge("smile"));
 }
}
```

59. 一个有 n 级的台阶，一次可以走 1 级、2 级或 3 级，问走完 n 级台阶有多少种走法

思路：本题使用递归可以很方便求出结果。台阶级数任意，但是走法只有三种，所以每次递归计算三种执行次数就可得出结果。

```
//可以通过递归求解。
public class GoSteps {
 public static int countWays(int n) {
 if(n < 0) {
 return 0;
 }
 else if(n == 0) {
 return 1;
 }
 else {
 return countWays(n - 1) + countWays(n - 2) + countWays(n -3);
 }
 }
 public static void main(String[] args) {
 System.out.println(countWays(5)); // 13
 }
}
```

60. 编写一个程序，统计文本文件中内容的总行数

思路：这是一个 IO 流问题，而且是一个字符流，读取行数，BufferedReader 可以一次读取一行，所以直接使用这个流循环读取到文件末尾就可计算出总行数。

```java
package com.wdzl.demo;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Test {

    public static void main(String[] args) {
        try {
            FileReader fw = new FileReader("ResizeImage.java");
            BufferedReader br = new BufferedReader(fw);
            int count = 0;
             //一次读取一行
            while(br.readLine()!=null){
                count++;
            }
            System.out.println("总行数: "+count);
            br.close();
            fw.close();
        } catch (IOException e) {

            e.printStackTrace();
        }
    }

}
```