Java面试题总结：基础及语法169道

收集整理:秦疆 联系方式QQ:24736743 微信:qinlvejiang

答案来源收集与互联网,部分内容经供参考,代码全部为手写验证通过.

1~20

1. Java跨平台原理（字节码文件、虚拟机）

C/C++语言都直接编译成针对特定平台机器码。如果要跨平台，需要使用相应的编译器重新编译。

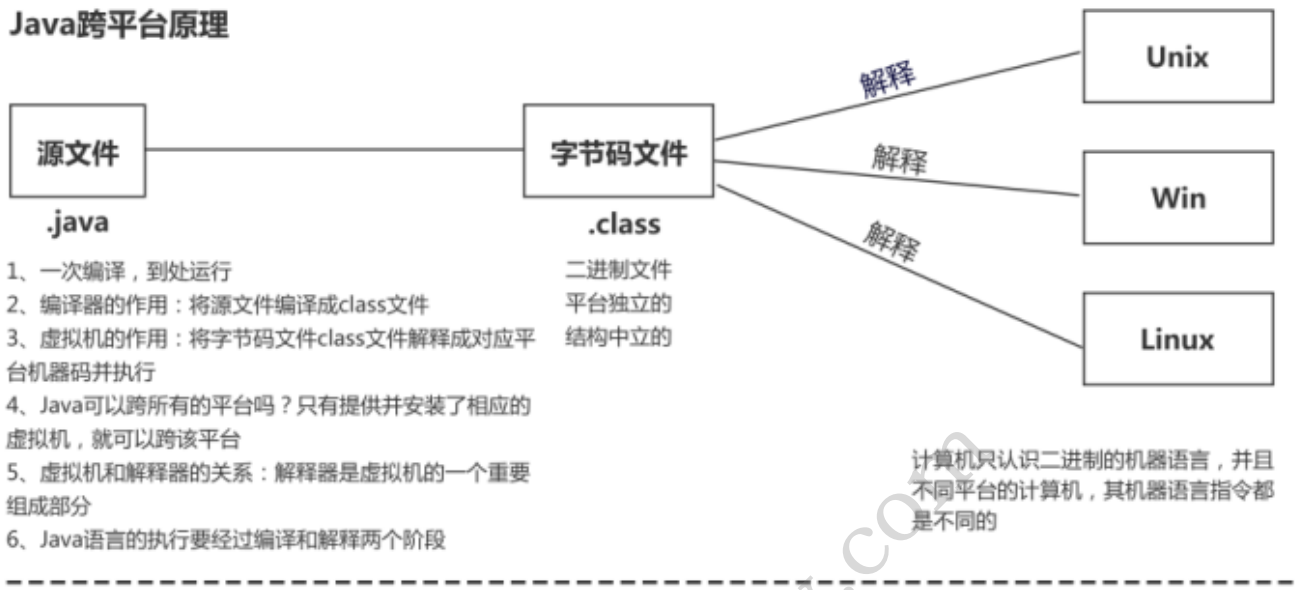
Java源程序（.java）要先编译成与平台无关的字节码文件(.class)，然后字节码文件再解释成机器码运行。解释是通过Java虚拟机来执行的。

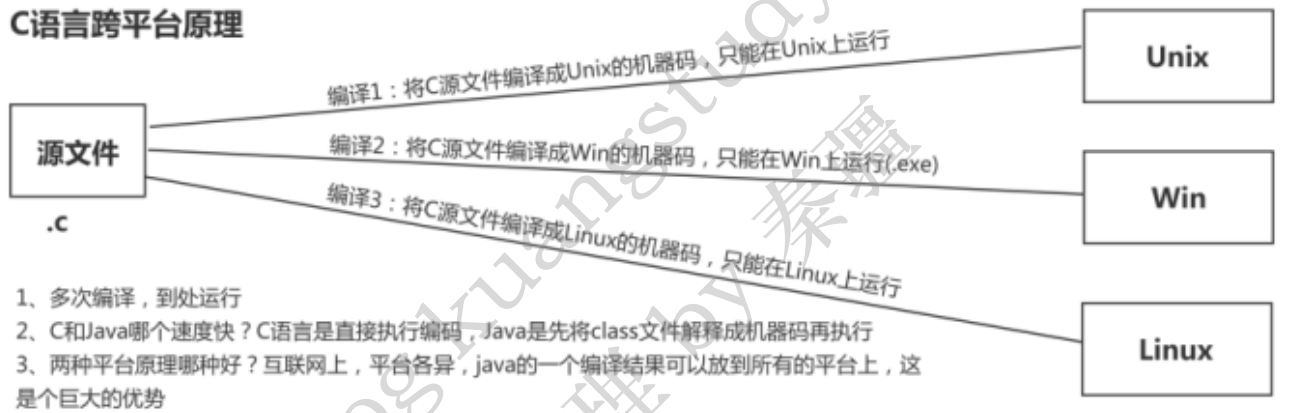
字节码文件不面向任何具体平台，只面向虚拟机。

Java虚拟机是可运行Java字节码文件的虚拟计算机。不同平台的虚拟机是不同的，但它们都提供了相同的接口。

Java语言具有一次编译，到处运行的特点。就是说编译后的.class可以跨平台运行，前提是该平台具有相应的Java虚拟机。但是性能比C/C++要低。

Java的跨平台原理决定了其性能没有C/C++高





2. Java的安全性

语言层次的安全性主要体现在：

Java取消了强大但又危险的指针，而代之以引用。由于指针可进行移动运算，指针可随便指向一个内存区域，而不管这个区域是否可用，这样做是危险的，因为原来这个内存地址可能存储着重要数据或者是其他程序运行所占用的，并且使用指针也容易数组越界。

垃圾回收机制：

不需要程序员直接控制内存回收，由垃圾回收器在后台自动回收不再使用的内存。避免程序忘记及时回收，导致内存泄露。避免程序错误回收程序核心类库的内存，导致系统崩溃。

异常处理机制：

Java异常机制主要依赖于try、catch、finally、throw、throws五个关键字。

强制类型转换：

只有在满足强制转换规则的情况下才能强转成功。

底层的安全性可以从以下方面来说明:

Java在字节码的传输过程中使用了公开密钥加密机制(PKC)。

在运行环境提供了四级安全性保障机制：

字节码校验器 -类装载器 -运行时内存布局 -文件访问限制

---

3. Java三大版本

Java2平台包括标准版（J2SE）、企业版（J2EE）和微缩版（J2ME）三个版本：

Standard Edition(标准版) J2SE 包含那些构成Java语言核心的类。

比如：数据库连接、接口定义、输入/输出、网络编程

Enterprise Edition(企业版) J2EE 包含J2SE 中的类，并且还包含用于开发企业级应用的类。

比如servlet、JSP、XML、事务控制

Micro Edition(微缩版) J2ME 包含J2SE中一部分类，用于消费类电子产品的软件开发。

比如：呼机、智能卡、手机、PDA、机顶盒

他们的范围是：

J2SE包含于J2EE中，J2ME包含了J2SE的核心类，但新添加了一些专有类应用场合，API的覆盖范围各不相同。

---

4. 什么是JVM？什么是JDK？ 什么是JRE？

JVM：

JVM是*Java Virtual Machine*（Java虚拟机）的缩写，它是整个Java实现跨平台的最核心的部分，所有的java程序会首先被编译为.class的类文件，这种类文件可以在虚拟机上执行，也就是说class并不直接与机器的操作系统相对应，而是经过虚拟机间接与操作系统交互，由虚拟机将程序解释给本地系统执行。JVM是Java平台的基础，和实际的机器一样，它也有自己的指令集，并且在运行时操作不同的内存区域。 JVM通过抽象操作系统和CPU结构，提供了一种与平台无关的代码执行方法，即与特殊的实现方法、主机硬件、主机操作系统无关。JVM的主要工作是解释自己的指令集（即字节码）到CPU的指令集或对应的系统调用，保护用户免被恶意程序骚扰。 JVM对上层的Java源文件是不关心的，它关注的只是由源文件生成的类文件（.class文件）。

JRE：

JRE是*java runtime environment*（java运行环境）的缩写。光有JVM还不能让class文件执行，因为在解释class的时候JVM需要调用解释所需要的类库lib。在JDK的安装目录里你可以找到jre目录，里面有两个文件夹bin和lib,在这里可以认为bin里的就是jvm，lib中则是jvm工作所需要的类库，而jvm和lib和起来就称为jre。所以，在你写完java程序编译成.class之后，你可以把这个.class文件和jre一起打包发给朋友，这样你的朋友就可以运行你写程序了（jre里有运行.class的java.exe）。JRE是Sun公司发布的一个更大的系统，它里面就有一个JVM。JRE就与具体的CPU结构和操作系统有关，是运行Java程序必不可少的（除非用其他一些编译环境编译成.exe可执行文件.....），JRE的地位就象一台PC机一样，我们写好的Win32应用程序需要操作系统帮我们运行，同样的，我们编写的Java程序也必须要JRE才能运行。

JDK：

JDK是*java development kit*（java开发工具包）的缩写。每个学java的人都会先在机器上装一个JDK，那 让我们看一下JDK的安装目录。在目录下面有六个文件夹、一个src类库源码压缩包、和其他几个声明文件。其中，真正在运行java时起作用的是以下四个文件夹：bin、include、lib、jre。现在我们可以看出这样一个关系，JDK包含JRE，而JRE包含JVM。

bin: 最主要的是编译器(javac.exe)

include: java和JVM交互用的头文件

lib：类库

jre: java运行环境

（注意：这里的bin、lib文件夹和jre里的bin、lib是不同的）

总的来说JDK是用于java程序的开发,而jre则是只能运行class而没有编译的功能。eclipse、idea等其他IDE有自己的编译器而不是用JDK bin目录中自带的，所以在安装时你会发现他们只要求你选jre路径就ok了。

JDK,JRE,JVM三者关系概括如下：

jdk是JAVA程序开发时用的开发工具包，其内部也有JRE运行环境JRE。JRE是JAVA程序运行时需要的运行环境，就是说如果你光是运行JAVA程序而不是去搞开发的话，只安装JRE就能运行已经存在的JAVA程序了。JDk、JRE内部都包含JAVA虚拟机JVM，JAVA虚拟机内部包含许多应用程序的类的解释器和类加载器等等。

---

5. Java三种注释类型

共有单行注释、多行注释、文档注释3种注释类型。

单行注释，采用“//”方式.只能注释一行代码。如：

```
//类成员变量
```

多行注释，采用“/*...*/”方式，可注释多行代码，其中不允许出现嵌套。如：

```
/*
System.out.println("a");
System.out.println("b");
System.out.println("c");
*/
```

文档注释，采用“/...*/”方式，如：**

```
/**
* @author 狂神说
* @descripted QQ:24736743  wechat:qinl0vejiang
*/
```

---

6. 八种基本数据类型及其字节数

数据类型		关键字	字节数
数值型	整数型	byte	1
		short	2
		int	4
		long	8
	浮点型	float	4
		double	8
布尔型		boolean	1（位）
字符型		char	2

7. i++和++i的异同之处

共同点：

1、i++和++i都是变量自增1，都等价于i=i+1

2、如果i++,++i是一条单独的语句，两者没有任何区别

3、i++和++i的使用仅仅针对变量。 5++和++5会报错，因为5不是变量。

不同点：

如果i++,++i不是一条单独的语句，他们就有区别

i++ ：先运算后增1。如：

```
1 `int` `x=``5``;``int` `y=x++;``System.out.println(``"x="``+x+``", y="``+y);`` ``//以上代码运行后输出结果为：x=6, y=5
```

++i ： 先增1后运算。如：

```
1 `int` `x=``5``;``int` `y=++x;``System.out.println(``"x="``+x+``", y="``+y);`` ``//以上代码运行后输出结果为：x=6, y=6`
```

8. &和&&，|和|| 的区别和联系

&和&&的联系(共同点)：

&和&&都可以用作逻辑与运算符，但是要看使用时的具体条件来决定。

操作数1&操作数2，操作数1&&操作数2，

表达式1&表达式2，表达式1&&表达式2，

情况1：当上述的操作数是boolean类型变量时，&和&&都可以用作逻辑与运算符。

情况2：当上述的表达式结果是boolean类型变量时，&和&&都可以用作逻辑与运算符。表示逻辑与(and)，当运算符两边的表达式的结果或操作数都为true时，整个运算结果才为true，否则，只要有一方为false，结果都为false。

&和&&的区别(不同点)：

1. & 称为逻辑与运算符，&& 称为短路与运算符，也可叫逻辑与运算符。

对于&：无论任何情况，&两边的操作数或表达式都会参与计算。

对于&&：当&&左边的操作数为false或左边表达式结果为false时，&&右边的操作数或表达式将不参与计算，此时最终结果都为false。

综上所述，如果逻辑与运算的第一个操作数是false或第一个表达式的结果为false时，对于第二个操作数或表达式是否进行运算，对最终的结果没有影响，结果肯定是false。推介平时多使用&&，因为它效率更高些。

2. &还可以用作位运算符。当&两边操作数或两边表达式的结果不是boolean类型时，&用于按位与运算符的操作。

|和||的区别和联系与&和&&的区别和联系类似

---

9. 用最有效率的方法算出2乘以8等于多少

使用位运算来实现效率最高。位运算符是对操作数以二进制比特位为单位进行操作和运算，操作数和结果都是整型数。

对于位运算符“<<”, 是将一个数左移n位，就相当于乘以了2的n次方，那么，一个数乘以8只要将其左移3位即可，位运算cpu直接支持的，效率最高。所以，2乘以8等于几的最效率的方法是2 << 3

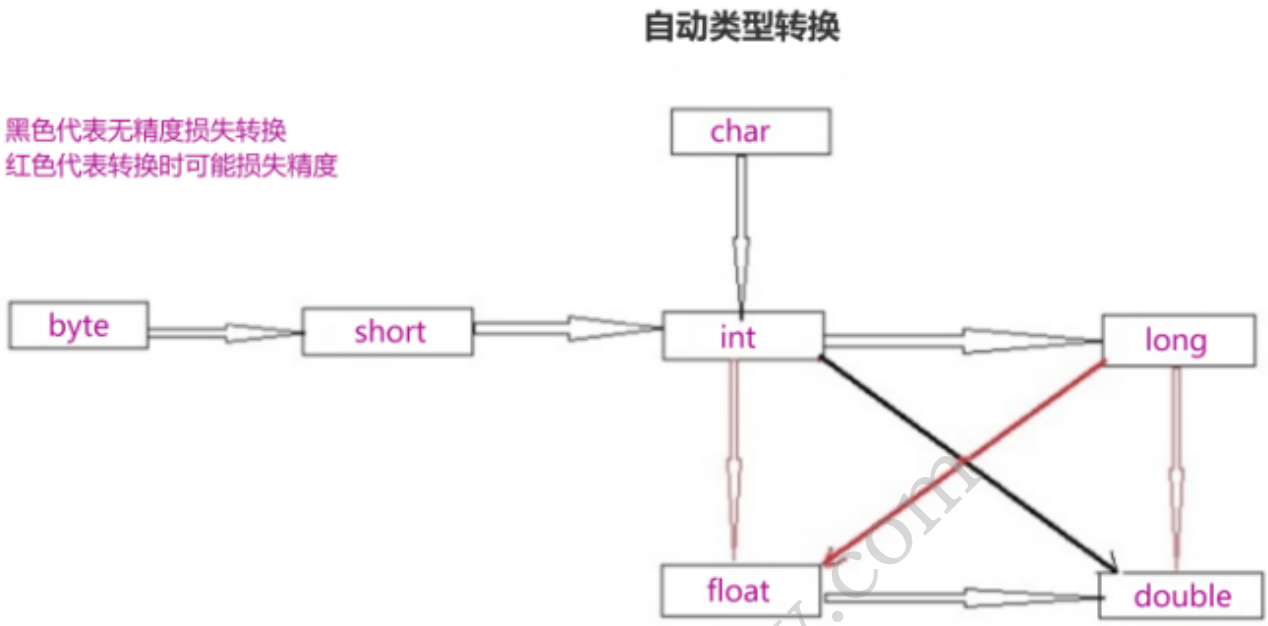
---

10. 基本数据类型的类型转换规则

基本类型转换分为自动转换和强制转换。

自动转换规则：

容量小的数据类型 可以自动转换成容量大的数据类型，也可以说低级自动向高级转换。这儿的容量指的不是字节数，而是指类型表述的范围。



强制转换规则：

高级变为低级需要强制转换。

如何转换：

（1）、赋值运算符"="右边的转换，先自动转换成表达式中级别最高的数据类型，再进行运算。

（2）、赋值运算符"="两侧的转换，若左边级别>右边级别，会自动转换；若左边级别 == 右边级别，不用转换；若左边级别 < 右边级别，需强制转换。

（3）、可以将整型常量直接赋值给byte, short, char等类型变量，而不需要进行强制类型转换，前提是不超出其表述范围，否则必须进行强制转换。

---

11. if和switch的异同之处

相同之处：

都是分支语句，多超过一种的情况进行判断处理。

不同之处：

switch更适合用于多分支情况，就是有很多种情况需要判断处理，判断条件类型单一，只有一个入口，在分支执行完后（如果没有break跳出），不加判断地执行下去; 而if—elseif---else多分枝主要适用于分支较少的分支结构，判断类型不是单一，只要一个分支被执行后，后边的分支不再执行。

switch为等值判断（不允许比如>= <=），而if为等值和区间都可以，if的使用范围大。

---

12. while和do-while循环的区别

while先判断后执行，第一次判断为false，循环体一次都不执行。

do while先执行 后判断，最少执行1次。

如果while循环第一次判断为true, 则两种循环没有区别。

13. break和continue的作用

break: 结束当前循环并退出当前循环体。break还可以退出switch语句

continue: 循环体中后续的语句不执行，但是循环没有结束，继续进行循环条件的判断（for循环还会i++）。continue只是结束本次循环。

14. 请使用递归算法计算n！

Recursion 读法:[rɪˈkɜːrʃn] 递归

一个正整数的阶乘（factorial）是所有小于及等于该数的正整数的积，并且0的阶乘为1。 自然数n的阶乘写作n!。1808年，基斯顿·卡曼引进这个表示法。 亦即n!=1×2×3×...×n。阶乘亦可以递归方式定义：0!=1，n!=(n-1)!×n。

```
/*
 * n阶乘
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class Recursion {

    //阶乘算法
    public int factorial(int num){
        if (num==0 || num== 1){
            return 1;
        }else{
            return num*factorial(num-1);
        }
    }

    public static void main(String[] args) {
        Recursion recursion = new Recursion();
        int n = 0;
        int factorial = recursion.factorial(n);
        System.out.println(factorial);
    }

}
```


15. 递归的定义和优缺点

递归算法是一种直接或者间接地调用自身算法的过程。在计算机编写程序中，递归算法对解决一大类问题是十分有效的，它往往使算法的描述简洁而且易于理解。

递归算法解决问题的特点：

(1) 递归就是在过程或函数里调用自身。

(2) 在使用递归策略时，必须有一个明确的递归结束条件，称为递归出口。

(3) 递归算法解题通常显得很简洁，但运行效率较低。所以一般不提倡用递归算法设计程序。

(4) 在递归调用的过程当中系统为每一层的返回点、局部量等开辟了栈来存储。递归次数过多容易造成栈溢出等。所以一般不提倡用递归算法设计程序。

---

16. 数组的特征

数组是（相同类型数据）的（有序）（集合）

数组会在内存中开辟一块连续的空间，每个空间相当于之前的一个变量，称为数组的元素element

元素的表示 数组名[下标或者索引] `scores[7] scores[0] scores[9]`

索引从0开始

每个数组元素有默认值 `double 0.0; boolean false; int 0`

数组元素有序的，不是大小顺序，是索引的顺序

数组中可以存储基本数据类型，可以存储引用数据类型；但是对于一个数组而言，数组的类型是固定的，只能是一个

length:数组的长度

数组的长度是固定的，一经定义，不能再发生变化（数组的扩容）

---

17. 请写出冒泡排序代码

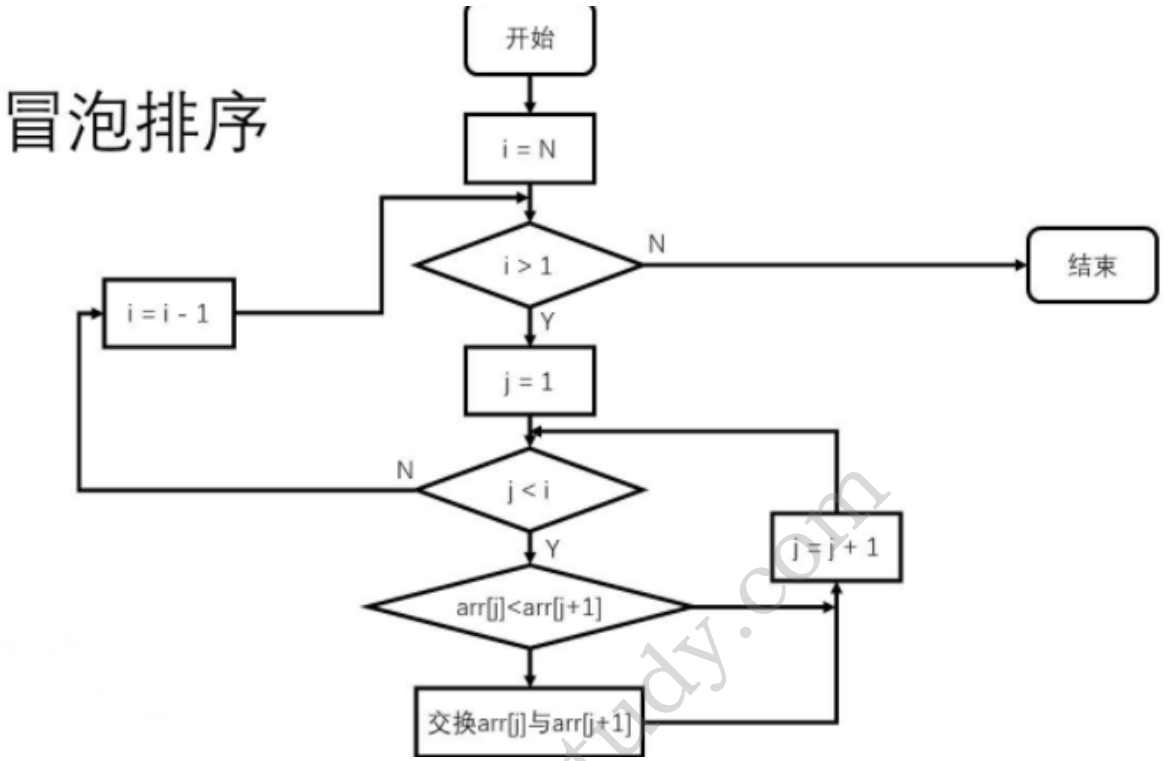
冒泡排序（Bubble Sort），是一种计算机科学领域的较简单的排序算法。

它重复地走访过要排序的元素列，依次比较两个相邻的元素，如果他们的顺序（如从大到小、首字母从A到Z）错误就把他们交换过来。走访元素的工作是重复地进行直到没有相邻元素需要交换，也就是说该元素列已经排序完成。

这个算法的名字由来是因为越大的元素会经由交换慢慢"浮"到数列的顶端（升序或降序排列），就如同碳酸饮料中二氧化碳的气泡最终会上浮到顶端一样，故名"冒泡排序"。

冒泡排序算法的原理如下：

1. 比较相邻的元素。如果第一个比第二个大，就交换他们两个。
2. 对每一对相邻元素做同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。
3. 针对所有的元素重复以上的步骤，除了最后一个。
4. 持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。




```java
/*
 * 冒泡排序
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class Bubble {

    public int[] sort(int[] array) {
        int temp = 0;
       // 外层循环，它决定一共走几趟  //-1为了防止溢出
        for (int i = 0; i < array.length - 1; i++) {
            int flag = 0; //通过符号位可以减少无谓的比较，如果已经有序了，就退出循环
            //内层循环，它决定每趟走一次
            for (int j = 0; j < array.length - i - 1; j++) {
                //如果后一个大于前一个,则换位
                if (array[j + 1] > array[j]) {
                    temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                    flag = 1;
                }
            }
            if(flag == 0){
                break;
            }
        }
        return array;
    }

    public static void main(String[] args) {
        Bubble bubble = new Bubble();
```

```
        int[] array = {2,5,1,6,4,9,8,5,3,1,2,0};
        int[] sort = bubble.sort(array);
        for (int num:sort){
            System.out.print(num+"\t");
        }
    }
}
```

18. 请写出选择排序的代码

选择排序（Selection sort）是一种简单直观的排序算法。它的工作原理是每一次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。以此类推，直到全部待排序的数据元素排完。 选择排序是不稳定的排序方法。

排序方法	最好时间	平均时间	最坏时间	辅助空间	稳定性
直接插入	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
二分插入	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
希 尔		$O(n^{1.25})$		O(1)	不稳定
冒 泡	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
快 速	O(nlgn)	O(nlgn)	$O(n^2)$	O(lgn)	不稳定
直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	不稳定
堆	O(nlgn)	O(nlgn)	O(nlgn)		不稳定
归 并	O(nlgn)	O(nlgn)	O(nlgn)	O(n)	稳定
基 数	O(d(rd+n))	O(d(rd+n))	O(d(rd+n))	O(rd+n)	稳定


```
/*
 * 选择排序
 * @Description  QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class SelectSort {

    public int[] sort(int arr[]) {
        int temp = 0;
        for (int i = 0; i < arr.length - 1; i++) {
            // 认为目前的数就是最小的，记录最小数的下标
            int minIndex = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[minIndex] > arr[j]) {
                    // 修改最小值的下标
                    minIndex = j;
                }
            }
            // 当退出for就找到这次的最小值,就需要交换位置了
            if (i != minIndex) {
```

```
                //交换当前值和找到的最小值的位置
                temp = arr[i];
                arr[i] = arr[minIndex];
                arr[minIndex] = temp;
            }
        }
        return arr;
    }

    public static void main(String[] args) {
        SelectSort selectSort = new SelectSort();
        int[] array = {2,5,1,6,4,9,8,5,3,1,2,0};
        int[] sort = selectSort.sort(array);
        for (int num:sort){
            System.out.print(num+"\t");
        }
    }
}
```

19. 请写出插入排序的代码

有一个已经有序的数据序列，要求在这个已经排好的数据序列中插入一个数，但要求插入后此数据序列仍然有序，这个时候就要用到一种新的排序方法——插入排序法,

插入排序的基本操作就是将一个数据插入到已经排好序的有序数据中，从而得到一个新的、个数加一的有序数据，算法适用于少量数据的排序，时间复杂度为O(n^2)。是稳定的排序方法。插入算法把要排序的数组分成两部分：第一部分包含了这个数组的所有元素，但将最后一个元素除外（让数组多一个空间才有插入的位置），而第二部分就只包含这一个元素（即待插入元素）。在第一部分排序完成后，再将这个最后元素插入到已排好序的第一部分中。

插入排序的基本思想是：每步将一个待排序的记录，按其关键码值的大小插入前面已经排序的文件中适当位置上，直到全部插入完为止。

包括：直接插入排序，二分插入排序（又称折半插入排序），链表插入排序，希尔排序（又称缩小增量排序）。属于稳定排序的一种（通俗地讲，就是两个相等的数不会交换位置）。

一般来说，插入排序都采用in-place在数组上实现。具体算法描述如下： 1. 从第一个元素开始，该元素可以认为已经被排序 2. 取出下一个元素，在已经排序的元素序列中从后向前扫描 3. 如果该元素（已排序）大于新元素，将该元素移到下一位置 4. 重复步骤3，直到找到已排序的元素小于或者等于新元素的位置 5. 将新元素插入到下一位置中 6. 重复步骤2~5

如果比较操作的代价比交换操作大的话，可以采用二分查找法来减少比较操作的数目。该算法可以认为是插入排序的一个变种，称为二分查找排序。

```
/*
 * 插入排序
 * @Description  QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class InsertSort {

    private int[] sort(int[]arr){
        //如果传入的数组为空或者只有一个值,就直接返回
```

```
        if(arr == null || arr.length < 2){
            return arr;
        }
        //不为空则进循环判断
        //外层循环控制总数量
        for(int i=1;i<arr.length;i++){
            //内层循环依次减少并提出结果
            for(int j=i;j>0;j--){
                //如果当前数字小于前一个,则交换,否则不变
                if(arr[j]<arr[j-1]){
                    int temp=arr[j];
                    arr[j]=arr[j-1];
                    arr[j-1]=temp;
                }else{
                    break;
                }
            }
        }
        return arr;
    }

    public static void main(String[] args) {
        InsertSort insertSort = new InsertSort();
        int[] array = {2,5,1,6,4,9,8,5,3,1,2,0};
        int[] sort = insertSort.sort(array);
        for (int num:sort){
            System.out.print(num+"\t");
        }
    }
}
```


20. 可变参数的作用和特点

详解 : https://www.cnblogs.com/uptownBoy/articles/1698335.html

总结1：可变参数

1.可变参数的形式 ...

2.可变参数只能是方法的形参

3.可变参数对应的实参可以0,1,2.....n个，也可以是一个数组

4.在可变参数的方法中，将可变参数当做数组来处理

5.可变参数最多有一个，只能是最后一个

6.可变参数好处：方便 简单 减少重载方法的数量

7.如果定义了可变参数的方法，不允许同时定义相同类型数组参数的方法

总结2： 数组做形参和可变参数做形参联系和区别

联系：

1.实参都可以是数组；

2.方法体中，可变参数本质就是当做数组来处理

区别：

1.个数不同 可变参数只能有一个数组参数可以多个

2.位置不同 可变参数只能是最后一个 数组参数位置任意

3.实参不同 可变参数实参可以0,1,2.....个，也可以是一个数组，数组的实参只能是数组

```
/*
 * @Description  QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
private static int sum(int... values) {
    int sum = 0;
    for (int i = 0; i < values.length; i++) {
        sum += values[i];
    }
    return sum;
}
```

21~40

21. 类和对象的关系

类是对象的抽象，而对象是类的具体实例。 类是抽象的，不占用内存，而对象是具体的，占用存储空间。 类是用于创建对象的蓝图，类就是一个带方法和变量的特定类型

类和对象好比图纸和实物的关系，模具和铸件的关系。

比如人类就是一个概念，人类具有身高，体重等属性。人类可以做吃饭、说话等方法。

小明就是一个具体的人，也就是实例，他的属性是具体的身高200cm，体重180kg，他做的方法是具体的吃了一碗白米饭，说了“12345”这样一句话。

还可以有小红,小蓝等具体的人即对象,但他们都是人类的实例,即类.

---

22. 面向过程和面向对象的区别

两者都是软件开发思想，先有面向过程，后有面向对象。在大型项目中，针对面向过程的不足推出了面向对象开发思想。

	面向过程	面向对象
区别	事物比较简单，可以用线性的思维去解决	事物比较复杂，使用简单的线性思维无法解决
共同点	面向过程和面向对象都是解决实际问题的一种思维方式	
	二者相辅相成，并不是对立的。 解决复杂问题，通过面向对象方式便于我们从宏观上把握事物之间复杂的关系、方便我们分析整个系统;具体到微观操作，仍然使用面向过程方式来处理	

比喻:

面向过程是蛋炒饭，面向对象是盖浇饭。盖浇饭的好处就是“菜”“饭”分离，从而提高了制作盖浇饭的灵活性。饭不满意就换饭，菜不满意换菜。用软件工程的专业术语就是“可维护性”比较好，“饭” 和“菜”的耦合度比较低。蛋炒饭就不行了, 如果不满意就要丢掉重做;

区别:

编程思路不同： 面向过程以实现功能的函数开发为主，而面向对象要首先抽象出类、属性及其方法，然后通过实例化类、执行方法来完成功能。

封装性：都具有封装性，但是面向过程是封装的是功能，而面向对象封装的是数据和功能。

面向对象具有继承性和多态性，而面向过程没有继承性和多态性，所以面向对象优势是明显。

方法重载和方法重写（覆盖）的区别:

	英文	位置不同	作用不同
重载	overload	同一个类中	在一个类里面为一种行为提供多种实现方式并提高可读性
重写	override	子类和父类间	父类方法无法满足子类的要求，子类通过方法重写满足要求

	修饰符	返回值	方法名	参数	抛出异常
重载	无关	无关	相同	不同	无关
重写	大于等于	小于等于	相同	相同	小于等于

23. this和super关键字的作用

this是对象内部指代自身的引用,同时也是解决成员变量和局部变量同名问题；**this可以调用成员变量，不能调用局部变量**；this也可以调用成员方法，但是在普通方法中可以省略this，在构造方法中不允许省略，必须是构造方法的第一条语句。而且在静态方法当中不允许出现this关键字。

super代表对当前对象的直接父类对象的引用，super可以调用直接父类的成员变量（注意权限修饰符的影响，比如不能访问private成员）

super可以调用直接父类的成员方法（注意权限修饰符的影响，比如不能访问private成员）；super可以调用直接父类的构造方法，只限构造方法中使用，且必须是第一条语句。

24. static关键字的作用

static可以修饰变量、方法、代码块和内部类

static属性属于这个类所有，即由该类创建的所有对象共享同一个static属性。可以对象创建后通过对象名.属性名和类名.属性名两种方式来访问。也可以在没有创建任何对象之前通过类名.属性名的方式来访问。

.static变量和非static变量的区别(都是成员变量，不是局部变量)

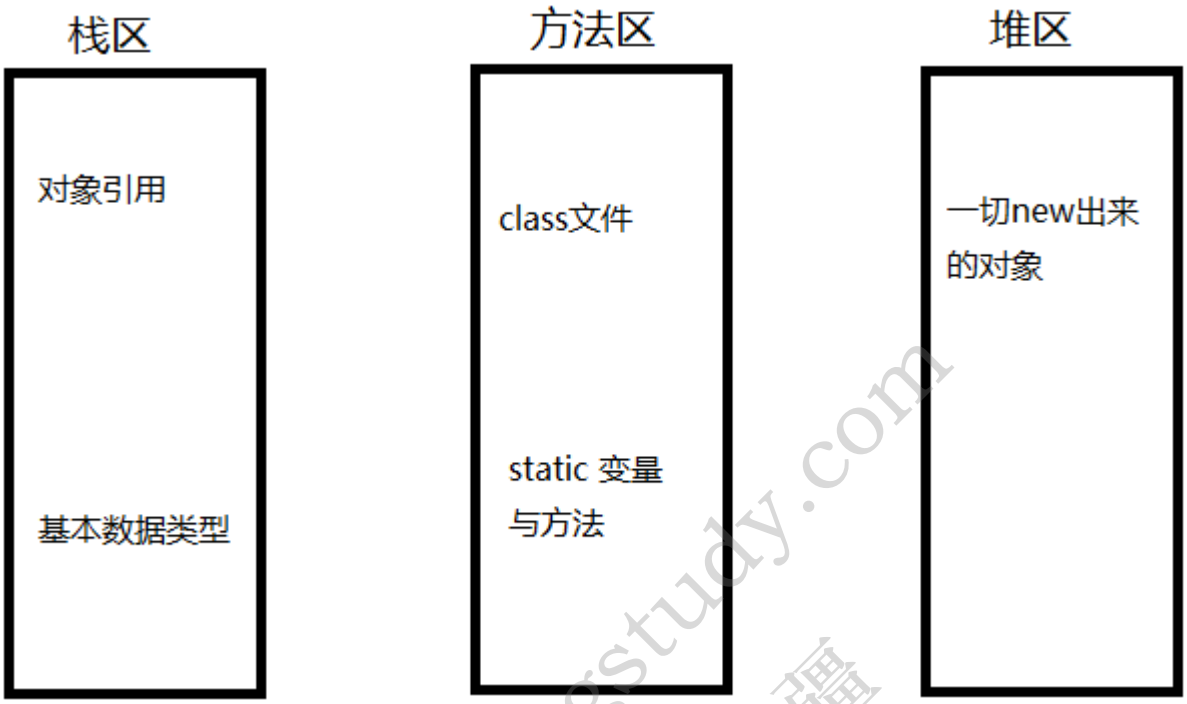
1.在内存中份数不同

不管有多少个对象，static变量只有1份。对于每个对象，实例变量都会有单独的一份

static变量是属于整个类的，也称为类变量。而非静态变量是属于对象的，也称为实例变量

2.在内存中存放的位置不同

静态变量存在方法区中， 实例变量存在堆内存中 *



3.访问的方式不同

实例变量： *对象名.变量名 stu1.name="小明明";*

静态变量：*对象名.变量名 stu1.schoolName="西二旗小学";* 不推荐如此使用

类名.变量名 Student.schoolName="东三旗小学"; 推荐使用

4.在内存中分配空间的时间不同

实例变量：创建对象的时候才分配了空间。静态变量：第一次使用类的时候

Student.schoolName="东三旗小学";或者Student stu1 = new Student("小明","男",20,98);

static方法也可以通过对象名.方法名和类名.方法名两种方式来访问

static代码块。当类被第一次使用时（可能是调用static属性和方法，或者创建其对象）执行静态代码块，且只被执行一次，主要作用是实现static属性的初始化。

static内部类：属于整个外部类，而不是属于外部类的每个对象。不能访问外部类的非静态成员（变量或者方法），可以访问外部类的静态成员

---

25. final和abstract关键字的作用

final和abstract是功能相反的两个关键字，可以对比记忆

abstract可以用来修饰类和方法，不能用来修饰属性和构造方法；使用abstract修饰的类是抽象类，需要被继承，使用abstract修饰的方法是抽象方法，需要子类被重写。

final可以用来修饰类、方法和属性，不能修饰构造方法。使用final修饰的类不能被继承，使用final修饰的方法不能被重写，使用final修饰的变量的值不能被修改，所以就成了常量。

特别注意：final修饰基本类型变量，其值不能改变，由原来的变量变为常量；但是final修饰引用类型变量，栈内存中的引用不能改变，但是所指向的堆内存中的对象的属性值仍旧可以改变。例如:

```
/*
 * @Description  QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
class Test {
    public static void main(String[] args) {
        final People people = new People("狂神");
        dog.name = "秦疆";//正确
        people = new People("秦疆");//错误
    }
}
```

26. final、finally、finalize的区别

final修饰符(关键字)如果一个类被声明为final，意味着它不能再派生出新的子类，不能作为父类被继承例如：String类、Math类等。将变量或方法声明为final，可以保证它们在使用中不被改变。被声明为final的变量必须在声明时给定初值，而在以后的引用中只能读取，不可修改。被声明为final的方法也同样只能使用，不能重写，但是能够重载。使用final修饰的对象，对象的引用地址不能变，但是对象的值可以变!

finally在异常处理时提供 finally 块来执行任何清除操作。如果有finally的话，则不管是否发生异常，finally语句都会被执行。一般情况下，都把关闭物理连接(IO流、数据库连接、Socket连接)等相关操作，放入到此代码块中.。

finalize方法名。Java 技术允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要清理工作。finalize() 方法是在垃圾收集器删除对象之前被调用的。它是在 Object 类中定义的，因此所有的类都继承了它。子类覆盖 finalize() 方法以整理系统资源或者执行其他清理工作。一般情况下，此方法由JVM调用，程序员不要去调用!

27. 写出java.lang.Object类的六个常用方法

(1)public boolean **equals**(java.lang.Object)比较对象的地址值是否相等，如果子类重写，则比较对象的内容是否相等；

(2)public native int **hashCode()** 获取哈希码

(3)public java.lang.String **toString**() 把数据转变成字符串

(4)public final native java.lang.Class **getClass**() 获取类结构信息

(5)protected void **finalize**() throws java.lang.Throwable垃圾回收前执行的方法

(6)protected native Object **clone**() throws java.lang.CloneNotSupportedException 克隆

(7)public final void **wait**() throws java.lang.InterruptedException多线程中等待功能

(8)public final native void **notify**() 多线程中唤醒功能

(9)public final native void **notifyAll**() 多线程中唤醒所有等待线程的功能

28. 权限修饰符的区别

	同一个类	同一个包中	子类	所有类
private	*			
default	*	*		
protected	*	*	*	
public	*	*	*	*

类的访问权限只有两种:

public 公共的 可被同一项目中所有的类访问。 (必须与文件名同名)

default 默认的 可被同一个包中的类访问。

成员(成员变量或成员方法)访问权限共有四种：

public 公共的 可以被项目中所有的类访问。(项目可见性)

protected 受保护的 可以被这个类本身访问;同一个包中的所有其他的类访问;被它的子类(同一个包以及不同包中的子类)访问。(子类可见性)

default 默认的被这个类本身访问;被同一个包中的类访问。(包可见性)

private 私有的 只能被这个类本身访问。(类可见性)

---

29. 继承条件下构造方法的执行过程

继承条件下构造方法的调用规则如下：

情况1：如果子类的构造方法中没有通过super显式调用父类的有参构造方法，也没有通过this显式调用自身的其他构造方法，则系统会默认先调用父类的无参构造方法。在这种情况下，写不写“super();”语句，效果是一样的。

情况2：如果子类的构造方法中通过super显式调用父类的有参构造方法，那将执行父类相应构造方法，而不执行父类无参构造方法。

情况3：如果子类的构造方法中通过this显式调用自身的其他构造方法，在相应构造方法中应用以上两条规则。

特别注意的是，如果存在多级继承关系，在创建一个子类对象时，以上规则会多次向更高一级父类应用，一直到执行顶级父类Object类的无参构造方法为止。

---

30. ==和equals的区别和联系

“==”是关系运算符，equals()是方法，同时他们的结果都返回布尔值；

“==”使用情况如下：

a) 基本类型，比较的是值

b) 引用类型，比较的是地址

c) 不能比较没有父子关系的两个对象

equals()方法使用如下：

a) 系统类一般已经覆盖了equals()，比较的是内容。

b) 用户自定义类如果没有覆盖equals()，将调用父类的equals（比如是Object），而Object的equals的比较是地址（return (this == obj);）

c) 用户自定义类需要覆盖父类的equals()

注意：Object的==和equals比较的都是地址，作用相同

---

31. 谈谈Java的多态

多态性是OOP中的一个重要特性，主要是用来实现动态联编的，换句话说，就是程序的最终状态只有在执行过程中才被决定而非在编译期间就决定了。这对于大型系统来说能提高系统的灵活性和扩展性。

多态可以让我们不用关心某个对象到底是什么具体类型，就可以使用该对象的某些方法，从而实现更加灵活的编程，提高系统的可扩展性。

实现多态的三个条件（前提条件，向上转型、向下转型）

1、继承的存在；（继承是多态的基础，没有继承就没有多态）

2、子类重写父类的方法。（多态下会调用子类重写后的方法）

3、父类引用变量指向子类对象。（涉及子类到父类的类型转换）

向上转型 Animal a = new Cat();

将一个父类的引用指向一个子类对象，称为向上转型，自动进行类型转换。此时通过父类引用变量调用的方法是子类覆盖或继承父类的方法，而不是父类的方法,此时通过父类引用变量无法调用子类特有的方法.

向下转型 Cat a2 = (Cat)a;

将一个指向子类对象的引用赋给一个子类的引用，成为向下转型，此时必须进行强制类型转换。向下转型必须转换为父类引用指向的真实子类类型，否则将出现ClassCastException，不是任意的强制转换.

向下转型时可以结合使用instanceof运算符进行强制类型转换，比如出现转换异常--ClassCastException.比如：本来是狗，我把它转成猫。就会报这个异常。

---

32. 简述Java的垃圾回收机制

传统的C/C++语言，需要程序员负责回收已经分配内存。

显式回收垃圾回收的缺点：

1）程序忘记及时回收，从而导致内存泄露，降低系统性能。

2）程序错误回收程序核心类库的内存，导致系统崩溃。

Java语言不需要程序员直接控制内存回收，是由JRE在后台自动回收不再使用的内存，称为垃圾回收机制，简称GC；

1）可以提高编程效率。

2）保护程序的完整性。

3）其开销影响性能。Java虚拟机必须跟踪程序中有用的对象，确定哪些是无用的。

垃圾回收机制的特点:

1）垃圾回收机制回收JVM堆内存里的对象空间,不负责回收栈内存数据。

2) 对其他物理连接，比如数据库连接、输入流输出流、Socket连接无能为力。

3) 垃圾回收发生具有不可预知性，程序无法精确控制垃圾回收机制执行。

4) 可以将对象的引用变量设置为null，暗示垃圾回收机制可以回收该对象。

现在的JVM有多种垃圾回收实现算法，表现各异。

垃圾回收机制回收任何对象之前，总会先调用它的finalize方法（如果覆盖该方法，让一个新的引用变量重新引用该对象，则会重新激活对象）。

程序员可以通过System.gc()或者Runtime.getRuntime().gc()来通知系统进行垃圾回收，会有一些效果，但是系统是否进行垃圾回收依然不确定。

永远不要主动调用某个对象的finalize方法，应该交给垃圾回收机制调用。

---

33. 基本数据类型和包装类

1) 八个基本数据类型的包装类

基本数据类型	包装类
byte	Byte
boolean	Boolean
short	Short
char	Character
int	Integer
long	Long
float	Float
double	Double

2) 为什么为基本类型引入包装类

2.1、基本数据类型有方便之处，简单、高效。

2.2、 但是Java中的基本数据类型却是不面向对象的（没有属性、方法），这在实际使用时存在很多的不便（比如集合的元素只能是Object）.

为了解决这个不足，在设计类时为每个基本数据类型设计了一个对应的类进行包装，这样八个和基本数据类型对应的类统称为包装类(Wrapper Class)。

3) 包装类和基本数据类型之间的转换

3.1包装类------ wrapperInstance.xxxValue() ------>基本数据类型

3.2包装类<---new WrapperClass(primitive)-- new WrapperClass(string)------基本数据类型

4)自动装箱和自动拆箱

JDK1.5提供了自动装箱（autoboxing）和自动拆箱（autounboxing）功能, 从而实现了包装类和基本数据类型之间的自动转换

5)、包装类还可以实现基本类型变量和字符串之间的转换

基本类型变量------------String.valueof()------------>字符串

基本类型变量<------------WrapperClass.parseXxx(string)------------字符串

---

34. Integer与int的区别

int是java提供的8种原始数据类型之一，Java为每个原始类型提供了封装类，Integer是java为int提供的封装类。

int的默认值为0，而Integer的默认值为null，即Integer可以区分出未赋值和值为0的区别，int则无法表达出未赋值的情况，例如，要想表达出没有参加考试和考试成绩为0的区别，则只能使用Integer。在JSP开发中，Integer的默认为null，所以用el表达式在文本框中显示时，值为空白字符串，而int默认的默认值为0，所以用el表达式在文本框中显示时，结果为0，所以，int不适合作为web层的表单数据的类型。

在Hibernate中，如果将OID定义为Integer类型，那么Hibernate就可以根据其值是否为null而判断一个对象是否是临时的，如果将OID定义为了int类型，还需要在hbm映射文件中设置其unsaved-value属性为0。

另外，Integer提供了多个与整数相关的操作方法，例如，将一个字符串转换成整数，Integer中还定义了表示整数的最大值和最小值的常量。

---

35. java.sql.Date和java.util.Date的联系和区别

1）、java.sql.Date是java.util.Date的子类，是一个包装了毫秒值的瘦包装器，允许 JDBC 将毫秒值标识为 SQL DATE 值。毫秒值表示自 1970 年 1 月 1 日 00:00:00 GMT 以来经过的毫秒数。 为了与 SQL DATE 的定义一致，由 java.sql.Date 实例包装的毫秒值必须通过将时间、分钟、秒和毫秒设置为与该实例相关的特定时区中的零来"规范化"。 说白了，java.sql.Date就是与数据库Date相对应的一个类型，而java.util.Date是纯java的Date。

2）、JAVA里提供的日期和时间类java.sql.Date和java.sql.Timestamp,只会从数据库里读取某部分值，这有时会导致丢失数据。例如一个包含2002/05/22 5:00:57 PM的字段，读取日期时得到的是2002/05/22,而读取时间时得到的是5:00:57 PM. 你需要了解数据库里存储时间的精度。有些数据库，比如MySQL,精度为毫秒，然而另一些数据库，包括Oracle,存储SQL DATE类型数据时，毫秒部分的数据是不保存的。

以下操作中容易出现不易被发现的BUG：获得一个JAVA里的日期对象。 从数据库里读取日期，试图比较两个日期对象是否相等。如果毫秒部分丢失，本来认为相等的两个日期对象，用Equals方法可能返回false。.sql.Timestamp类比java.util.Date类精确度要高。

`java.sql.Date`和`java.util.Date` 最大的不同在于`java.sql.Date`只记录日期，而没有具体这一天的时间。所以举例来说，如果当前是2009-12-24 23:20，你创建一个`java.sql.Date` 将只记下2009-12-24这个信息。若你需要保留时间进行JDBC操作，请使用`java.sql.Timestamp`代替。

总之，java.util.Date 就是Java的日期对象，而java.sql.Date 是针对SQL语句使用的，只包含日期而没有时间部分。

---

36. 递归应用题

题目：使用递归算法输出某个目录下及其子目录下所有文件.

递归：自动调用自己，需要定义递归出口.

题目分析 : 参数为一个指定的目录 . 输出所有文件列表;

```
import java.io.File;
/*
 * @Description  QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class ShowFile {

/*
  静态私有化方法的特点：
    1. 方法只能在类内部使用。
    2. 性能提高，静态的函数无需检测this指针是否为空。
    （一般函数的第一个参数是对象本身，而静态的可以为null,不需要额外的检查）
*/
    private static void test(String path) {

        //list()方法是返回某个目录下的所有文件和目录的文件名，返回的是String数组
        //listFiles()方法是返回某个目录下所有文件和目录的绝对路径，返回的是File数组
        File f = new File(path);
        File[] fs = f.listFiles();

        //如果没有文件,返回空;
        if (fs == null) {
            return;
        }
        //递归遍历输出
        for (File file : fs) {
            //如果是文件就直接打印出来,否则就递归循环
            if (file.isFile()) {
                System.out.println(file.getPath());
            } else {
                test(file.getPath());
            }
        }
    }

    public static void main(String[] args) {
        String path = "E:\\生活\\图片";
        test(path);
    }

}
```


37. 关于Java编译，下面哪一个正确（ ）

A	**Java程序经编译后产生machine code (机器代码)**
B	Java程序经编译后会生产byte code (字节码)
C	Java程序经编译后会产生DLL (动态链接库)
D	以上都不正确

答案：B

分析： Java是解释型语言，编译出来的是字节码； 因此A不正确，C是C/C++语言编译动态链接库的文件为.DLL； 正确答案为B

---

38. 下列说法正确的有（ ）

A	**class中的construtor不可省略**
B.	construtor与class同名，但方法不能与class同名
C.	construtor在一个对象被new时执行
D.	一个class只能定义一个construtor

答案：C

分析：A：如果class中的construtor省略不写，系统会默认提供一个无参构造

B：方法名可以与类名同名，只是不符合命名规范

D：一个class中可以定义N多个construtor，这些construtor构成构造方法的重载

---

39. Java中接口的修饰符可以为（ ）

A	**private**
B.	**protected**
C.	**final**
D.	**abstract**

答案：D

分析：接口中的访问权限修饰符只可以是public或default

接口中的所有的方法必须要实现类实现，所以不能使用final

接口中所有的方法默认都是abstract的，所以接口可以使用abstract修饰，但通常abstract可以省略不写

---

40. 给定以下代码，程序将输出 （ ）

```java
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
class A {
    public A(){
        System.out.print("A");
    }
}
class B extends A{
    public B(){
        System.out.print("B");
    }
    public static void main(String[] args) {
        B b=new B();
    }
}
```

A	**不能通过编译**
B.	通过编译，输出AB
C.	通过编译，输出B
D.	通过编译，输出A

答案：B

分析：在继承关系下，创建子类对象，先执行父类的构造方法，再执行子类的构造方法。

41~60

41. 下列关于关键字的使用说法错误的是（ ）

A.	**abstract不能与final并列修饰同一个类**
B.	abstract类中可以有private的成员
C.	abstract方法必须在abstract类中
D.	static方法能处理非static的属性

答案：D 分析：因为static得方法在装载class得时候首先完成，比 构造方法早，此时非static得属性和方法还没有完成初始化所以不能调用。

---

42. 下列哪些说法是正确的（ ）

A.	**程序员必须创建一个线程来释放内存**
B.	内存回收程序负责释放无用内存
C.	内存回收程序允许程序员直接释放内存
D.	内存回收程序可以在指定的时间释放内存对象

答案：B

分析： A. 程序员不需要创建线程来释放内存. C. 也不允许程序员直接释放内存. D. 不一定在什么时刻执行垃圾回收.

---

43. 选出合理的标识符（ ）

A.	**_sysl_111**
B.	2 mail
C.	$change
D.	class

答案：AC

分析： 标识符的命令规范，可以包含字母、数字、下划线、$，不能以数字开头，不能是Java关键字

---

44. 下列说法正确的是（ ）

A.	**java.lang.Cloneable是类**
B.	java.langRunnable是接口
C.	Double对象在java.lang包中
D.	Double a=1.0是正确的java语句

答案：BCD

分析：java.lang.Cloneable是接口

---

45. 定义一个类名为”MyClass.java”的类，并且该类可被一个工程中的所有类访问，那么该类的正确声明为（ ）

A.	**private class MyClass extends Object**
B.	class MyClass extends Object
C.	public class MyClass
D.	public class MyClass extends Object

答案：CD

分析： A 类的访问权限只能是public或default

B使用默认访问权限的类，只能在本包中访问

---

46. 面向对象的特征有哪些方面？请用生活中的例子来描述。

面向对象的三大特征：封装、继承、多态。

举例：

比如设计一个游戏:

我建了一个对象，叫人。 属性 : 性别，年龄，职业，等级，血量 方法 : 逃跑，吃饭，睡觉，死。

我现在创建了一个对象，名叫战士。 战士的属性 : 性别，年龄，职业，等级，战斗力，血量。 它的方法 : 战斗，逃跑，吃饭，睡觉，死。

我让人，成为战士的父类，战士可以直接继承人的属性和方法。 战士修改成— 属性 : 战斗力。 方法 : 战斗。

看上去战士的资料变少了，实际上没有，我们仍然可以调用方法—战士.死。 而且我们还可以重载战士.死的方法，简称重载死法。

我还建了一个对象—法师，父类也是人。 属性 : 法力值 方法 : 施法，泡妞。 你看，用了继承，创建对象变得更方便了。

再后来，我又建立了一个对象，叫怪物。 属性 : 等级，战力，血量。 方法 : 战斗，死。

建了个对象，叫白兔怪，父类怪物，可继承怪物所有的属性和方法。 属性 : 毛色。 方法 : 卖萌，吃胡萝卜。

---

47. 说明内存泄漏和内存溢出的区别和联系，结合项目经验描述Java程序中如何检测？如何解决？

1、内存泄漏memory leak :

是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄漏似乎不会有大的影响，但内存泄漏堆积后的后果就是内存溢出。

2、内存溢出 out of memory :

指程序申请内存时，没有足够的内存供申请者使用，或者说，给了你一块存储int类型数据的存储空间，但是你却存储long类型的数据，那么结果就是内存不够用，此时就会报错OOM,即所谓的内存溢出。

3、二者的关系：

内存泄漏的堆积最终会导致内存溢出 内存溢出就是你要的内存空间超过了系统实际分配给你的空间，此时系统相当于没法满足你的需求，就会报内存溢出的错误。 内存泄漏是指你向系统申请分配内存进行使用(new)，可是使用完了以后却不归还(delete)，结果你申请到的那块内存你自己也不能再访问（也许你把它的地址给弄丢了），而系统也不能再次将它分配给需要的程序。就相当于你租了个带钥匙的柜子，你存完东西之后把柜子锁上之后，把钥匙丢了或者没有将钥匙还回去，那么结果就是这个柜子将无法供给任何人使用，也无法被垃圾回收器回收，因为找不到他的任何信息。 内存溢出：一个盘子用尽各种方法只能装4个果子，你装了5个，结果掉倒地上不能吃了。这就是溢出。比方说栈，栈满时再做进栈必定产生空间溢出，叫上溢，栈空时再做退栈也产生空间溢出，称为下溢。就是分配的内存不足以放下数据项序列,称为内存溢出。说白了就是我承受不了那么多，那我就报错。

4、内存泄漏的分类（按发生方式来分类）

常发性内存泄漏。发生内存泄漏的代码会被多次执行到，每次被执行的时候都会导致一块内存泄漏。 偶发性内存泄漏。发生内存泄漏的代码只有在某些特定环境或操作过程下才会发生。常发性和偶发性是相对的。对于特定的环境，偶发性的也许就变成了常发性的。所以测试环境和测试方法对检测内存泄漏至关重要。 一次性内存泄漏。发生内存泄漏的代码只会被执行一次，或者由于算法上的缺陷，导致总会有一块仅且一块内存发生泄漏。比如，在类的构造函数中分配内存，在析构函数中却没有释放该内存，所以内存泄漏只会发生一次。 隐式内存泄漏。程序在运行过程中不停的分配内存，但是直到结束的时候才释放内存。严格的说这里并没有发生内存泄漏，因为最终程序释放了所有申请的内存。但是对于一个服务器程序，需要运行几天，几周甚至几个月，不及时释放内存也可能导致最终耗尽系统的所有内存。所以，我们称这类内存泄漏为隐式内存泄漏。

5、内存溢出的原因及解决方法：

（1）内存溢出原因：

内存中加载的数据量过于庞大，如一次从数据库取出过多数据； 集合类中有对对象的引用，使用完后未清空，使得JVM不能回收； 代码中存在死循环或循环产生过多重复的对象实体； 使用的第三方软件中的BUG； 启动参数内存值设定的过小

（2）内存溢出的解决方案：

第一步，修改JVM启动参数，直接增加内存。(-Xms，-Xmx参数一定不要忘记加。)

第二步，检查错误日志，查看“OutOfMemory”错误前是否有其 它异常或错误。

第三步，对代码进行走查和分析，找出可能发生内存溢出的位置。

重点排查以下几点：

1.检查对数据库查询中，是否有一次获得全部数据的查询。一般来说，如果一次取十万条记录到内存，就可能引起内存溢出。这个问题比较隐蔽，在上线前，数据库中数据较少，不容易出问题，上线后，数据库中数据多了，一次查询就有可能引起内存溢出。因此对于数据库查询尽量采用分页的方式查询。

2.检查代码中是否有死循环或递归调用。

3.检查是否有大循环重复产生新对象实体。

4.检查对数据库查询中，是否有一次获得全部数据的查询。一般来说，如果一次取十万条记录到内存，就可能引起内存溢出。这个问题比较隐蔽，在上线前，数据库中数据较少，不容易出问题，上线后，数据库中数据多了，一次查询就有可能引起内存溢出。因此对于数据库查询尽量采用分页的方式查询。

5.检查List、MAP等集合对象是否有使用完后，未清除的问题。List、MAP等集合对象会始终存有对对象的引用，使得这些对象不能被GC回收。

第四步，使用内存查看工具动态查看内存使用情况

---

48. 什么是Java的序列化，如何实现Java的序列化？列举在哪些程序中见过Java序列化？

Java中的序列化机制能够将一个实例对象（只序列化对象的属性值，而不会去序列化什么所谓的方法。）的状态信息写入到一个字节流中使其可以通过socket进行传输、或者持久化到存储数据库或文件系统中；然后在需要的时候通过字节流中的信息来重构一个相同的对象。一般而言，要使得一个类可以序列化，只需简单实现java.io.Serializable接口即可。

对象的序列化主要有两种用途： 1）把对象的字节序列永久地保存到硬盘上，通常存放在一个文件中； 2）在网络上传送对象的字节序列。

在很多应用中，需要对某些对象进行序列化，让它们离开内存空间，入住物理硬盘，以便长期保存。比如最常见的是Web服务器中的Session对象，当有 10万用户并发访问，就有可能出现10万个Session对象，内存可能吃不消，于是Web容器就会把一些seesion先序列化到硬盘中，等要用了，再把保存在硬盘中的对象还原到内存中。

当两个进程在进行远程通信时，彼此可以发送各种类型的数据。无论是何种类型的数据，都会以二进制序列的形式在网络上传送。发送方需要把这个Java对象转换为字节序列，才能在网络上传送；接收方则需要把字节序列再恢复为Java对象。

---

49. 不通过构造函数也能创建对象吗？

Java创建对象的几种方式（重要）：

1、 用new语句创建对象，这是最常见的创建对象的方法。

2、 运用反射手段,调用java.lang.Class或者java.lang.reflect.Constructor类的newInstance()实例方法。

3、 调用对象的clone()方法。

4、运用反序列化手段，调用java.io.ObjectInputStream对象的 readObject()方法。

(1)和(2)都会明确的显式的调用构造函数 ；(3)是在内存上对已有对象的影印，所以不会调用构造函数 ；(4)是从文件中还原类的对象，也不会调用构造函数。

---

50. 匿名内部类可不可以继承或实现接口。为什么？

匿名内部类是没有名字的内部类,不能继承其它类,但一个内部类可以作为一个接口,由另一个内部类实现.

1、由于匿名内部类没有名字，所以它没有构造函数。因为没有构造函数，所以它必须完全借用父类的构造函数来实例化，换言之：匿名内部类完全把创建对象的任务交给了父类去完成。

2、在匿名内部类里创建新的方法没有太大意义，但它可以通过覆盖父类的方法达到神奇效果，如上例所示。这是多态性的体现。

3、因为匿名内部类没有名字，所以无法进行向下的强制类型转换，持有对一个匿名内部类对象引用的变量类型一定是它的直接或间接父类类型。

匿名类是不能有名称的类，所以没办法引用它们。必须在创建时，作为new语句的一部分来声明它们。

这就要采用另一种形式的new语句，如下所示： `new <类或接口> <类的主体>`

这种形式的new语句声明一个新的匿名类，它对一个给定的类进行扩展，或者实现一个给定的接口。它还创建那个类的一个新实例，并把它作为语句的结果而返回。要扩展的类和要实现的接口是new语句的操作数，后跟匿名类的主体。

如果匿名类对另一个类进行扩展，它的主体可以访问类的成员、覆盖它的方法等等，这和其他任何标准的类都是一样的。如果匿名类实现了一个接口，它的主体必须实现接口的方法。

注意匿名类的声明是在编译时进行的，实例化在运行时进行。这意味着for循环中的一个new语句会创建相同匿名类的几个实例，而不是创建几个不同匿名类的一个实例。

从技术上说，匿名类可被视为非静态的内部类，所以它们具有和方法内部声明的非静态内部类一样的权限和限制。

如果要执行的任务需要一个对象，但却不值得创建全新的对象（原因可能是所需的类过于简单，或者是由于它只在一个方法内部使用），匿名类就显得非常有用。匿名类尤其适合在Swing应用程序中快速创建事件处理程序。

具体可参考博文：https://blog.csdn.net/liupeng900605/article/details/7723529

51. 在Java中，为什么基本类型不能做为HashMap的键值，而只能是引用类型，把引用类型做为HashMap的健值，需要注意哪些地方。

在Java中是使用泛型来约束HashMap中的key和value的类型的，即HashMap<K, V>；而泛型在Java的规定中必须是对象Object类型的，也就是说HashMap<K, V>可以理解为HashMap<Object,Object>，很显然基本数据类型不是Object类型的，因此不能作为键值，只能是引用类型。虽然我们在HashMap中可以这样添加数据：“map.put(1, “Java”)；”，但实际上是将其中的key值1进行了自动装箱操作，变为了Integer类型。

引用数据类型分为两类：系统提供的引用数据类型（如包装类、String等）以及自定义引用数据类型。系统提供的引用数据类型中已经重写了HashCode()和equals()两个方法，所以能够保证Map中key值的唯一性；但是自定义的引用数据类型需要自己重写HashCode()和equals()这两个方法，以保证Map中key值的唯一性。

52. 简述Java中如何实现多态

实现多态有三个前提条件：

1、 继承的存在；（继承是多态的基础，没有继承就没有多态）。

2、子类重写父类的方法。（多态下会调用子类重写后的方法）。

3、父类引用变量指向子类对象。（涉及子类到父类的类型转换）。

最后使用父类的引用变量调用子类重写的方法即可实现多态。即,同一个方法，可以有不同的展现结果;

53. 以下对继承的描述错误的是 ()

A.	**Java中的继承允许一个子类继承多个父类**
B.	父类更具有通用性，子类更具体
C.	Java中的继承存在着传递性
D.	当实例化子类时会递归调用父类中的构造方法

答案：A

分析：Java是单继承的，一个类只能继承一个父类。

54. Java 中 Math.random（ ）/Math.random（ ）值为？

```
public class Test {
    public static void main(String[] args) {
        //调用这个Math.Random()函数能够返回带正号的double值，该值大于等于0.0且小于1.0，即取值范围是[0.0,1.0)的左闭右开区间，返回值是一个伪随机选择的数，在该范围内（近似）均匀分布。
        System.out.println(Math.random()/Math.random());
    }
}
```

如果除数与被除数均不为0.0的话，则取值范围为[0, +∞]。+∞在Java中显示的结果为Infinity。

如果除数与被除数均为0.0的话，则运行结果为NaN（Not a Number的简写），计算错误。

---

55. Java中，如果Manager是Employee的子类，那么`Pair <Manager>`是`Pair <Employee>`的子类吗？

不是，两者没有任何关联；

Pair是单独的类，只不过用不同类型的参数（泛型）进行了相应的实例化而已；所以,`Pair<Manager>`和`Pair<Employee>`不是子类的关系。

---

56. 接口和抽象类的区别

相同点

抽象类和接口均包含抽象方法，类必须实现所有的抽象方法 .

抽象类和接口都不能实例化，他们位于继承树的顶端，用来被其他类继承和实现

两者的区别主要体现在两方面：语法方面和设计理念方面

语法方面的区别是比较 低层次的，非本质的，**主要表现在：**

接口中只能定义全局静态常量，不能定义变量。抽象类中可以定义常量和变量。

接口中所有的方法都是全局抽象方法。抽象类中可以有0个、1个或多个，甚至全部都是抽象方法。

抽象类中可以有构造方法，但不能用来实例化，而在子类实例化是执行，完成属于抽象类的初始化操作。接口中不能定义构造方法。

一个类只能有一个直接父类（可以是抽象类），但可以充实实现多个接口。一个类使用extends来继承抽象类，使用implements来实现接口。

二者的主要区别还是在设计理念上，其决定了某些情况下到底使用抽象类还是接口。

抽象类体现了一种继承关系，目的是复用代码，抽象类中定义了各个子类的相同代码，可以认为父类是一个实现了部分功能的“中间产品”，而子类是“最终产品”。父类和子类之间必须存在“is-a:继承”的关系，即父类和子类在概念本质上应该是相同的。

接口并不要求实现类和接口在概念本质上一致的，仅仅是实现了接口定义的约定或者能力而已。接口定义了“做什么”，而实现类负责完成“怎么做”，体现了功能（规范）和实现分离的原则。接口和实现之间可以认为是一种“has-a:组合的关系”

- IS-A、HAS-A和USE-A都是用来便是类与类之间的关系
- IS-A表示继承。父类与子类，具有很高的耦合度。
- HAS-A表示组合。是整体与部分的关系，同时它们的生命周期都是一样的。
- USE-A表示依赖。依然是其中一个拥有另外一个，但是不负责销毁，也就是声明周期不一样。

---

57. 同步代码块和同步方法有什么区别

相同点：

同步方法就是在方法前加关键字synchronized，然后被同步的方法一次只能有一个线程进入，其他线程等待。而同步代码块则是在方法内部使用大括号使得一个代码块得到同步。同步代码块会有一个同步的“目标”，使得同步块更加灵活一些（同步代码块可以通过“目标”决定需要锁定的对象）。一般情况下，如果此“目标”为this，那么同步方法和同步代码块没有太大的区别。

区别：

同步方法直接在方法上加synchronized实现加锁，同步代码块则在方法内部加锁。很明显，同步方法锁的范围比较大，而同步代码块范围要小点。一般同步的范围越大，性能就越差。所以一般需要加锁进行同步的时候，范围越小越好，这样性能更好。

---

58. 静态内部类和内部类有什么区别

静态内部类不需要有指向外部类的引用。但非静态内部类需要持有对外部类的引用。

静态内部类可以有静态成员(方法，属性)，而非静态内部类则不能有静态成员(方法，属性)。

非静态内部类能够访问外部类的静态和非静态成员。静态内部类不能访问外部类的非静态成员，只能访问外部类的静态成员。

实例化方式不同：

1) 静态内部类：不依赖于外部类的实例，直接实例化内部类对象 [是真的类]

2) 非静态内部类：通过外部类的对象实例生成内部类对象 [是在对象中new出来的类]

---

59. 反射的概念与作用

反射的概念：

反射，一种计算机处理方式。是程序可以访问、检测和修改它本身状态或行为的一种能力。

- Java反射可以于运行时加载,探知和使用编译期间完全未知的类.
- 程序在运行状态中, 可以动态加载一个只有名称的类, 对于任意一个已经加载的类,都能够知道这个类的所有属性和方法; 对于任意一个对象,都能调用他的任意一个方法和属性;
- 加载完类之后, 在堆内存中会产生一个Class类型的对象(一个类只有一个Class对象), 这个对象包含了完整的类的结构信息,而且这个Class对象就像一面镜子,透过这个镜子看到类的结构,所以被称之为:反射.
- java反射使得我们可以在程序运行时动态加载一个类，动态获取类的基本信息和定义的方法,构造函数,域等。
- 除了检阅类信息外，还可以动态创建类的实例，执行类实例的方法，获取类实例的域值。反射使java这种静态语言有了动态的特性。

反射的作用：

通过反射可以使程序代码访问装载到JVM 中的类的内部信息

1) 获取已装载类的属性信息

2) 获取已装载类的方法

3) 获取已装载类的构造方法信息

反射的优点：

增加程序的灵活性。 如struts中。请求的派发控制。 当请求来到时。struts通过查询配置文件。找到该请求对应的action。 然后通过反射实例化action。并调用响应method。 如果不适用反射，那么你就只能写死到代码里了。 所以说，一个灵活，一个不灵活。 很少情况下是非用反射不可的。大多数情况下反射是为了提高程序的灵活性。 因此一般框架中使用较多。因为框架要适用更多的情况。对灵活性要求较高。

---

60. 提供Java存取数据库能力的包是（ ）

A.	**java.sql**
B.	java.awt
C.	java.lang
D.	java.swing

答案：A

分析：java.awt和javax.swing两个包是图形用户界面编程所需要的包；java.lang包则提供了Java编程中用到的基础类。

61~80

61. 下列运算符合法的是()

A.	**&&**
B.	<>
C.	if
D.	=

答案：AD

分析：

&&是逻辑运算符中的短路与；

<>表示不等于，但是Java中不能这么使用，应该是!=；

if不是运算符；

=是赋值运算符。

---

62. 执行如下程序代码，c的值打印出来是?

```
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
```

```java
public class Test {
    public static void main(String[] args) {
        int a = 0;
        int c = 0;
        do{
            --c; //先自减  c = -1
            a = a - 1;  // a = -1
        } while (a > 0);  //a > 0 会一直循环
        System.out.println(c);
    }
}
```

do-while循环的特点是先执行后判断，所以代码先执行--c操作，得到c为-1，之后执行a=a-1的操作，得到a为-1，然后判断a是否大于0，判断条件不成立，退出循环，输出c为-1。

---

63. 下列哪一种叙述是正确的()

A.	**abstract修饰符可修饰字段，方法和类**
B.	抽象方法的body部分必须用一对大括号{}包住
C.	声明抽象方法，大括号可有可无
D.	声明抽象方法不可写出大括号

答案：D

分析：

abstract只能修饰方法和类，不能修饰字段；A错

抽象方法不能有方法体，即没有{}；B.C错

---

64. 下列语句正确的是()

A.	**形式参数可被视为local Variable**
B.	形式参数可被所有的字段修饰符修饰
C.	形式参数为方法被调用时，真正被传递的参数
D.	形式参数不可以是对象

答案：A

分析：

local Variable为局部变量，形参和局部变量一样都只有在方法内才会发生作用，也只能在方法中使用，不会在方法外可见；

对于形式参数只能用final修饰符，其它任何修饰符都会引起编译器错误；

真正被传递的参数是实参；

形式参数可是基本数据类型也可以是引用类型（对象）。

---

65. 下列哪种说法是正确的()

A.	**实例方法可直接调用超类的实例方法**
B.	实例方法可直接调用超类的类方法
C.	实例方法可直接调用其他类的实例方法
D.	实例方法可直接调用本类的类方法

答案：D

分析：

实例方法不可直接调用超类的私有实例方法；

实例方法不可直接调用超类的私有的类方法；

要看访问权限。

---

66. Java程序的种类有()

A.	**类 (Class)**
B.	Applet
C.	Application
D.	Servlet

答案：BCD

分析：

是Java中的类，不是程序；

内嵌于Web文件中，由浏览器来观看的Applet；

可独立运行的 Application；

服务器端的 Servlet。

---

67. 下列说法正确的有()

A.	**环境变量可在编译source code时指定**
B.	在编译程序时，所指定的环境变置不包括class path
C.	javac 一次可同时编译数个Java 源文件
D.	javac.exe能指定编译结果要置于哪个目录（directory)

答案：BCD

分析：

环境变量一般都是先配置好再编译源文件。

---

68. 下列标识符不合法的有()

A.	**new**
B.	$Usdollars
C.	1234
D.	car.taxi

答案：ACD

分析：

new是Java的关键字；

C. 数字不能开头；

D. 不能有“.”，# 这种特殊字符。

---

69. 下列说法错误的有()

A.	**数组是一种对象**
B.	数组属于一种原生类
C.	int number[]=(31,23,33,43,35,63)
D.	数组的大小可以任意改变

答案：BCD

分析：

B. Java中的原生类（即基本数据类型）有8种，但不包括数组；

C. 语法错误，应该“{…}”，而不是“(…)”；

D. 数组的长度一旦确定就不能修改。

---

70. 不能用来修饰interface的有()

A.	**private**
B.	public
C.	protected
D.	static

答案：ACD

分析：

能够修饰interface的只有public、abstract以及默认的三种修饰符。

---

71. 下列正确的有()

A.	**call by value不会改变实际参数的数值**
B.	call by reference能改变实际参数的参考地址
C.	call by reference 不能改变实际参数的参考地址
D.	call by reference 能改变实际参数的内容

答案：ACD

分析：

Java中参数的传递有两种，一种是按值传递（call by value：传递的是具体的值，如基础数据类型），另一种是按引用传递（call by reference：传递的是对象的引用，即对象的存储地址）。前者不能改变实参的数值，后者虽然不能改变实参的参考地址，但可以通过该地址访问地址中的内容从而实现内容的改变。

---

72. 下列说法错误的有()

A.	**在类方法中可用this来调用本类的类办法**
B.	在类方法中调用本类的类方法时可以直接调用
C.	在类方法中只能调用本类中的类方法
D.	在类方法中绝对不能调用实例方法

答案：ACD

分析：

类方法(Static)是在类加载时被加载到方法区存储的，此时还没有创建对象，所以不能使用this或者super关键字；

C. 在类方法中还可以调用其他类的类方法；

D. 在类方法可以通过创建对象来调用实例方法。

73. 下列说法错误的有()

A.	**Java面向对象语言允许单独的过栈与函数存在**
B.	Java面向对象语言允许单独的方法存在
C.	Java语言中的方法属于类中的成员（member）
D.	Java语言中的方法必定隶属于某一类（对象），调用方法与过程或函数相同

答案：ABC

分析：

B. Java不允许单独的方法，过程或函数存在，需要隶属于某一类中；

C. 静态方法属于类的成员，非静态方法属于对象的成员。

74. 下列说法错误的有()

A.	**能被java.exe成功运行的java class文件必须有main()方法**
B.	J2SDK就是Java API
C.	Appletviewer.exe可利用jar选项运行.jar文件
D.	能被Appletviewer成功运行的java class文件必须有main()方法

答案：BCD

分析：

B. J2SDK是sun公司编程工具，API是指的应用程序编程接口；

C. Appletviewer.exe就是用来解释执行java applet应用程序的，一种执行HTML文件上的Java小程序类的Java浏览器；

D. 能被Appletviewer成功运行的java class文件可以没有main（ ）方法。

75. 请问0.3332的数据类型是()

A.	**float**
B.	double
C.	Float
D.	Double

答案：B 分析：小数默认是双精度浮点型即double类型的。

76. Java接口的修饰符可以为()

A.	**private**
B.	protected
C.	final
D.	abstract

答案：D

分析：

能够修饰interface的只有public、abstract以及默认的三种修饰符。

---

77. 不通过构造函数也能创建对象么()

A.	**是**
B.	否

答案：A

分析：

Java创建对象的几种方式：

(1) 用new语句创建对象，这是最常见的创建对象的方法。

(2) 运用反射手段,调用java.lang.Class或者java.lang.reflect.Constructor类的newInstance()实例方法。

(3) 调用对象的clone()方法。

(4) 运用反序列化手段，调用java.io.ObjectInputStream对象的readObject()方法。

(1)和(2)都会明确的显式的调用构造函数；(3)是在内存上对已有对象的影印，所以不会调用构造函数；(4)是从文件中还原类的对象，也不会调用构造函数。

---

78. 存在使i+1<i的数么?

存在, byte , int 等的的最大值, 加1后变为负数.

```java
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class Test {
    public static void main(String[] args) {
        byte a = 127;
        byte b = (byte) (a+1);
        System.out.println(a+"\t"+b);
    }
}
//输出结果
127 -128
```

---

79. 接口可否继承接口？抽象类是否可实现接口？抽象类是否可继承实体类？

接口可以继承接口

抽象类可以实现接口

抽象类可以继承实体类。

---

80. int与Integer有什么区别？

int是java提供的8种原始数据类型之一。Java为每个原始类型提供了封装类，Integer是java为int提供的封装类。int的默认值为0，而Integer的默认值为null，即Integer可以区分出未赋值和值为0的区别，int则无法表达出未赋值的情况，例如，要想表达出没有参加考试和考试成绩为0的区别，则只能使用Integer。在JSP开发中，Integer的默认为null，所以用el表达式在文本框中显示时，值为空白字符串，而int默认的默认值为0，所以用el表达式在文本框中显示时，结果为0，所以，int不适合作为web层的表单数据的类型。

在Hibernate中，如果将OID定义为Integer类型，那么Hibernate就可以根据其值是否为null而判断一个对象是否是临时的，如果将OID定义为了int类型，还需要在hbm映射文件中设置其unsaved-value属性为0。

另外，Integer提供了多个与整数相关的操作方法，例如，将一个字符串转换成整数，Integer中还定义了表示整数的最大值和最小值的常量。

81~100

81. 可序列化对象为什么要定义serialversionUID值?

SerialVersionUid，简言之，其目的是序列化对象版本控制，有关各版本反序列化时是否兼容。如果在新版本中这个值修改了，新版本就不兼容旧版本，反序列化时会抛出InvalidClassException异常。如果修改较小，比如仅仅是增加了一个属性，我们希望向下兼容，老版本的数据都能保留，那就不用修改；如果我们删除了一个属性，或者更改了类的继承关系，必然不兼容旧数据，这时就应该手动更新版本号，即SerialVersionUid。

说白了就是为了一个兼容性判断.

---

82. 写一个Java正则，能过滤出html中的title中的链接地址和标题.

`<a \b[^>]+\bhref="([^"]*)"[^>]* > ([\s\S]*?) </a>` 分组1和分组2即为href和value

---

83. 十进制数72转换成八进制数是多少？

110

```
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class Test {
    public static void main(String[] args) {
        //10进制转2进制
        System.out.println(Integer.toBinaryString(72));
        //10进制转16进制
        System.out.println(Integer.toHexString(72));
        //10进制转8进制
        System.out.println(Integer.toOctalString(72));
    }
}
//输出
1001000
48
110
```

---

84. Java程序中创建新的类对象，使用关键字new，回收无用的类对象使用关键字free正确么？

Java程序中创建新的类对象，使用关键字new是正确的; 回收无用的类对象使用关键字free是错误的.

垃圾回收机制回收任何对象之前，总会先调用它的finalize方法（如果覆盖该方法，让一个新的引用变量重新引用该对象，则会重新激活对象）。

程序员可以通过System.gc()或者Runtime.getRuntime().gc()来通知系统进行垃圾回收，会有一些效果，但是系统是否进行垃圾回收依然不确定。

---

85. Class类的getDeclaredFields()方法与getFields()的区别？

getDeclaredFields(): 可以获取所有本类自己声明的方法, 不能获取继承的方法

getFields(): 只能获取所有public声明的方法, 包括继承的方法

---

86. 在switch和if-else语句之间进行选取，当控制选择的条件不仅仅依赖于一个x时，应该使用switch结构；正确么？

不正确。

通常情况下，进行比较判断的处理，switch 和if-else可以互相转换来写；if-else作用的范围比switch-case作用范围要大，但是当switch-case和if-else都可以用的情况下，通常推荐使用switch-case。

比如：

```
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public static void main(String[] args) {
    char ch = 'a';

    switch (ch) {
        case 'a':
            System.out.println("A");
            break;
        case 'b':
            System.out.println("B");
            break;
        case 'c':
            System.out.println("C");
            break;
        case 'd':
            System.out.println("D");
            break;
        case 'e':
            System.out.println("E");
            break;
        default:
            System.out.println("other");
            break;
    }
}
```

换为if-else

```
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public static void main(String[] args) {
    char ch = 'a';

    if (ch == 'a') {
        System.out.println("A");
    } else if (ch == 'b') {
        System.out.println('B');
    } else if (ch == 'c') {
        System.out.println("C");
    } else if (ch == 'd') {
        System.out.println("D");
    } else if (ch == 'e') {
        System.out.println("E");
    } else {
        System.out.println("Other");
```

```
    }
}
```

87. 描述&和&&的区别。

&和&&的联系(共同点)：

&和&&都可以用作逻辑与运算符，但是要看使用时的具体条件来决定。

操作数1&操作数2，操作数1&&操作数2，

表达式1&表达式2，表达式1&&表达式2，

情况1：当上述的操作数是boolean类型变量时，&和&&都可以用作逻辑与运算符。

情况2：当上述的表达式结果是boolean类型变量时，&和&&都可以用作逻辑与运算符。

表示逻辑与(and)，当运算符两边的表达式的结果或操作数都为true时，整个运算结果才为true，否则，只要有一方为false，结果都为false。

&和&&的区别(不同点)：

(1)、&逻辑运算符称为逻辑与运算符，&&逻辑运算符称为短路与运算符，也可叫逻辑与运算符。

对于&：无论任何情况，&两边的操作数或表达式都会参与计算。

对于&&：当&&左边的操作数为false或左边表达式结果为false时，&&右边的操作数或表达式将不参与计算，此时最终结果都为false。

综上所述，如果逻辑与运算的第一个操作数是false或第一个表达式的结果为false时，对于第二个操作数或表达式是否进行运算，对最终的结果没有影响，结果肯定是false。推介平时多使用&&，因为它效率更高些。

(2)、&还可以用作位运算符。当&两边操作数或两边表达式的结果不是boolean类型时，&用于按位与运算符的操作。

88. 使用final关键字修饰符一个变量时，是引用不能变，还是引用的对象不能变？

final修饰基本类型变量，其值不能改变。

但是final修饰引用类型变量，栈内存中的引用不能改变，但是所指向的堆内存中的对象的属性值仍旧可以改变。例如

```
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class Test {
    public static void main(String[] args) {
        final Dog dog = new Dog("狗子");
        dog.name = "泰迪";//正确
        dog = new Dog("田园犬");//错误
    }
}
```

89. 请解释以下常用正则含义：\d,\D,\s,.,*,?,|,[0-9]{6},\d+

\d: 匹配一个数字字符。等价于[0-9]

\D: 匹配一个非数字字符。等价于^ [0-9]

\s: 匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [\f\n\r\t\v]

. ：匹配除换行符 \n 之外的任何单字符。要匹配 . ，请使用 . 。

* ：匹配前面的子表达式零次或多次。要匹配 * 字符，请使用 *。

+ ：匹配前面的子表达式一次或多次。要匹配 + 字符，请使用 +。

|:将两个匹配条件进行逻辑"或"（ Or ）运算

[0-9]{6}:匹配连续6个0-9之间的数字

\d+ ：匹配至少一个0-9之间的数字

---

90. 已知表达式int m[] = {0,1,2,3,4,5,6}; 下面那个表达式的值与数组的长度相等()

A.	**m.length()**
B.	m.length
C.	m.length()+1
D.	m.length+1

答案：B

分析：数组的长度是.length

---

91. 下面那些声明是合法的？()

A.	**long l = 4990**
B.	int i = 4L
C.	float f = 1.1
D.	double d = 34.4

答案：AD

分析：B int属于整数型应该是int=4 C应该是float f=1.1f

---

92. 以下选项中选择正确的java表达式()

A.	**int k=new String(“aa”)**
B.	String str = String(“bb”)
C.	char c=74;
D.	long j=8888;

答案：CD

分析：A需要强制类型转换 B String str =new String(“bb”)

93.下列代码的输出结果是

```
public static void main(String[] args) {
    System.out.println(""+("12"=="12"&&"12".equals("12")));
}
```

true

---

94. 以下哪些运算符是含有短路运算机制的？

A.	**&**
B.	&&
C.	|
D.	||

答案：BD

分析：A C是逻辑计算

---

95. 下面哪个函数是public void example(){....}的重载函数？（ ）

A.	**private void example (int m) {...}**
B.	public int example ( ) {...}
C.	public void example2 ( ) {...}
D.	public int example (int m.float f) {...}

答案：AD

分析：BC定义的是新函数

---

96. 给定某java程序片段，该程序运行后，j的输出结果为（ ）

```
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class Test {
    public static void main(String[] args) {
        int  i=1;
        int  j=i++; //2
        // 2>3 ,不执行if
        if ((j>++j)&&(i++==j)){
            j+=i;
        }
        //输出j=2
        System.out.println(j);
    }
}
```

输出：2

---

97. 在java中，无论测试条件是什么，下列（ ）循环将至少执行一次。

A.	**for**
B.	do...while
C.	while
D.	while...do

答案：B

分析： ACD都不一定进行循环

---

98. 打印结果?

```
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class Test {
    public static void main(String[] args) {
        Test t=new Test();
        int b = t.get();
        System.out.println(b);
    }

    public int get()
    {
```

```
        try {
            return 1;
        }finally{
            return 2;
        }
    }
}
```

最终打印结果 : 2

---

99. 指出下列程序的运行结果

```
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class Test {
    public static void main(String[] args) {
        int i=9;
        switch (i) {
            default:
                System.out.println("default");
            case 0:
                System.out.println("zero");
                break;
            case 1:
                System.out.println("one");
                break;
            case 2:
                System.out.println("two");
                break;
        }
    }
}
```


输出结果 :

default zero

---

100. 解释继承、重载、覆盖。

继承（英语：inheritance）是面向对象软件技术当中的一个概念。如果一个类别A“继承自”另一个类别B，就把这个A称为“B的子类别”，而把B称为“A的父类别”也可以称“B是A的超类”。继承可以使得子类别具有父类别的各种属性和方法，而不需要再次编写相同的代码。在令子类别继承父类别的同时，可以重新定义某些属性，并重写某些方法，即覆盖父类别的原有属性和方法，使其获得与父类别不同的功能。另外，为子类别追加新的属性和方法也是常见的做法。一般静态的面向对象编程语言，继承属于静态的，意即在子类别的行为在编译期就已经决定，无法在执行期扩充。

那么如何使用继承呢？用extends关键字来继承父类。

如上面A类与B类，当写继承语句时， class A类 extends B类{ } 其中A类是子类，B类是父类。

	英文	位置不同	作用不同
重载	overload	同一个类中	在一个类里面为一种行为提供多种实现方式并提高可读性
重写	override	子类和父类间	父类方法无法满足子类的要求，子类通过方法重写满足要求

	修饰符	返回值	方法名	参数	抛出异常
重载	无关	无关	相同	不同	无关
重写	大于等于	小于等于	相同	相同	小于等于

101~120

101. 什么是编译型语言，什么是解释型语言？java可以归类到那种？

计算机不能直接理解高级语言，只能理解和运行机器语言，所以必须要把高级语言翻译成机器语言，计算机才能运行高级语言所编写的程序。翻译的方式有两种，一个是编译，一个是解释。

用编译型语言写的程序执行之前，需要一个专门的编译过程，通过编译系统把高级语言翻译成机器语言，把源高级程序编译成为机器语言文件，比如windows下的exe文件。以后就可以直接运行而不需要编译了，因为翻译只做了一次，运行时不需要翻译，所以一般而言，编译型语言的程序执行效率高。

解释型语言在运行的时候才翻译，比如VB语言，在执行的时候，专门有一个解释器能够将VB语言翻译成机器语言，每个语句都是执行时才翻译。这样解释型语言每执行一次就要翻译一次，效率比较低。

编译型与解释型，两者各有利弊。前者由于程序执行速度快，同等条件下对系统要求较低，因此像开发操作系统、大型应用程序、数据库系统等时都采用它，像C/C++、Pascal/Object Pascal（Delphi）等都是编译语言，而一些网页脚本、服务器脚本及辅助开发接口这样的对速度要求不高、对不同系统平台间的兼容性有一定要求的程序则通常使用解释性语言，如JavaScript、VBScript、Perl、Python、Ruby、MATLAB 等等。

JAVA语言是一种编译型-解释型语言，同时具备编译特性和解释特性（其实，确切的说java就是解释型语言，其所谓的编译过程只是将.java文件编程成平台无关的字节码.class文件，并不是向C一样编译成可执行的机器语言，在此请读者注意Java中所谓的“编译”和传统的“编译”的区别）。作为编译型语言，JAVA程序要被统一编译成字节码文件——文件后缀是class。此种文件在java中又称为类文件。java类文件不能再计算机上直接执行，它需要被java虚拟机翻译成本地的机器码后才能执行，而java虚拟机的翻译过程则是解释性的。java字节码文件首先被加载到计算机内存中，然后读出一条指令，翻译一条指令，执行一条指令，该过程被称为java语言的解释执行，是由java虚拟机完成的。

102. 简述操作符（&，|）与操作符（&&，||）的区别

&和&&的联系(共同点)：

&和&&都可以用作逻辑与运算符，但是要看使用时的具体条件来决定。

操作数1&操作数2，操作数1&&操作数2，

表达式1&表达式2，表达式1&&表达式2，

情况1：当上述的操作数是boolean类型变量时，&和&&都可以用作逻辑与运算符。

情况2：当上述的表达式结果是boolean类型变量时，&和&&都可以用作逻辑与运算符。表示逻辑与(and)，当运算符两边的表达式的结果或操作数都为true时，整个运算结果才为true，否则，只要有一方为false，结果都为false。

&和&&的区别(不同点)：

1. & 称为逻辑与运算符，&& 称为短路与运算符，也可叫逻辑与运算符。

对于&：无论任何情况，&两边的操作数或表达式都会参与计算。

对于&&：当&&左边的操作数为false或左边表达式结果为false时，&&右边的操作数或表达式将不参与计算，此时最终结果都为false。

综上所述，如果逻辑与运算的第一个操作数是false或第一个表达式的结果为false时，对于第二个操作数或表达式是否进行运算，对最终的结果没有影响，结果肯定是false。推介平时多使用&&，因为它效率更高些。

2. &还可以用作位运算符。当&两边操作数或两边表达式的结果不是boolean类型时，&用于按位与运算符的操作。

|和||的区别和联系与&和&&的区别和联系类似

---

103. try{}里面有一个return语句，那么紧跟在这个try后的finally, 里面的语句在异常出现后，都会执行么？为什么？

在异常处理时提供 finally 块来执行任何清除操作。

如果有finally的话，则不管是否发生异常，finally语句都会被执行，包括遇到return语句。

finally中语句不执行的唯一情况中执行了System.exit(0)语句。

---

104. 有一段java应用程序，它的主类名是al，那么保存它的源文件可以是？()

A.	**al.java**
B.	al.class
C.	al
D.	都对

答案：A

分析：.class是java的解析文件

---

105. Java类可以作为（ ）

A	**类型定义机制**
B.	数据封装机制
C.	类型定义机制和数据封装机制
D.	上述都不对

答案： C

106. 在调用方法时，若要使方法改变实参的值，可以？（ ）

A	**用基本数据类型作为参数**
B.	用对象作为参数
C.	A和B都对
D.	A和B都不对

答案：B

分析：基本数据类型不能改变实参的值

107. Java语言具有许多优点和特点，哪个反映了java程序并行机制的（ ）

A	**安全性**
B.	多线性
C.	跨平台
D.	可移植

答案：BC

108. 下关于构造函数的描述错误是（ ）

A	**构造函数的返回类型只能是void型**
B.	构造函数是类的一种特殊函数，它的方法名必须与类名相同
C.	构造函数的主要作用是完成对类的对象的初始化工作
D.	一般在创建新对象时，系统会自动调用构造函数

答案：A

分析：构造函数的名字与类的名字相同，并且不能指定返回类型。

---

110. 下面代码执行后的输出是什么（ ）

```
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class Test {
    public static void main(String[] args) {
        outer:
        for (int i = 0; i < 3; i++)
            inner:for (int j = 0; j < 2; j++) {
                if (j == 1)
                    continue outer;
                System.out.println(j + " and " + i);
            }
    }
}
```

结果:

0 and 0 0 and 1 0 and 2

---

111. 给出如下代码，如何使成员变量m被函数fun()直接访问()

```
public class Test {
    private int m;

    public static void fun() {
// some code…
    }
}
```

A	**将private int m 改为 protected int m**
B.	将private int m 改为 public int m
C.	将private int m 改为 static int m
D.	将private int m 改为int m

答案：C

---

112. 下面哪几个函数是public void example（ ）{....}的重载函数（ ）

A	**public void example (int m) {...}**
B.	public int example (int m) {...}
C.	public void example2 ( ) {...}
D.	public int example (int m , float f) {...}

答案：ABD

---

113. 请问以下代码执行会打印出什么？

父类

```
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
class FatherClass {
    public FatherClass() {
        System.out.println("FatherClassCreate");
    }
}
```

子类

```
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
class ChildClass extends FatherClass {
    public ChildClass() {
        System.out.println("ChildClass Create");
    }
    public static void main(String[] args) {
        FatherClass fc = new FatherClass();
        ChildClass cc = new ChildClass();
    }
}
```

结果:

FatherClassCreate

FatherClassCreate

ChildClass Create

---

114. 如果有两个类A、B（注意不是接口），你想同时使用这两个类的功能，那么你会如何编写这个C类呢？

因为类A、B不是接口，所以是不可以直接实现的，但可以将A、B类定义成父子类，那么C类就能实现A、B类的功能了。假如A为B的父类，B为C的父类，此时C就能使用A、B的功能。

---

115. 一个类的构造方法是否可以被重载（overloading），是否可以被子类重写（overrding）？

构造方法可以被重载，但是构造方法不能被重写，子类也不能继承到父类的构造方法

---

116. Java中byte表示的数值范围是什么？

范围是-128至127

---

117. 如何将日期类型格式化为：2013-02-18 10:53:10？

```java
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class Test {
    public static void main(String[] args) {
        Date date= null;
        //第一步：将字符串（2013-02-18 10:53:10）转换成日期Date
        try {
            DateFormat sdf=new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
            String sdate="2013-02-18 10:53:10";
            date = sdf.parse(sdate);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        System.out.println(date);
       //第二步：将日期Date转换成字符串String
        DateFormat  sdf2=new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
        String sdate2=sdf2.format(date);
        System.out.println(sdate2);
    }
}
```

---

118. 不通过构造函数也能创建对象吗（ ）

可以创建

Java创建对象的几种方式（重要）：(1) 用new语句创建对象，这是最常见的创建对象的方法。 (2) 运用反射手段,调用java.lang.Class或者java.lang.reflect.Constructor类的newInstance()实例方法。 (3) 调用对象的clone()方法。 (4) 运用反序列化手段，调用java.io.ObjectInputStream对象的 readObject()方法。 (1)和(2)都会明确的显式的调用构造函数 ；(3)是在内存上对已有对象的影印，所以不会调用构造函数 ；(4)是从文件中还原类的对象，也不会调用构造函数。

---

119. 下面哪些是对称加密算法（ ）

A.	**DES**
B.	MD5
C.	DSA
D.	RSA

答案：A 分析：常用的对称加密算法有：DES、3DES、RC2、RC4、AES 常用的非对称加密算法有：RSA、DSA、ECC 使用单向散列函数的加密算法：MD5、SHA

---

120. 下面的代码段，当输入为2的时候返回值是（ ）

```
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class Test {
    public static void main(String[] args) {
        getValue(2);
    }
    public static int getValue(int i){
        int result = 0;
        switch(i){
            case 1:
                result=result + i;
            case 2:
                result=result+i*2;
            case 3:
                result=result+i*3;
        }
        return result;
    }
}
```

A	**0**
B.	2
C.	4
D.	10

答案：D

分析：

result = 0 + 2 * 2 = 4;

case穿透：

result = 4+6 = 10;

121~140

121. 以下Java代码段会产生几个对象

```
public class Test {
    public static void main(String[] args) {
        String a="a";
        String b="b";
        String c="c";
        c=a+""+b+""+c;
        System.out.print(c);;
    }
}
```

结果 : 一个对象，因为编译期进行了优化，3个字符串常量直接折叠为一个

---

122. Math.round（-11.2）的运行结果是。

答案: -11

分析：

小数点后第一位=5 正数：Math.round(11.5)=12 负数：Math.round(-11.5)=-11 **小数点后第一位<5** 正数：Math.round(11.46)=11 负数：Math.round(-11.46)=-11 **小数点后第一位>5** 正数：Math.round(11.68)=12 负数：Math.round(-11.68)=-12 根据上面例子的运行结果，我们还可以按照如下方式**总结**，或许更加容易记忆： 参数的小数点后第一位<5，运算结果为参数整数部分。 参数的小数点后第一位>5，运算结果为参数整数部分绝对值+1，符号（即正负）不变。 参数的小数点后第一位=5，正数运算结果为整数部分+1，负数运算结果为整数部分。

终结：大于五全部加，等于五正数加，小于五全不加。

---

123. 十进制数278的对应十六进制数

```
System.out.println(Integer.toHexString(278));
```

结果 : 116

---

124. Java中int,long占用的字节数分别是

1：“字节”是byte，“位”是bit ；

2： 1 byte = 8 bit ；

char 在Java中是2个字节。java采用unicode，2个字节（16位）来表示一个字符。

short 2个字节 int 4个字节 long 8个字节

---

125. System.out.println('a'+1);的结果是

'a'是char型，1 是int行，int与char相加，char会被强转为int行，char的ASCII码对应的值是97，所以加一起打印98

---

126. 下列语句那一个正确（ ）

A.	**java程序经编译后会产生machine code**
B.	java程序经编译后会产生 byte code
C.	java程序经编译后会产生DLL
D.	以上都不正确

答案：B

分析：java程序编译后会生成字节码文件,就是.class文件

---

127. 下列说法正确的有（ ）

A.	**class中的constructor不可省略**
B.	constructor必须与class同名，但方法不能与class同名
C.	constructor在一个对象被new时执行
D.	一个class只能定义一个constructor

答案：C

A : 可以省略，默认有无参构造

B : 方法可以同名，但是不建议这样写

D : 构造方法的重载

---

128. 执行如下程序代码后，c的值是（ ）

```
public class Test {
    public static void main(String[] args) {
        int a=0;
        int c=0;
        do {
            --c;
            a = a-1;
        }while (a>0);
    }
}
```

A.	**0**
B.	1
C.	-1
D.	死循环

答案：C

do{...}while(...);语句至少执行一次

---

129. 下列哪一种叙述是正确的（ ）

A.	**abstract修饰符可修饰字段、方法和类**
B.	抽象方法的body部分必须用一对大括号 { } 包住
C.	声明抽象方法，大括号可有可无
D.	声明抽象方法不可写出大括号

答案：D

分析： abstract不能修饰字段。既然是抽象方法，当然是没有实现的方法，根本就没有body部分。

---

130. 下列语句正确的是（ ）

A.	**形式参数可被视为local variable**
B.	形式参数可被字段修饰符修饰
C.	形式参数为方法被调用时，真正被传递的参数
D.	形式参数不可以是对象

答案A：

分析：

A：形式参数可被视为local variable。形参和局部变量一样都不能离开方法。都只有在方法内才会发生作用，也只有在方法中使用，不会在方法外可见。

B：对于形式参数只能用final修饰符，其它任何修饰符都会引起编译器错误。但是用这个修饰符也有一定的限制，就是在方法中不能对参数做任何修改。 不过一般情况下，一个方法的形参不用final修饰。只有在特殊情况下，那就是：方法内部类。 一个方法内的内部类如果使用了这个方法的参数或者局部变量的话，这个参数或局部变量应该是final。

C：形参的值在调用时根据调用者更改，实参则用自身的值更改形参的值（指针、引用皆在此列），也就是说真正被传递的是实参。

D：方法的参数列表指定要传递给方法什么样的信息，采用的都是对象的形式。因此，在参数列表中必须指定每个所传递对象的类型及名字。想JAVA中任何传递对象的场合一样，这里传递的实际上也是引用，并且引用的类型必须正确。--《Thinking in JAVA》

---

131. 成员变量用static修饰和不用static修饰有什么区别？

1，两个变量的生命周期不同。

成员变量随着对象的创建而存在，随着对象的被回收而释放。 静态变量随着类的加载而存在，随着类的消失而消失。 **2，调用方式不同。** 成员变量只能被对象调用。 静态变量可以被对象调用，还可以被类名调用。

对象调用：p.country

类名调用 ：Person.country

3，别名不同。

成员变量也称为实例变量。 静态变量称为类变量。

4，数据存储位置不同。

成员变量数据存储在堆内存的对象中，所以也叫对象的特有数据. 静态变量数据存储在方法区(共享数据区)的静态区，所以也叫对象的共享数据.

---

132. 如果变量用final修饰，则怎样？如果方法final修饰，则怎样？

1、用final修饰的类不能被扩展，也就是说不可能有子类； 2、用final修饰的方法不能被替换或隐藏： ①使用final修饰的实例方法在其所属类的子类中不能被替换（overridden）； ②使用final修饰的静态方法在其所属类的子类中不能被重定义（redefined）而隐藏（hidden）；

3、用final修饰的变量最多只能赋值一次，在赋值方式上不同类型的变量或稍有不同： ①静态变量必须明确赋值一次（不能只使用类型缺省值）；作为类成员的静态变量，赋值可以在其声明中通过初始化表达式完成，也可以在静态初始化块中进行；作为接口成员的静态变量，赋值只能在其声明中通过初始化表达式完成；

②实例变量同样必须明确赋值一次（不能只使用类型缺省值）；赋值可以在其声明中通过初始化表达式完成，也可以在实例初始化块或构造器中进行； ③方法参数变量在方法被调用时创建，同时被初始化为对应实参值，终止于方法体（body）结束，在此期间其值不能改变；

④构造器参数变量在构造器被调用（通过实例创建表达式或显示的构造器调用）时创建，同时被初始化为对应实参值，终止于构造器体结束，在此期间其值不能改变； ⑤异常处理器参数变量在有异常被try语句的catch子句捕捉到时创建，同时被初始化为实际的异常对象，终止于catch语句块结束，在此期间其值不能改变； ⑥局部变量在其值被访问之前必须被明确赋值；

---

133. 在二进制数据中，小数点向右移一位，则数据（ ）

A.	**除以10**
B.	除以2
C.	乘以2
D.	乘以10

答案：C

小数点右移一位.是扩大到原来的2倍

---

134. 面向对象的特征有哪些方面?

面向对象的特征主要有以下几个方面：

1、抽象：抽象是将一类对象的共同特征总结出来构造类的过程，包括数据抽象和行为抽象两方面。抽象只关注对象有哪些属性和行为，并不关注这些行为的细节是什么。

2、继承：继承是从已有类得到继承信息创建新类的过程。提供继承信息的类被称为父类（超类、基类）；得到继承信息的类被称为子类（派生类）。继承让变化中的软件系统有了一定的延续性，同时继承也是封装程序中可变因素的重要手段（如果不能理解请阅读阎宏博士的《Java与模式》或《设计模式精解》中关于桥梁模式的部分）。

3、封装：通常认为封装是把数据和操作数据的方法绑定起来，对数据的访问只能通过已定义的接口。面向对象的本质就是将现实世界描绘成一系列完全自治、封闭的对象。我们在类中编写的方法就是对实现细节的一种封装；我们编写一个类就是对数据和数据操作的封装。可以说，封装就是隐藏一切可隐藏的东西，只向外界提供最简单的编程接口（可以想想普通洗衣机和全自动洗衣机的差别，明显全自动洗衣机封装更好因此操作起来更简单；我们现在使用的智能手机也是封装得足够好的，因为几个按键就搞定了所有的事情）。

4、多态性：多态性是指允许不同子类型的对象对同一消息作出不同的响应。简单的说就是用同样的对象引用调用同样的方法但是做了不同的事情。多态性分为编译时的多态性和运行时的多态性。如果将对象的方法视为对象向外界提供的服务，那么运行时的多态性可以解释为：当A系统访问B系统提供的服务时，B系统有多种提供服务的方式，但一切对A系统来说都是透明的（就像电动剃须刀是A系统，它的供电系统是B系统，B系统可以使用电池供电或者用交流电，甚至还有可能是太阳能，A系统只会通过B类对象调用供电的方法，但并不知道供电系统的底层实现是什么，究竟通过何种方式获得了动力）。方法重载（overload）实现的是编译时的多态性（也称为前绑定），而方法重写（override）实现的是运行时的多态性（也称为后绑定）。运行时的多态是面向对象最精髓的东西，要实现多态需要做两件事：1. 方法重写（子类继承父类并重写父类中已有的或抽象的方法）；2. 对象造型（用父类型引用引用子类型对象，这样同样的引用调用同样的方法就会根据子类对象的不同而表现出不同的行为）。

---

135. float f=3.4;是否正确?

不正确。

3.4是双精度数，将双精度型（double）赋值给浮点型（float）属于下转型（down-casting，也称为窄化）会造成精度损失，因此需要强制类型转换float f =(float)3.4; 或者写成float f =3.4F;。

---

136. short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗?

对于short s1 = 1; s1 = s1 + 1;由于1是int类型，因此s1+1运算结果也是int 型，需要强制转换类型才能赋值给short型。而short s1 = 1; s1 += 1;可以正确编译，因为s1+= 1;相当于s1 = (short)(s1 + 1);其中有隐含的强制类型转换。

---

137. Java 有没有goto?

goto 是Java中的保留字，在目前版本的Java中没有使用。（根据James Gosling（Java之父）编写的《The Java Programming Language》一书的附录中给出了一个Java关键字列表，其中有goto和const，但是这两个是目前无法使用的关键字，因此有些地方将其称之为保留字，其实保留字这个词应该有更广泛的意义，因为熟悉C语言的程序员都知道，在系统类库中使用过的有特殊意义的单词或单词的组合都被视为保留字）

138. int 和Integer 有什么区别?

Java是一个近乎纯洁的面向对象编程语言，但是为了编程的方便还是引入不是对象的基本数据类型，但是为了能够将这些基本数据类型当成对象操作，Java为每一个基本数据类型都引入了对应的包装类型（wrapper class），int的包装类就是Integer，从JDK 1.5开始引入了自动装箱/拆箱机制，使得二者可以相互转换。

Java 为每个原始类型提供了包装类型：

原始类型: boolean，char，byte，short，int，long，float，double

包装类型：Boolean，Character，Byte，Short，Integer，Long，Float，Double

```
public static void main(String[] args) {
    Integer a = new Integer(3);
    Integer b = 3;                 // 将3自动装箱成Integer类型
    int c = 3;
    System.out.println(a == b); // false 两个引用没有引用同一对象
    System.out.println(a == c); // true a自动拆箱成int类型再和c比较
}
```

补充：最近还遇到一个面试题，也是和自动装箱和拆箱相关的，代码如下所示：

```
public static void main(String[] args) {
    Integer f1 = 100, f2 = 100, f3 = 150, f4 = 150;
    System.out.println(f1 == f2);
    System.out.println(f3 == f4);
}
```

如果不明就里很容易认为两个输出要么都是true要么都是false。首先需要注意的是f1、f2、f3、f4四个变量都是Integer对象，所以下面的==运算比较的不是值而是引用。装箱的本质是什么呢？当我们给一个Integer对象赋一个int值的时候，会调用Integer类的静态方法valueOf，如果看看valueOf的源代码就知道发生了什么。

```
public static Integer valueOf(int i) {
    if (i >= IntegerCache.low && i <= IntegerCache.high)
        return IntegerCache.cache[i + (-IntegerCache.low)];
    return new Integer(i);
}
```

IntegerCache是Integer的内部类，其代码如下所示：

```
/* Cache to support the object identity semantics of autoboxing for values between
     * -128 and 127 (inclusive) as required by JLS.
     *
     * The cache is initialized on first usage.  The size of the cache
     * may be controlled by the {@code -XX:AutoBoxCacheMax=<size>} option.
     * During VM initialization, java.lang.Integer.IntegerCache.high property
     * may be set and saved in the private system properties in the
     * sun.misc.VM class.
     */

private static class IntegerCache {
```

```
    static final int low = -128;
    static final int high;
    static final Integer cache[];

    static {
        // high value may be configured by property
        int h = 127;
        String integerCacheHighPropValue =
            sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");
        if (integerCacheHighPropValue != null) {
            try {
                int i = parseInt(integerCacheHighPropValue);
                i = Math.max(i, 127);
                // Maximum array size is Integer.MAX_VALUE
                h = Math.min(i, Integer.MAX_VALUE - (-low) -1);
            } catch( NumberFormatException nfe) {
                // If the property cannot be parsed into an int, ignore it.
            }
        }
        high = h;

        cache = new Integer[(high - low) + 1];
        int j = low;
        for(int k = 0; k < cache.length; k++)
            cache[k] = new Integer(j++);

        // range [-128, 127] must be interned (JLS7 5.1.7)
        assert IntegerCache.high >= 127;
    }

    private IntegerCache() {}
}
```


简单的说，如果字面量的值在-128到127之间，那么不会new新的Integer对象，而是直接引用常量池中的Integer对象，所以上面的面试题中f1,f2的结果是true，而f3==f4的结果是false。越是貌似简单的面试题其中的玄机就越多，需要面试者有相当深厚的功力。

---

139. &和&&的区别？

&运算符有两种用法：

(1)按位与；(2)逻辑与。

&&运算符是短路与运算。逻辑与跟短路与的差别是非常巨大的，虽然二者都要求运算符左右两端的布尔值都是true整个表达式的值才是true。&&之所以称为短路运算是因为，如果&&左边的表达式的值是false，右边的表达式会被直接短路掉，不会进行运算。很多时候我们可能都需要用&&而不是&，例如在验证用户登录时判定用户名不是null而且不是空字符串，应当写为：username != null &&!username.equals(“”)，二者的顺序不能交换，更不能用&运算符，因为第一个条件如果不成立，根本不能进行字符串的equals比较，否则会产生NullPointerException异常。

注意：逻辑或运算符（|）和短路或运算符（||）的差别也是如此。

*补充：*如果你熟悉JavaScript，那你可能更能感受到短路运算的强大，想成为JavaScript的高手就先从玩转短路运算开始吧。

140. Math.round(11.5) 等于多少? Math.round(-11.5)等于多少?

Math.round(11.5)的返回值是12，Math.round(-11.5)的返回值是-11。四舍五入的原理是在参数上加0.5然后进行下取整。

141~160

141. swtich 是否能作用在byte 上，是否能作用在long 上，是否能作用在String上?

早期的JDK中，switch（expr）中，expr可以是byte、short、char、int。从1.5版开始，Java中引入了枚举类型（enum），expr也可以是枚举，从JDK 1.7版开始，还可以是字符串（String）。长整型（long）是不可以的。

142. 用最有效率的方法计算2乘以8?

答：2 << 3（左移3位相当于乘以2的3次方，右移3位相当于除以2的3次方）。

补充：我们为编写的类重写hashCode方法时，可能会看到如下所示的代码，其实我们不太理解为什么要使用这样的乘法运算来产生哈希码（散列码），而且为什么这个数是个素数，为什么通常选择31这个数？前两个问题的答案你可以自己百度一下，选择31是因为可以用移位和减法运算来代替乘法，从而得到更好的性能。说到这里你可能已经想到了：31 * num <==> (num << 5) - num，左移5位相当于乘以2的5次方（32）再减去自身就相当于乘以31。现在的VM都能自动完成这个优化。

```
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + areaCode;
    result = prime * result
        + ((lineNumber == null) ? 0 : lineNumber.hashCode());
    result = prime * result + ((prefix == null) ? 0 : prefix.hashCode());
    return result;
}
```


143. 在Java 中，如何跳出当前的多重嵌套循环？

在最外层循环前加一个标记如A，然后用break A;可以跳出多重循环。（Java中支持带标签的break和continue语句，作用有点类似于C和C++中的goto语句，但是就像要避免使用goto一样，应该避免使用带标签的break和continue，因为它不会让你的程序变得更优雅，很多时候甚至有相反的作用，所以这种语法其实不知道更好）

144. 构造器（constructor）是否可被重写（override）?

构造器不能被继承，因此不能被重写，但可以被重载。

145. 两个对象值相同(x.equals(y) == true)，但却可有不同的hash code，这句话对不对？

答：不对，如果两个对象x和y满足x.equals(y) == true，它们的哈希码（hash code）应当相同。Java对于eqauls方法和hashCode方法是这样规定的：(1)如果两个对象相同（equals方法返回true），那么它们的hashCode值一定要相同；(2)如果两个对象的hashCode相同，它们并不一定相同。当然，你未必要按照要求去做，但是如果你违背了上述原则就会发现在使用容器时，相同的对象可以出现在Set集合中，同时增加新元素的效率会大大下降（对于使用哈希存储的系统，如果哈希码频繁的冲突将会造成存取性能急剧下降）。

补充：关于equals和hashCode方法，很多Java程序都知道，但很多人也就是仅仅知道而已，在Joshua Bloch的大作《Effective Java》（很多软件公司，《Effective Java》、《Java编程思想》以及《重构：改善既有代码质量》是Java程序员必看书籍，（如果你还没看过，那就赶紧去亚马逊买一本吧）中是这样介绍equals方法的：首先equals方法必须满足自反性（x.equals(x)必须返回true）、对称性（x.equals(y)返回true时，y.equals(x)也必须返回true）、传递性（x.equals(y)和y.equals(z)都返回true时，x.equals(z)也必须返回true）和一致性（当x和y引用的对象信息没有被修改时，多次调用x.equals(y)应该得到同样的返回值），而且对于任何非null值的引用x，x.equals(null)必须返回false。

实现高质量的equals方法的诀窍包括：

1. 使用==操作符检查“参数是否为这个对象的引用”；

2. 使用instanceof操作符检查“参数是否为正确的类型”；

3. 对于类中的关键属性，检查参数传入对象的属性是否与之相匹配；

4. 编写完equals方法后，问自己它是否满足对称性、传递性、一致性；

5. 重写equals时总是要重写hashCode；

6. 不要将equals方法参数中的Object对象替换为其他的类型，在重写时不要忘掉@Override注解。

---

146. 当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递?

是值传递。

Java 编程语言只有值传递参数。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的属性可以在被调用过程中被改变，但对象的引用是永远不会改变的。C++和C#中可以通过传引用或传输出参数来改变传入的参数的值。

补充：Java中没有传引用实在是非常的不方便，这一点在Java 8中仍然没有得到改进，正是如此在Java编写的代码中才会出现大量的包装类（将需要通过方法调用修改的引用置于一个Wrapper类中，再将Wrapper对象传入方法），这样的做法只会让代码变得臃肿，尤其是让从C和C++转型为Java程序员的开发者无法容忍。

---

147. 重载（Overload）和重写（Override）的区别。重载的方法能否根据返回类型进行区分?

Java的三大特征之一，多态，包括方法的多态和对象的多态；

方法的重载和重写都是实现多态的方式，区别在于前者实现的是编译时的多态性，而后者实现的是运行时的多态性。

重载（overload）发生在同一个类中，相同的方法，如果有不同的参数列表（参数类型不同、参数个数不同或者二者都不同）则视为重载；

重写（override）发生在子类与父类之间也就是继承机制当中，当父类的方法不能满足子类的要求，此时子类重写父类的方法；

要求：方法名、形参列表相同；返回值类型和异常类型，子类小于等于父类；访问权限，子类大于等于父类，切记父类的私有方法以及被final修饰的方法不能被子类重写；重载对返回类型没有特殊的要求。

---

148. 华为的面试题中曾经问过这样一个问题：为什么不能根据返回类型来区分重载，为什么？

方法的重载，即使返回值类型不同，也不能改变实现功能相同或类似这一既定事实；同时方法的重载只是要求两同三不同，即在同一个类中，相同的方法名称，参数列表当中的参数类型、个数、顺序不同；跟权限修饰符和返回值类无关

---

149. 静态嵌套类(Static Nested Class)和内部类（Inner Class）的不同？

内部类就是在一个类的内部定义的类，内部类中不能定义静态成员（静态成员不是对象的特性，只是为了找一个容身之处，所以需要放到一个类中而已，这么一点小事，你还要把它放到类内部的一个类中，过分了啊！提供内部类，不是为让你干这种事情，无聊，不让你干。我想可能是既然静态成员类似c语言的全局变量，而内部类通常是用于创建内部对象用的，所以，把"全局变量"放在内部类中就是毫无意义的事情，既然是毫无意义的事情，就应该被禁止），内部类可以直接访问外部类中的成员变量，内部类可以定义在外部类的方法外面，也可以定义在外部类的方法体中，

```
public class Outer
{
    int out_x  = 0;
    public void method()
    {
        Inner1 inner1 = new Inner1();
        public class Inner2  //在方法体内部定义的内部类
        {
            public method()
            {
                out_x = 3;
            }
        }
        Inner2 inner2 = new Inner2();
    }

    public class Inner1  //在方法体外面定义的内部类
    {
    }

}
```

在方法体外面定义的内部类的访问类型可以是public,protecte,默认的，private等4种类型，这就好像类中定义的成员变量有4种访问类型一样，它们决定这个内部类的定义对其他类是否可见；对于这种情况，我们也可以在外面创建内部类的实例对象，创建内部类的实例对象时，一定要先创建外部类的实例对象，然后用这个外部类的实例对象去创建内部类的实例对象，代码如下：

```
Outer outer = new Outer();
Outer.Inner1 inner1 = outer.new Innner1();
```

在方法内部定义的内部类前面不能有访问类型修饰符，就好像方法中定义的局部变量一样，但这种内部类的前面可以使用final或abstract修饰符。这种内部类对其他类是不可见的其他类无法引用这种内部类，但是这种内部类创建的实例对象可以传递给其他类访问。这种内部类必须是先定义，后使用，即内部类的定义代码必须出现在使用该类之前，这与方法中的局部变量必须先定义后使用的道理也是一样的。这种内部类可以访问方法体中的局部变量，但是，该局部变量前必须加final修饰符。

对于这些细节，只要在eclipse写代码试试，根据开发工具提示的各类错误信息就可以马上了解到。

在方法体内部还可以采用如下语法来创建一种匿名内部类，即定义某一接口或类的子类的同时，还创建了该子类的实例对象，无需为该子类定义名称：

```
public class Outer
{
    public void start()
    {
        new Thread(
            new Runable(){
                public void run(){};
            }
        ).start();
    }
}
```

最后，在方法外部定义的内部类前面可以加上static关键字，从而成为Static Nested Class，它不再具有内部类的特性，所有，从狭义上讲，它不是内部类。Static Nested Class与普通类在运行时的行为和功能上没有什么区别，只是在编程引用时的语法上有一些差别，它可以定义成public、protected、默认的、private等多种类型，而普通类只能定义成public和默认的这两种类型。在外面引用Static Nested Class类的名称为“外部类名.内部类名”。在外面不需要创建外部类的实例对象，就可以直接创建Static Nested Class，例如，假设Inner是定义在Outer类中的Static Nested Class，那么可以使用如下语句创建Inner类：

```
Outer.Inner inner = newOuter.Inner();
```

由于static Nested Class不依赖于外部类的实例对象，所以，**static Nested Class能访问外部类的非static成员变量(不能直接访问，需要创建外部类实例才能访问非静态变量)。**当在外部类中访问Static Nested Class时，可以直接使用Static Nested Class的名字，而不需要加上外部类的名字了，在Static Nested Class中也可以直接引用外部类的static的成员变量，不需要加上外部类的名字。

在静态方法中定义的内部类也是Static Nested Class，这时候不能在类前面加static关键字，静态方法中的Static Nested Class与普通方法中的内部类的应用方式很相似，它除了可以直接访问外部类中的static的成员变量，还可以访问静态方法中的局部变量，但是，该局部变量前必须加final修饰符。

备注：首先根据你的印象说出你对内部类的总体方面的特点：例如，在两个地方可以定义，可以访问外部类的成员变量，不能定义静态成员，这是大的特点。然后再说一些细节方面的知识，例如，几种定义方式的语法区别，静态内部类，以及匿名内部类。

Static Nested Class是被声明为静态（static）的内部类，它可以不依赖于外部类实例被实例化。而通常的内部类需要在外部类实例化后才能实例化，其语法看起来挺诡异的，如下所示。

```
/**
* 扑克类（一副扑克）
 */
```

```java
public class Poker {
    private static String[] suites = {"黑桃", "红桃", "草花", "方块"};
    private static int[] faces = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
    private Card[] cards;
    /**
      * 构造器
     */
    public Poker() {
        cards = new Card[52];
        for(int i = 0; i < suites.length; i++) {
            for(int j = 0; j < faces.length; j++) {
                cards[i * 13 + j] = new Card(suites[i], faces[j]);
            }
        }
    }

    /**
          * 洗牌 （随机乱序）
          */
    public void shuffle() {
        for(int i = 0, len = cards.length; i < len; i++) {
            int index = (int) (Math.random() * len);
            Card temp = cards[index];
            cards[index] = cards[i];
            cards[i] = temp;
        }
    }
    /**
     * 发牌
     * @param index 发牌的位置
     */
    public Card deal(int index) {
        return cards[index];
    }

    /**
     * 卡片类（一张扑克）
     * [内部类]
     * @author sxt
     */
    public class Card {
        private String suite;   // 花色
        private int face;       // 点数
        public Card(String suite, int face) {
            this.suite = suite;
            this.face = face;
        }
        @Override
        public String toString() {
            String faceStr = "";
            switch(face) {
                case 1: faceStr = "A"; break;
                case 11: faceStr = "J"; break;
```

```
                case 12: faceStr = "Q"; break;
                case 13: faceStr = "K"; break;
                default: faceStr = String.valueOf(face);
            }
            return suite + faceStr;
        }
    }
}
```

测试类:

```
class PokerTest {
    public static void main(String[] args) {
        Poker poker = new Poker();
        poker.shuffle();            // 洗牌
        Poker.Card c1 = poker.deal(0);  // 发第一张牌
        // 对于非静态内部类Card
        // 只有通过其外部类Poker对象才能创建Card对象
        Poker.Card c2 = poker.new Card("红心", 1);    // 自己创建一张牌
        System.out.println(c1);     // 洗牌后的第一张
        System.out.println(c2);     // 打印: 红心A
    }
}
```


150. 抽象的（abstract）方法是否可同时是静态的（static）,是否可同时是本地方法（native），是否可同时被synchronized修饰?

都不能。

抽象方法需要子类重写，而静态的方法是无法被重写的，因此二者是矛盾的。本地方法是由本地代码（如C代码）实现的方法，而抽象方法是没有实现的，也是矛盾的。synchronized和方法的实现细节有关，抽象方法不涉及实现细节，因此也是相互矛盾的。

151. 静态变量和实例变量的区别？

静态变量是被static修饰符修饰的变量，也称为类变量，它属于类，不属于类的任何一个对象，一个类不管创建多少个对象，静态变量在内存中有且仅有一个拷贝；实例变量必须依存于某一实例，需要先创建对象然后通过对象才能访问到它，静态变量可以实现让多个对象共享内存。

两者的相同点：都有默认值而且在类的任何地方都可以调用。在Java开发中，上下文类和工具类中通常会有大量的静态成员。

152. 是否可以从一个静态（static）方法内部发出对非静态（non-static）方法的调用？

不可以，静态方法只能访问静态成员，因为非静态方法的调用要先创建对象，因此在调用静态方法时可能对象并没有被初始化。

153. 如何实现对象克隆？

有两种方式：

1.实现Cloneable接口并重写Object类中的clone()方法；

2.实现Serializable接口，通过对象的序列化和反序列化实现克隆，可以实现真正的深度克隆。

注意：基于序列化和反序列化实现的克隆不仅仅是深度克隆，更重要的是通过泛型限定，可以检查出要克隆的对象是否支持序列化，这项检查是编译器完成的，不是在运行时抛出异常，这种是方案明显优于使用Object类的clone方法克隆对象。

---

154. 接口是否可继承（extends）接口? 抽象类是否可实现（implements）接口? 抽象类是否可继承具体类（concrete class）?

接口可以继承接口。抽象类可以实现(implements)接口，抽象类可以继承具体类。抽象类中可以有静态的main方法。

备注：只要明白了接口和抽象类的本质和作用，这些问题都很好回答，你想想，如果你是java语言的设计者，你是否会提供这样的支持，如果不提供的话，有什么理由吗？如果你没有道理不提供，那答案就是肯定的了。

只有记住抽象类与普通类的唯一区别就是不能创建实例对象和允许有abstract方法。

---

155. 一个".java"源文件中是否可以包含多个类（不是内部类）？有什么限制？

可以，但一个源文件中最多只能有一个公开类（public class）而且文件名必须和公开类的类名完全保持一致。

---

156. Anonymous Inner Class(匿名内部类)是否可以继承其它类？是否可以实现接口？

可以继承其他类或实现其他接口，在Swing编程中常用此方式来实现事件监听和回调。 但是有一点需要注意，它只能继承一个类或一个接口。

---

157. 内部类可以引用它的包含类（外部类）的成员吗？有没有什么限制？

一个内部类对象可以访问创建它的外部类对象的成员，包括私有成员。

如果要访问外部类的局部变量，此时局部变量必须使用final修饰，否则无法访问。

---

158. Java 中的final关键字有哪些用法？

(1)修饰类：表示该类不能被继承；

(2)修饰方法：表示方法不能被重写但是允许重载；

(3)修饰变量：表示变量只能一次赋值以后值不能被修改（常量）；

(4)修饰对象：对象的引用地址不能变，但是对象的初始化值可以变。

---

159. 指出下面程序的运行结果

```
1 /*
```

```
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class Test {
    public static void main(String[] args) {
        A ab = new B();
        ab = new B();
    }
}

class A{
    static{
        System.out.print("1");
    }
    public A(){
        System.out.print("2");
    }
}
class B extends A{
    static{
        System.out.print("a");
    }
    public B(){
        System.out.print("b");
    }
}
```

结果 : 1a2b2b

静态代码块只执行一次 , 在类初始化的时候 ,

构造方法会向上执行父级的 .

---

160. 说说数据类型之间的转换

1)如何将字符串转换为基本数据类型？

2)如何将基本数据类型转换为字符串？

答：

1)调用基本数据类型对应的包装类中的方法parseXXX(String)或valueOf(String)即可返回相应基本类型；

2)一种方法是将基本数据类型与空字符串（“”）连接（+）即可获得其所对应的字符串；另一种方法是调用String 类中的valueOf(...)方法返回相应字符串

161~169

161. 如何实现字符串的反转及替换？

方法很多，可以自己写实现也可以使用String或StringBuffer / StringBuilder中的方法。有一道很常见的面试题是用递归实现字符串反转，代码如下所示：

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

string.substring(from)：此时相当于从from位置截取到原字符串末尾

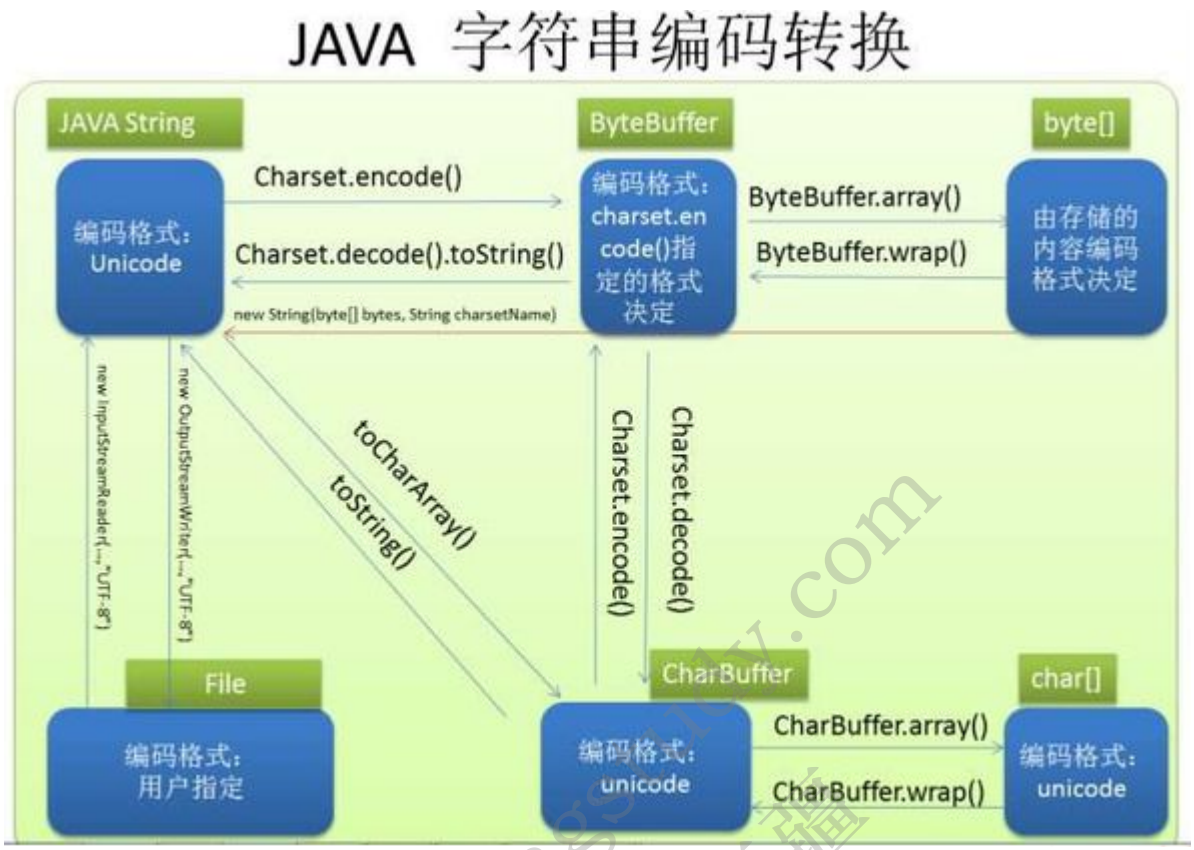
charAt() 方法用于返回指定索引处的字符。索引范围为从 0 到 length() - 1。

---

162. 怎样将GB2312编码的字符串转换为ISO-8859-1编码的字符串？

通过JDK1.6知道String类中getBytes（”编码“）方法可以将一个数用指定的编码转成一个字节数组，String中通过指定的 charset解码指定的 byte 数组，构造一个新的 String。代码如下：

```java
try{
    String s = "java学习";
    System.out.println(s);

    String result = new String(s.getBytes("GB2312"),"iso-8859-1");

    System.out.println(s);

} catch (UnsupportedEncodingException e) {

    // TODO Auto-generated catch block

    e.printStackTrace();

}
```



getBytes() 方法有两种形式：

1、getBytes(String charsetName): 使用指定的字符集将字符串编码为 byte 序列，并将结果存储到一个新的 byte 数组中。

2、getBytes(): 使用平台的默认字符集将字符串编码为 byte 序列，并将结果存储到一个新的 byte 数组中。

---

163. Java中的日期和时间

1)如何取得年月日、小时分钟秒？

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class Test {
    public static void main(String[] args) throws ParseException {

        Calendar now = Calendar.getInstance();
        System.out.println("年: " + now.get(Calendar.YEAR));
        System.out.println("月: " + (now.get(Calendar.MONTH) + 1) + "");
        System.out.println("日: " + now.get(Calendar.DAY_OF_MONTH));
        System.out.println("时: " + now.get(Calendar.HOUR_OF_DAY));
        System.out.println("分: " + now.get(Calendar.MINUTE));
```

```
        System.out.println("秒: " + now.get(Calendar.SECOND));
        System.out.println("当前时间毫秒数：" + now.getTimeInMillis());
        System.out.println(now.getTime());

        Date d = new Date();
        System.out.println(d);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        String dateNowStr = sdf.format(d);
        System.out.println("格式化后的日期：" + dateNowStr);

        String str = "2012-1-13 17:26:33"; //要跟上面sdf定义的格式一样
        Date today = sdf.parse(str);
        System.out.println("字符串转成日期：" + today);

    }
}
```

2)如何取得从1970年1月1日0时0分0秒到现在的毫秒数？

```
Calendar.getInstance().getTimeInMillis();
System.currentTimeMillis();
```

3)如何取得某月的最后一天？

```
Calendar time = Calendar.getInstance();
time.getActualMaximum(Calendar.DAY_OF_MONTH);
```

4)如何格式化日期？

利用java.text.DataFormat 的子类（如SimpleDateFormat类）中的format(Date)方法可将日期格式化。

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
String dateNowStr = sdf.format(d);
System.out.println("格式化后的日期：" + dateNowStr);

String str = "2012-1-13 17:26:33"; //要跟上面sdf定义的格式一样
Date today = sdf.parse(str);
System.out.println("字符串转成日期：" + today);
```


---

164. 打印昨天的当前时刻

```java
/*
 * @Description QQ:24736743  微信:qinl0vejiang  狂神编程群:851857656
 * @Author 狂神说
 **/
public class Test {
    public static void main(String[] args) throws ParseException {

        Calendar cal = Calendar.getInstance();
        cal.add(Calendar.DATE, -1);
        System.out.println(cal.getTime());

    }
}
```

165. Java反射技术主要实现类有哪些，作用分别是什么？

在JDK中，主要由以下类来实现Java反射机制，这些类都位于java.lang.reflect包中

1）Class类：代表一个类

2）Field 类：代表类的成员变量(属性)

3）Method类：代表类的成员方法

4）Constructor 类：代表类的构造方法

5）Array类：提供了动态创建数组，以及访问数组的元素的静态方法

166. Class类的作用？生成Class对象的方法有哪些？

Class类是Java 反射机制的起源和入口，用于获取与类相关的各种信息，提供了获取类信息的相关方法。Class类继承自Object类

Class类是所有类的共同的图纸。每个类有自己的对象，好比图纸和实物的关系；每个类也可看做是一个对象，有共同的图纸Class，存放类的 结构信息，能够通过相应方法取出相应信息：类的名字、属性、方法、构造方法、父类和接口

方 法	示 例
对象名 .getClass()	String str="bdqn"; Class clazz = str.getClass();
对象名 .getSuperClass()	Student stu = new Student(); Class c1 = stu.getClass(); Class c2 = stu.getSuperClass();
Class.forName()	Class clazz = Class.forName("java.lang.Object"); Class.forName("oracle.jdbc.driver.OracleDriver");
类名.class	Class c1 = String.class; Class c2 = Student.class; Class c2 = int.class
包装类.TYPE	Class c1 = Integer.TYPE; Class c2 = Boolean.TYPE;

167. 反射的使用场合和作用、及其优缺点

1）使用场合：

在编译时根本无法知道该对象或类可能属于哪些类，程序只依靠运行时信息来发现该对象和类的真实信息。

2）主要作用：

通过反射可以使程序代码访问装载到JVM 中的类的内部信息，获取已装载类的属性信息，获取已装载类的方法，获取已装载类的构造方法信息

3）反射的优点

反射提高了Java程序的灵活性和扩展性，降低耦合性，提高自适应能力。它允许程序创建和控制任何类的对象，无需提前硬编码目标类；反射是其它一些常用语言，如C、C++、Fortran 或者Pascal等都不具备的

4）Java反射技术应用领域很广，如软件测试等；许多流行的开源框架例如Struts、Hibernate、Spring在实现过程中都采用了该技术

5）反射的缺点

性能问题：使用反射基本上是一种解释操作，用于字段和方法接入时要远慢于直接代码。因此Java反射机制主要应用在对灵活性和扩展性要求很高的系统框架上,普通程序不建议使用。

使用反射会模糊程序内部逻辑：程序人员希望在源代码中看到程序的逻辑，反射等绕过了源代码的技术，因而会带来维护问题。反射代码比相应的直接代码更复杂。

---

168. 面向对象设计原则有哪些

面向对象设计原则是面向对象设计的基石，面向对象设计质量的依据和保障，设计模式是面向对象设计原则的经典应用

1）单一职责原则SRP

2）开闭原则OCP

3）里氏替代原则LSP

4）依赖注入原则DIP

5）接口分离原则ISP

6）迪米特原则LOD

7）组合/聚合复用原则CARP

8）开闭原则具有理想主义的色彩，它是面向对象设计的终极目标。其他设计原则都可以看作是开闭原则的实现手段或方法

---

169. char型变量中能不能存储一个中文汉字？

1.java采用unicode编码，2个字节（16位）来表示一个字符， 无论是汉字还是数字，字母，或其他语言都可以存储。

2.char 在java中是2个字节，所以可以存储中文